

Terminaison en temps moyen fini de systèmes de règles probabilistes

THÈSE

présentée et soutenue publiquement le 17 Septembre 2007

pour l'obtention du

Doctorat de l'Institut National Polytechnique de Lorraine
(spécialité informatique)

par

Florent Garnier

Composition du jury

<i>Président :</i>	Jean-François Monin	Professeur, Université Joseph Fourier Grenoble I
<i>Rapporteurs :</i>	Catuscia Palamidessi Laurent Fribourg	Directeur de Recherche, INRIA Futurs Saclay et LIX Directeur de Recherche, CNRS, LSV ENS Cachan
<i>Examineurs :</i>	Ye Qiong Song Claude Kirchner Olivier Bournez Jean-François Monin Francis Klay	Professeur, INPL Directeur de Recherche, INRIA et LORIA Chargé de Recherche, INRIA Professeur, Université Joseph Fourier Grenoble I Ingénieur de Recherche, Orange France Télécom R&D

Mis en page avec la classe thloria.

Remerciements

Je tiens à remercier en premier lieu mon encadrant Olivier Bournez qui en plus de m'avoir proposé ce sujet de thèse, s'est toujours montré disponible pour me venir en aide et m'apporter ses conseils précieux. Je remercie également mon directeur de thèse Claude Kirchner pour sa patience, ses grandes qualités pédagogiques ainsi que sa gentillesse.

Je remercie chaleureusement les personnes qui ont accepté d'être membres de mon jury, en particulier Catuscia Palamidessi et Laurent Fribourg qui ont accepté d'être mes rapporteurs.

Je ne remerciais jamais assez mes amis et collègues, Emmanuel, Germain, Antoine qui ont facilité mon arrivée à Nancy et qui m'ont appuyé moralement et parfois matériellement durant toute la durée de ma thèse. Je remercie également tous les membres de l'équipe PROTHEO avec qui j'ai passé des moments forts agréables dans le cadre de la recherche et parfois en dehors.

Merci à mes parents ainsi et à mes deux frères Sylvain et Ghislain pour leur appui moral. Je remercie aussi Gérôme pour m'avoir transmis la passion du cyclisme à la montagne. Et merci encore à tous mes amis que je n'ai pas cités ici, mais qui mériteraient de l'être.

*J'ai rarement eu des
journées aussi longues
que durant ma thèse.
En effet, à chaque fois
que j'ai promis quelque chose
pour le lendemain, il m'a
fallu six mois pour le faire.*

Table des matières

Introduction	1
--------------	---

Partie I Notions préliminaires 7

Chapitre 1
Présentation de la réécriture et de la réécriture sous stratégie

1.1	Relations binaires et systèmes de réduction abstraits	9
1.2	Le principe de l'induction bien fondée	11
1.3	Terminaison des systèmes abstraits de réduction	12
1.4	Les algèbres de termes	14
1.4.1	Substitutions et relation de réduction sur les termes	16
1.4.2	Σ -identités et relations de réduction associées	17
1.5	Les systèmes de réécriture	18
1.6	Terminaison des systèmes de réécriture	19
1.6.1	Les ordres de réduction	20
1.6.2	Méthode d'interprétation	21
1.6.3	Les ordres de simplification	24
1.7	Les stratégies de réécriture	24
1.7.1	Présentation de la notion de stratégie de réécriture	24

Chapitre 2
Les modèles de systèmes probabilistes

2.1	Rappels sur la théorie des probabilités	27
2.1.1	Phénomènes aléatoires	27
2.1.2	Propriétés importantes des probabilités	29
2.1.3	Espérance d'une variable aléatoire	30

2.1.4	Convergence d'une suite de variables aléatoires	31
2.1.5	Probabilité conditionnelle et espérance conditionnelle . . .	33
2.2	Martingales, surmartingales et sous martingales	35
2.2.1	Martingales	35
2.3	Chaînes de Markov	37
2.3.1	Chaînes de Markov à espace discret et à temps discret . .	37
2.3.2	Comportement asymptotique des chaînes de Markov . . .	38
2.3.3	Récurrence et transience	40
2.3.4	Convergence en loi des chaînes de Markov	42
2.3.5	Exemples d'applications	42
2.3.6	Temps moyen d'atteinte d'un intervalle pour une sur-martingale	43
2.4	Processus de décision Markovien	44
2.4.1	Exécutions	45
2.4.2	Ensembles mesurables d'exécutions	46
2.4.3	Politiques	46
2.5	Problèmes classiques	47
2.5.1	Le problème du plus court chemin stochastique (SSP) . . .	47
2.5.2	Méthode de résolution de SSP	48
2.6	Vérification de propriétés sur des modèles probabilistes	49
2.6.1	Méthodes formelles	49
2.6.2	La vérification dans le cas général	50
2.6.3	Quelques propriétés importantes	50
2.6.4	Aperçus des méthodes de vérifications de propriétés sur les Processus de décision Markoviens	51
2.6.5	Logique probabiliste arborescente	52
2.6.6	Model checking de pCTL	53
2.7	Logiciels de model-checking probabiliste	53

Partie II Réécriture probabiliste

55

Chapitre 3

Les Systèmes Abstraites de Réduction Probabilistes

3.1	Présentation du modèle	57
3.2	Terminaison des Systèmes Abstraites de Réduction Probabiliste . .	61

3.3	Prouver la terminaison presque sûre positive	63
3.4	En résumé	69

Chapitre 4 Les systèmes de réécriture probabilistes

4.1	Introduction	71
4.2	Définition du modèle	71
4.3	Terminaison des systèmes de réécriture probabilistes	78
4.4	Critères de terminaison presque sûre positive	80
4.5	Application à quelques exemples simples	81
4.6	Preuve de terminaison grâce à une méthode d'interprétation polynomiale.	83
4.7	Résumé du chapitre	85

Chapitre 5 La terminaison presque sûre positive sous stratégies

5.1	Introduction	87
5.2	Stratégies et terminaison sous stratégie	87
5.3	Terminaison presque sûre sous stratégie.	92
5.3.1	Généralisation	93
5.4	Applications	99

Partie III Exemple d'application 101

Chapitre 6 Produit synchronisé d'automates temporisés

6.1	Les Automates temporisés à la Alur et Dill	103
6.1.1	Séquence temporelle	103
6.1.2	Contraintes et opérations sur les horloges	104
6.2	Composition parallèle.	107
6.2.1	Comportements non déterministes.	113
6.3	Produit synchronisé d'automates avec variables	117
6.3.1	Automates temporisés avec variables	118
6.3.2	Produit synchronisé via une ressource de synchronisation	120

6.3.3 Représentation graphique 122

<p>Chapitre 7 Terminaison en temps moyen fini du protocole 802.11b CSMA/CA</p>

7.1 Description du protocole CSMA/CA 124

7.2 Modélisation du système 128

7.3 Implantation d'un simulateur. 131

 7.3.1 Algorithme de simulation du produit d'automates. 133

 7.3.2 Transition d'un automate 136

 7.3.3 Synchronisation et simulation de tout le système 139

7.4 La Simulation du CSMA-CA termine +.a.s. 145

 7.4.1 Construction d'un certificat de terminaison. 145

7.5 Conclusion 155

 7.5.1 En résumé 155

<p>Chapitre 8 Codage des systèmes de réécriture probabilistes</p>

8.1 Exprimer des systèmes de réécriture avec TOM 160

 8.1.1 Définir l'espace des termes 160

 8.1.2 Gestion des termes par TOM 161

 8.1.3 Manipuler les termes avec TOM 161

 8.1.4 Les stratégies de réécriture sous TOM 163

8.2 Codage des systèmes de réécriture probabilistes avec TOM 164

 8.2.1 Écriture des règles de réécriture probabilistes 164

 8.2.2 Appliquer les règles de réécriture probabilistes sous stratégie 167

8.3 Implémentation de quelques exemples 167

 8.3.1 Marche aléatoire 167

 8.3.2 Réécriture sous stratégie aléatoire Markovienne 169

 8.3.3 Réécriture sous une stratégie aléatoire non Markovienne . . 169

8.4 Conclusion 170

<p>Conclusion et perspectives</p>

Index **175**

Table des figures **179**

Introduction

Motivations

L'origine de ce travail réside dans la volonté que nous avons de définir un formalisme inspiré par la réécriture, permettant de spécifier des programmes où se combinent des phénomènes probabilistes et des phénomènes non déterministes. Ce formalisme devait également permettre de pouvoir étudier certaines propriétés probabilistes que vérifieraient les systèmes ainsi représentés. Il devait également être assez simple pour pouvoir représenter facilement divers objets et permettre qu'il soit implémentable. En effet, nos partenaires industriels avaient exprimé le besoin de pouvoir utiliser des outils de modélisation permettant d'une part de représenter certains protocoles réseau où des tirages probabilistes conditionnent les exécutions et d'autre part de permettre de vérifier si ces algorithmes accèdent en temps moyen fini à des états modélisant le succès de l'algorithme ou à des états représentant une panne.

Il existait également le désir d'adjoindre à la réécriture un mécanisme permettant d'exprimer des phénomènes probabilistes. Un travail de prospection avait déjà été réalisé dans ce sens avant le début de cette thèse [BK02, BH03], dans lequel les auteurs définissaient une notion de stratégie de réécriture probabiliste comme des algorithme appliquant des règles de réécriture classique suivant une loi de probabilité.

Il existait cependant déjà beaucoup de modèles de représentation de systèmes probabilistes où interviennent des phénomènes non-déterministes, souvent accompagnés de méthodes pour prouver des propriétés probabilistes intéressantes autant du point de vue esthétique et théorique que du point de vue applicatif. En effet, plusieurs méthodes de preuves d'accessibilité d'états pour les processus de décision Markoviens [Put94, Der70] avaient déjà permis de mettre en place des logiciels de vérification de modèles [KNP02, Pey03] et ceux-ci avaient déjà été utilisés pour vérifier la fiabilité de certains systèmes et des propriétés de qualité de service sur des algorithmes de télécommunication [KNS02].

Bien que le domaine d'étude des systèmes probabilistes soit déjà bien exploré, il manquait un formalisme inspiré par la réécriture permettant d'exprimer de façon concise des relations complexes sur un espace d'états non fini mais dénombrable, sur lequel il soit possible de traiter le problème de la terminaison ou des problèmes d'accessibilité de classes d'états. C'est pour contribuer à cette problématique qu'est venue

l'idée d'étendre le formalisme de la réécriture qui dans sa version classique permet de définir des relations complexes sur des ensembles de termes infinis mais dénombrables, et cela avec l'aide d'un nombre fini de règles de réécriture. Il suffisait alors de modifier la notion de règle de réécriture pour lui permettre d'exprimer un choix de successeur suivant une loi de probabilité ou bien de définir des « stratégies » choisissant suivant une loi de probabilité paramétrable. La réécriture présente surtout l'avantage d'être un domaine étudié depuis longtemps et sur lequel il existe de nombreuses méthodes de preuves concernant les résultats de terminaison qui nous intéressaient.

Nos partenaires industriels, soucieux de fournir des algorithmes satisfaisant des critères de qualité de service, s'intéressent tout particulièrement à ces questions d'accessibilité de classes d'états en temps moyen fini et de terminaison en temps moyen fini pour leurs algorithmes probabilistes. Il existait certes déjà des outils de vérification automatique de ce type de propriétés pour une instance donnée d'un problème, fournie notamment par la communauté « model-checking », mais nos partenaires exprimaient le besoin d'utiliser des méthodes formelles permettant de valider leur systèmes quelles que soient les instances.

Approche

Nous allons présenter par la suite, la manière avec laquelle nous nous sommes acquittés de ces différentes tâches. Notre approche consiste à spécifier des systèmes de façon formelle et vérifier des propriétés probabilistes sur ces systèmes :

- En ce qui concerne la spécification, nous décrirons le modèle de système de réécriture probabiliste que nous avons développé durant cette thèse, ainsi que la notion de stratégie pour les systèmes de réécriture probabilistes.
- Nous présenterons également nos méthodes de preuves permettant d'effectuer la vérification de la terminaison en temps moyen fini des systèmes de réécriture probabilistes ainsi que la terminaison en temps moyen fini de ces systèmes sous stratégie.
- Nous mettrons en évidence les possibilités d'application de nos systèmes en étudiant la terminaison en temps moyen fini d'un algorithme de télécommunication.

Modéliser avec des règles

Classiquement, la réécriture est un formalisme permettant de modéliser des systèmes divers et variés. On peut citer à titre d'exemples les machines de Turing, les réseaux de Pétri, les automates de Müller et de Buchi et les automates d'arbres. La réécriture est utilisée dans de nombreux logiciels, par exemple dans les logiciels Maple pour le calcul des bases de Groebner [Buc76] et les manipulations sur les polynômes qu'elles permettent. Il ne faut pas omettre de mentionner les logiciels de modélisation

de systèmes à base de règles de réécriture tels ELAN [BKK⁺98], TOM [MRV03], OBJ [GWM⁺93], Maude [CM00] et CiME [CMMU].

Nous voulions adjoindre à la réécriture classique le fait de pouvoir exprimer des transitions probabilistes. En effet la réécriture ne permet pas de représenter des processus comme les chaînes de Markov, les processus de décision Markovien où les algèbres de processus probabilistes. On a retenu deux méthodes permettant d'exprimer des choix de successeurs suivant des lois de probabilité.

- La première consiste à conserver le formalisme de la réécriture classique et d'y adjoindre un opérateur de choix probabiliste qui sélectionne une règle de réécriture parmi toutes celles qui peuvent s'appliquer, suivant une loi de probabilité passée en paramètre de l'opérateur,
- La seconde consiste à définir des règles de réécritures de la forme suivante :

$$l_1 \rightarrow \delta_1$$

où l_1 est un terme et δ_1 est une distribution sur les termes.

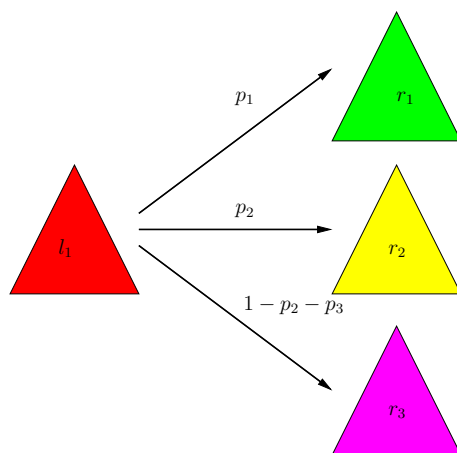


FIG. 1 – Relation de transition probabiliste

Pour donner une intuition de ce que nous voulons faire, considérons l'exemple représenté par la figure 1. On va exprimer pour chaque méthode une manière d'exprimer la transition probabiliste représentée par le graphique. Appelons le terme de gauche l_1 , puis les termes de droite, numérotés de haut en bas, r_1, r_2 et r_3 .

Dans le premier cas de figure, on définit un système de réécriture avec trois règles $R_1 := l_1 \rightarrow r_1$, $R_2 := l_1 \rightarrow r_2$ et $R_3 := l_1 \rightarrow r_3$. Puis, on définit un algorithme qui applique la règle R_1 avec probabilité p_1 , R_2 avec probabilité p_2 et R_3 avec probabilité $1 - p_1 - p_3$.

Dans le second cas, on représente cette transition grâce à ce que l'on définira

comme une règle de réécriture probabiliste, de la façon suivante :

$$l_1 \rightarrow \begin{cases} r_1 : p_1 \\ r_2 : p_2 \\ r_3 : p_3 \end{cases}$$

On peut utiliser les deux notions simultanément. En effet, on peut appliquer des règles de la forme $l \rightarrow \delta$ suivant une stratégie qui contrôle l'application de ces règles suivant un algorithme ou choisit la règle à appliquer en suivant une distribution de probabilité sur l'ensemble des règles pouvant être appliquées. Ce type de combinaison entre règles probabilistes et stratégies est utilisé pour décrire le comportement des processus de décision Markoviens [Put94], où les stratégies sont appelées politiques.

Nous avons donc dans un premier temps défini un modèle de système de réécriture probabiliste [BG05], composé de règles de la forme $l \rightarrow \delta$. Nous avons également eu besoin d'adapter les stratégies afin d'étudier le comportement des dérivations de ces systèmes.

Contrôler l'application des règles avec des stratégies

La notion de stratégie que nous introduisons s'inspire fortement des politiques présentées dans [Der70] et proches des adversaires de Segala et Lynch [SL95] et les ordonnanceurs de Lehman et Rabin [LR81], Vardi [Var85] et Pnueli et Zuck [PZ86].

L'exemple présenté par la figure 1 décrit un cas de figure où on choisit un successeur au terme de gauche suivant une distribution de probabilité bien déterminée. Néanmoins, il existe des cas de figure où plusieurs distributions de probabilités peuvent décrire le choix du successeur d'un terme donné. C'est pour résoudre ce cas de figure que nous avons défini un modèle de « stratégie » s'inspirant fortement des modèles décrits dans [Der70, SL95, LR81, Var85, PZ86].

La notion de stratégie nous a intéressé car elle permet d'étudier le comportement des programmes à base de règles probabilistes comme des processus stochastiques. Les stratégies résolvent les choix non déterministes suivant un algorithme déterministe ou suivant un tirage probabiliste. Cette propriété des stratégies permet d'étudier le problème de la terminaison de programmes à base de règles probabilistes et de définir des critères permettant de certifier la terminaison en temps moyen fini de ces mêmes programmes. Les stratégies ne sont pas à nos yeux qu'un simple outil technique que nous utilisons pour prouver certaines propriétés. Elles permettent en effet d'écrire des programmes ou des algorithmes probabilistes en sélectionnant des règles de réécriture en fonction des termes parcourus par le passé.

Preuve formelle de terminaison en temps moyen fini

Les principales propriétés sur lesquelles nous avons travaillé et obtenu des résultats significatifs concernent la terminaison en temps moyen fini des systèmes de réécriture probabilistes [BG05] et la terminaison de systèmes de réécriture probabilistes sous certaines classes de stratégies [BG06]. Cette étude généralise au cas probabiliste des travaux concernant la terminaison des systèmes de réécriture sous stratégie [FGK02b, FGK02a, AG97]. Comme nous le mentionnions dans le paragraphe précédent, nous avons défini une notion de stratégie inspirée des « politiques » utilisée pour étudier le comportement des processus de décision Markoviens et autres systèmes apparentés. Pour prouver la terminaison de ces systèmes, nous utilisons des outils mathématiques appelés « sur-martingales ». Grâce à certains théorèmes de convergence des « sur-martingales » nous avons pu trouver des conditions suffisantes, et nécessaires et suffisantes dans certains cas de systèmes de réécriture, impliquant la terminaison en temps moyen fini de ces systèmes dans le cas général et dans le cas où le choix d'application des règles est conditionné par une stratégie de réécriture.

Applications de ce modèle à un cas d'étude pratique

Afin d'étudier l'intérêt pratique du formalisme que nous avons développé, nous avons modélisé l'algorithme du « CSMA/CA 802.11b » utilisé par les réseaux sans-fil « WIFI » [con]. Pour faire cela nous avons d'une part étudié un formalisme permettant de modéliser la synchronisation de plusieurs automates temporisés par passage de messages sur un support modélisant les contraintes d'accessibilité entre les différentes parties du réseau. Une fois cela fait, nous exprimons un tel produit synchronisé grâce à un ensemble de règles de réécritures probabilistes et d'une stratégie chargée de gérer l'application des règles et nous prouvons que ce système termine bien en temps moyen fini.

Plan du document

Ce document se décompose en les trois parties suivantes,

Rappels sur quelques notions de base

Dans cette partie nous décrirons de manière succincte les outils et les notions sur lesquelles reposent les travaux présentés dans ce document. Nous commencerons par introduire le formalisme de la réécriture en se basant sur la présentation de [BN98], puis nous aborderons l'étude de la terminaison des systèmes de réécriture.

Nous présenterons par la suite les bases de la théorie des probabilités pour pouvoir discuter des théorèmes que nous avons utilisé pour mettre en place notre modèle de réécriture probabiliste ainsi que les preuves de terminaison associées.

Un modèle de réécriture probabiliste

Nous présentons dans le premier chapitre de cette partie notre modèle de réécriture probabiliste ainsi qu'un critère de terminaison en temps moyen fini. Nous introduisons également la notion de stratégie de réécriture probabiliste.

Dans le second chapitre de cette partie, nous discutons de la terminaison en temps moyen fini de systèmes de réécritures probabilistes, quand l'application des règles de réécritures sont conditionnées par des stratégies. Nous exhibons dans cette partie, des critères généraux que nous avons trouvé et qui concernent les conditions que doivent satisfaire les stratégies pour que la terminaison en temps moyen fini se produise sur les systèmes de réécriture probabilistes.

Applications

Dans cette partie composée de deux chapitres nous introduisons dans le premier chapitre un modèle de produit synchronisé d'automates temporisés capable de représenter des réseaux informatiques sans-fil communicant en utilisant le protocole « WIFI ». Nous avons défini un tel formalisme de telle façon qu'il puisse être facilement modélisé grâce à des systèmes de réécriture et des systèmes de réécriture probabilistes. Dans le second chapitre, on utilise ce modèle de produit synchronisé, nous modélisons un réseau de stations WIFI en le codant grâce à des règles de réécriture probabilistes et une stratégie. Nous montrons, en employant nos techniques de preuve de terminaison en temps moyen fini, que l'algorithme utilisé par le protocole WIFI permet à chacune des stations du réseau d'envoyer l'ensemble de ses messages au bout d'un temps moyen fini, dans le cas d'un départ simultané.

Première partie
Notions préliminaires

1

Présentation de la réécriture et de la réécriture sous stratégie

Nous allons introduire dans ce chapitre la notion de système de réécriture. Nous aborderons tout d'abord quelques résultats généraux sur les propriétés des relations binaires, puis nous présenterons la définition des systèmes de réécriture et nous étudierons les résultats classiques concernant la terminaison de ces systèmes de réécriture. Nous parlerons par la suite de la notion de stratégie de réécriture et du problème de la terminaison des systèmes de réécriture sous stratégie.

Nous allons ici présenter des résultats généraux permettant d'introduire les systèmes de réécriture classiques, en se basant sur la présentation de [BN98]. Nous allons présenter dans un premier temps la notion de dérivations générées sur des ensembles par des relation binaires ainsi que la terminaison de ces dérivations. Ensuite, nous présenterons les système de réécriture et nous mentionnerons quelques résultats classiques permettant de prouver leur terminaison. Enfin, nous présenterons la notion de réécriture sous stratégie.

1.1 Relations binaires et systèmes de réduction abstraits

Nous associons à un ensemble A une relation binaire \rightarrow et la notion classique de système de réduction abstrait :

Définition 1.1.1 (Système de réduction abstrait (ARS)). Un système de réduction abstrait est une paire (A, \rightarrow) composée d'un ensemble A et d'une relation binaire $\rightarrow \subseteq A \times A$.

Notation 1. On note $a \rightarrow b$ au lieu de $(a, b) \in \rightarrow$.

Les principales propriétés des systèmes abstraits de réduction étudiées dans la théorie de la réécriture sont la terminaison et la confluence des réductions. Pour

pouvoir étudier ces propriétés, nous devons dans un premier temps décrire de quelle façon on compose la relation \rightarrow avec elle-même et introduire quelques notations utiles. Une fois cela fait, nous présenterons brièvement quelques propriétés sur la relation de dérivation ainsi que des méthodes classiques permettant de montrer qu'un ARS satisfait ces propriétés.

Définition 1.1.2 (Composition de deux relations binaires). Soient $R \in A \times B$ et $S \subseteq B \times C$ deux relations binaires. La composition de R avec S est définie de la façon suivante :

$$R \circ S := \{(x, z) | \exists y \in B \text{ tel que } (x, y) \in R \wedge (y, z) \in S\}$$

Définition 1.1.3. On définit les différentes façons de composer \rightarrow avec elle-même et on associe à chacune de ces manières une notation propre :

$\xrightarrow{0} := \{(x, x) x \in A\}$	Identité
$\xrightarrow{i+1} := \xrightarrow{i} \circ \rightarrow$	$(i + 1)$ – ème composition $i \geq 0$
$\xrightarrow{+} := \bigcup_{i>0} \xrightarrow{i}$	Clôture transitive
$\xrightarrow{*} := \xrightarrow{+} \cup \xrightarrow{0}$	Clôture réflexive et transitive
$\xrightarrow{-1} := \{(y, x) x \rightarrow y\}$	Inverse
$\xleftarrow{-1} := \xrightarrow{-1}$	Inverse
$\leftrightarrow := \xleftarrow{-1} \cup \rightarrow$	Clôture symétrique
$\leftrightarrow^+ := (\leftrightarrow)^+$	Clôture symétrique transitive
$\leftrightarrow^* := (\leftrightarrow)^*$	Clôture symétrique transitive et réflexive

Remarque 1.1.1. On émet quelques précisions :

1. On peut exprimer certaines de ces compositions en termes de chemin dans le graphe de (A, \rightarrow) :

$x \xrightarrow{n} y$ Il existe un chemin de longueur n reliant x à y .

$x \xrightarrow{*} y$ Il existe un chemin de longueur finie reliant x à y .

$x \xrightarrow{+} y$ Il existe un chemin de longueur non nulle et fini reliant x à y .

2. Le terme « clôture » ne voit pas son sens habituel galvaudé. En effet si on prend en considération $\xrightarrow{*}$, il s'agit de la plus petite relation réflexive et transitive contenant la relation \rightarrow . On notera bien que malgré le fait que la clôture d'une relation P par une relation R n'existe pas nécessairement dans le cas général, elle existe bien dans les cas mentionnés grâce au fait que la réflexivité, la transitivité et la symétrie sont closes par intersection.
3. \leftrightarrow^* est la plus petite relation d'équivalence contenant \rightarrow .

Maintenant que nous avons introduit les principales notations concernant les relations de réduction, nous pouvons définir quelques propriétés sur les éléments de A par rapport à la relation \rightarrow d'un ARS.

Définition 1.1.4.

- x est réductible s'il existe y tel que $x \rightarrow y$.
- x est une forme normale si x n'est pas réductible.
- y est une forme normale pour x si $x \xrightarrow{*} y$ et y est une forme normale. Dans ce cas on note y par $x \downarrow$.
- y est un successeur direct de x si $x \rightarrow y$.
- y est un successeur de x si $x \xrightarrow{+} y$.
- x et y sont joignables s'il existe un élément z tel que $x \xrightarrow{*} z \xleftarrow{*} y$. On note dans ce cas $x \downarrow y$.

Et maintenant définissons certaines propriétés importantes des relations de réduction :

Définition 1.1.5. Une relation \rightarrow est appelée :

- Church-Rosser si $x \xleftrightarrow{*} y \Rightarrow x \downarrow y$
- Confluente si $y_1 \xleftarrow{*} x \xrightarrow{*} y_2 \Rightarrow y_1 \downarrow y_2$
- Terminante si il n'existe pas de dérivation infinie $a_0 \rightarrow a_1 \rightarrow \dots$
- Normalisante si tout élément a une forme normale
- Convergente si \rightarrow est terminante et confluente

1.2 Le principe de l'induction bien fondée

Dans cette partie on décrit un principe fondamental, l'induction bien fondée, parfois appelé récurrence Noethérienne. On va voir que cette propriété caractérise les relations terminantes. Le principe d'induction bien fondée (WFI pour Well Founded Induction) est en fait une généralisation de l'induction sur $(\mathbb{N}, >)$ pour au cas des systèmes de réduction terminants (A, \rightarrow) .

Soit P une propriété sur A , énonçons la propriété d'induction bien fondée à l'aide d'une règle d'inférence :

$$\frac{\forall x \in A. (\forall y \in A. x \xrightarrow{+} y \Rightarrow P(y)) \Rightarrow P(x)}{\forall x \in A. P(x)}$$

Cette notion d'induction bien fondée est importante à cause du théorème suivant :

Théorème 1.2.1. *La relation \rightarrow termine si et seulement si \rightarrow vérifie le principe d'induction bien fondée.*

La preuve de ce théorème se trouvent dans [BN98]. Ce résultat permet d'étudier des critères permettant de certifier la terminaison de certaines dérivations.

Parmi ceux-ci, on peut citer :

Définition 1.2.1. Une relation \rightarrow est dite,

- finiment branchante si chaque élément n'a qu'un nombre fini de successeurs directs,
- globalement finie si chaque élément n'a qu'un nombre fini de successeurs,
- acyclique s'il n'existe pas d'élément a tels que $a \xrightarrow{+} a$.

Remarque 1.2.1. La relation \rightarrow est globalement finie si et seulement si $\xrightarrow{+}$ est finiment branchante.

On peut maintenant citer deux résultats concernant les relations terminantes :

Lemme 1.2.1. *Une relation finiment branchante est globalement finie si elle termine.*

La réciproque de ce dernier résultat est fautive, à cause des cycles qui peuvent apparaître dans la relation \rightarrow .

Mais si on suppose qu'elle est acyclique,

Lemme 1.2.2. *Une relation acyclique est terminante si elle est globalement finie.*

Des deux lemmes précédents, on déduit :

Lemme 1.2.3 (Lemme de König). *Un arbre finiment branchant est infini si et seulement s'il possède un chemin de longueur infinie.*

1.3 Terminaison des systèmes abstraits de réduction

Nous pouvons maintenant discuter de la question de la terminaison des systèmes de réduction abstraits. Cette notion de terminaison est une notion primordiale, à la fois pour celui qui conçoit des programmes et pour le théoricien qui cherche à caractériser l'ensemble ou un sous ensemble des algorithmes terminants. Dans la section précédente, nous mentionnons le fait qu'il y a une équivalence entre la terminaison et le fait que la relation de réduction satisfasse une propriété d'induction bien fondée. Cette propriété permet de déduire plusieurs méthodes de preuve pour montrer la terminaison des systèmes abstraits de réduction. Une des façons de montrer qu'une relation de dérivation d'un système abstrait de dérivation (A, \rightarrow) termine est de montrer que cet ensemble est en fait inclus dans un autre système de réduction abstrait $(B, >)$ qui est terminant. Classiquement, pour montrer qu'une telle inclusion existe, on construit une application monotone de A vers B .

Définition 1.3.1 (Application monotone (Monotone mapping)). Une application ϕ d'un ARS (A, \rightarrow) vers un autre ARS $(B, >)$ est dite monotone si pour tout $(x, y) \in A^2$ tels que $x \rightarrow y$ on a $\phi(x) > \phi(y)$.

Lemme 1.3.1. *Soit ϕ une application monotone de l'ARS (A, \rightarrow) vers $(B, >)$. Si $>$ est une relation terminante alors la relation \rightarrow termine.*

Démonstration. Supposons qu'il existe une chaîne de longueur infinie $x_0 \rightarrow x_1 \rightarrow \dots$, alors il existe une dérivation $\phi(x_0) > \phi(x_1) > \dots$, ce qui contredit l'hypothèse que $>$ est terminante. \square

Définition 1.3.2 (Image inverse). Soit ϕ une application de (A, \rightarrow) vers $(B, >)$. On note $\phi^{-1}(>)$ l'ensemble $\phi^{-1}(>) = \{(x, x') \mid \phi(x) > \phi(x')\}$.

Remarque 1.3.1. On remarque que $\rightarrow \subseteq \phi^{-1}(>)$.

Pour montrer que la relation de réduction \rightarrow d'un ARS (A, \rightarrow) vérifie le principe d'induction bien fondée, il suffit de construire une application monotone de (A, \rightarrow) vers $(\mathbb{N}, >)$.

Exemple 1.3.1. Pour des chaînes construites sur un ensemble de symboles X , c'est à dire $A := X^*$, on peut donner deux exemples :

- La longueur de la chaîne. Ici ϕ est l'application qui à un élément de A associe le nombre d'occurrence des symboles de X , $\phi(w) = |w|$. Cette fonction permet de prouver la terminaison de relation de réduction telle que $uabbbv \rightarrow uaav$ où $a, b \in X$ sont fixés et $u, v \in A$ sont libres.
- Le nombre d'occurrences d'un caractère. Pour tout $a \in X$ on définit l'application ϕ_a qui à un mot w associe le nombre d'occurrences de a au sein de w . Ce type de fonction suffit à prouver la terminaison de relations telles que $uav \rightarrow ubv$ avec $u, v \in A$ arbitraires et $a, b \in X$ sont fixés.

Le résultat suivant est l'un des résultats fondamentaux dans le domaine de l'étude de la terminaison des ARS.

Proposition 1.3.1. *Une réduction \rightarrow finiment branchante termine si et seulement si il existe une application monotone de (A, \rightarrow) vers $(\mathbb{N}, >)$.*

Démonstration. Le sens indirect est une conséquence du fait que $>$ termine et que ϕ préserve cette propriété par passage à l'image inverse (cohérence).

Montrons que l'implication dans le sens direct est vraie. Soit \rightarrow une relation finiment branchante et terminante. On définit $\phi(x)$ comme étant l'application qui à x associe le nombre de successeurs de x pour la relation \rightarrow qui d'après le lemme 1.2.1 est fini. Comme \rightarrow termine et par conséquent est acyclique, $x \rightarrow x'$ implique que le nombre de successeurs de x' est strictement inférieur à celui de x . \square

On perd l'équivalence si on ne suppose pas que la relation \rightarrow est finiment branchante, comme en témoigne cet exemple :

Exemple 1.3.2. Soit $A := \mathbb{N} \times \mathbb{N}$ et soit \rightarrow la relation de réduction définie par les deux règles $(i + 1, j) \rightarrow (i, k)$ et $(i, j + 1) \rightarrow (i, j)$ pour tout $i, j, k \in \mathbb{N}$. Cette relation n'est pas finiment branchante à cause de la variable k de la première règle qui n'est pas contrainte par les variables du membre droit. En effet, $(i + 1, j)$ à une infinité dénombrable de successeurs. Il n'y a pas d'application monotone de $(\mathbb{N} \times \mathbb{N}, \rightarrow)$ vers $(\mathbb{N}, >)$. Pour voir cela, supposons qu'une telle application ϕ existe. Sous cette hypothèse on a grâce à la monotonie de ϕ les inégalités suivantes $k := \phi(1, 1) > \phi(0, k) > \dots > \phi(0, 0)$. Cette dernière propriété est en fait absurde, car il n'y a que k entiers naturels inférieurs à k alors que la chaîne $\phi(0, k) > \dots > \phi(0, 0)$ est de longueur $k + 1$.

Remarque 1.3.2. Le système de réduction précédent est en fait terminant, on prouve sa terminaison en utilisant par exemple un ordre lexicographique. Cela montre qu'il existe des systèmes terminants pour lesquels un ordre permet de prouver la terminaison mais pour lesquels la méthode classique d'une application monotone vers $(\mathbb{N}, \rightarrow)$ ne peut être utilisée.

1.4 Les algèbres de termes

Nous allons maintenant présenter une manière générique de construire des termes à partir d'une signature, c'est à dire d'un ensemble de symboles de fonctions et d'un ensemble de variables. Nous verrons par la suite plusieurs notions et outils permettant de manipuler ces termes, les parcourir, les comparer et les modifier.

Définition 1.4.1 (Signature). Une signature Σ est un ensemble des symboles pour lequel chaque symbole est associé à une valeur entière positive appelée arité. On note Σ_n le sous ensemble des symboles de Σ d'arité n . L'arité d'un symbole $f \in \Sigma$ est noté $|f|$.

Remarque 1.4.1. Naturellement, $\Sigma = \cup_{n \geq 0} \Sigma_n$. On appelle l'ensemble Σ_0 l'ensemble des termes « constants ».

Définition 1.4.2 (Algèbre de termes). Soit $\Sigma = \{f_1, \dots, f_n\}$ une signature. Soit X un ensemble de variables. La Σ -algèbre libre homogène engendrée par X avec $X \cap \Sigma = \emptyset$, notée $T(\Sigma, X)$, est le plus petit ensemble tel que :

- $X \subset T(\Sigma, X)$.
- Pour tout symbole f de Σ d'arité n et pour tout $t_1, \dots, t_n \in T(\Sigma, X)$ alors $f(t_1, \dots, t_n) \in T(\Sigma, X)$.

Désormais, on appellera $T(\Sigma, X)$ l'algèbre de termes engendrée par la signature Σ . Les éléments de $T(\Sigma, X)$ sont appelés les termes du premier ordre.

Ainsi on construit de façon récursive des ensembles de termes pouvant être infinis mais dénombrables. Cette structure permet entre autres de pouvoir raisonner par récurrence sur la structure des termes afin de prouver des propriétés.

Définition 1.4.3 (Algèbres initiales). Une algèbre initiale, notée $T(\Sigma)$, est une algèbre homogène engendrée par un ensemble vide de variables. Les termes d'une telle algèbre sont appelés les termes clos.

On peut représenter les termes d'une algèbre de termes grâce à une représentation arborescente. On utilise la notion de position pour numéroter les sous termes d'un terme.

Définition 1.4.4 (Position). Soient Σ une signature et X un ensemble de variables disjoint de Σ . Soient s, t deux éléments de $T(\Sigma, X)$. L'ensemble des positions du terme s est un ensemble $Pos(s)$ de chaînes d'entiers naturels, défini par induction de la façon suivante :

- Si $s = x \in X$ alors $Pos(s) = \{\epsilon\}$ où ϵ représente la chaîne vide.
- Si $s = f(s_1, \dots, s_n)$ alors

$$Pos(s) = \{\epsilon\} \cup \bigcup_{i=1}^n \{ip \mid p \in Pos(s_i)\}.$$

La position ϵ est appelée la position racine du terme s . Cette indexation des sous termes permet de définir un ordre, nommé l'ordre préfixe, car il correspond au parcours de graphe éponyme.

Définition 1.4.5 (Ordre préfixe). $p \leq q$ si et seulement si il existe p' tel que $pp' = q$.

Remarque 1.4.2. L'ordre préfixe est un ordre partiel sur les positions. Deux positions p et q sont dites parallèles ($p \parallel q$) si elles ne sont pas comparables pour l'ordre \leq .

Définition 1.4.6 (Taille d'un terme). On note par $||$ la fonction qui à un terme s associe le cardinal $|s|$ de $Pos(s)$.

Définition 1.4.7 (Sous terme à une position). Soit $s \in T(\Sigma, X)$ un terme. Pour $p \in Pos(s)$, le sous-terme de s à la position p , noté $s|_p$, est défini par induction sur la longueur de p :

$$\begin{aligned} s|_\epsilon &:= s, \\ f(s_1, \dots, s_n)|_{iq} &:= s_i|_q \text{ avec } i \leq n. \end{aligned}$$

Définition 1.4.8 (Remplacement d'un sous terme). Soit $s \in T(\Sigma, X)$ un terme, $p \in Pos(s)$ la position d'un des sous termes de s . On note par $s[t]_p$ le terme correspondant au remplacement du sous terme de s à la position p par le terme t . Ce remplacement se définit par récurrence sur la longueur de p :

$$\begin{aligned} s[t]_\epsilon &:= t, \\ f(s_1, \dots, s_n)[t]_{iq} &:= f(s_1, \dots, s_i[t]_q, \dots, s_n) \text{ avec } i \leq n. \end{aligned}$$

Définition 1.4.9. Pour un terme $s \in T(\Sigma, X)$, l'ensemble $Var(s)$ est l'ensemble des variables de s .

$$Var(s) := \{s \in X \mid \exists p \in Pos(s) s|_p = x\}$$

Remarque 1.4.3. Un terme s est terme clos si et seulement si $Var(s) = \emptyset$.

Le lemme suivant présente les principales propriétés des positions et des remplacements des sous termes.

Lemme 1.4.1. Soient s, t, r trois termes et p, q des chaînes d'entiers :

1. Si $pq \in Pos(s)$ alors $s|_{pq} = (s|_p)|_q$.
2. Si $p \in Pos(s)$ et $q \in Pos(t)$ alors

$$\begin{aligned} (s[t]_p)|_{pq} &= t|_q. \\ (s[t]_p)[r]_{pq} &= s[t[r]_q]_p. \end{aligned}$$

3. Si $pq \in Pos(s)$ alors,

$$\begin{aligned} (s[t]_{pq})|_p &= (s|_p)[t]_q, \\ (s[t]_p) &= s[t]_p. \end{aligned}$$

4. Si p et q sont deux positions parallèles de $s \in T(\Sigma, X)$, alors

$$\begin{aligned} (s[t]_p)|_q &= s|_q \\ (s[t]_p)[r]_q &= (s[r]_q)[t]_p. \end{aligned}$$

1.4.1 Substitutions et relation de réduction sur les termes

Nous allons maintenant définir la notion de substitution qui permet de remplacer des variables par des termes.

Définition 1.4.10 (substitution). Soit Σ une signature et X un ensemble de variables dénombrable. On appelle une $T(\Sigma, X)$ -substitution une fonction $\sigma : X \rightarrow T(\Sigma, X)$ telle que $Dom(\sigma) = \{x \mid \sigma(x) \neq x\}$ est de cardinalité finie. Si $Dom(\sigma) = \{x_1, \dots, x_n\}$ alors on peut représenter σ par $\sigma = \{x_1 \mapsto \sigma(x_1), \dots, x_n \mapsto \sigma(x_n)\}$. L'ensemble des $T(\Sigma, X)$ -substitutions sera noté $Sub(T(\Sigma, X))$ ou tout simplement Sub quand il n'y a pas d'ambiguïté possible à propos de $T(\Sigma, X)$.

Définition 1.4.11. On note $Ran(\sigma)$ l'ensemble image de $Dom(\sigma)$ par σ , c'est à dire $Ran(\sigma) = \{\sigma(x) \mid x \in Dom(\sigma)\}$. De même on note par $\mathcal{V}Ran(\sigma)$ l'ensemble des variables de $Ran(\sigma)$, c'est à dire $\mathcal{V}Rand(\sigma) = \cup_{x \in Dom(\sigma)} Var\{\sigma(x)\}$.

Définition 1.4.12 (Instanciation d'une variable par une substitution). Une substitution σ instancie une variable x si $x \in Dom(\sigma)$.

Toute $T(\Sigma, X)$ -substitution σ s'étend en une application $\hat{\sigma} : T(\Sigma, X) \mapsto T(\Sigma, X)$ de la façon suivante :

$$\begin{aligned} \hat{\sigma}(x) &:= \sigma(x) && \text{Si } x \in X \\ \hat{\sigma}(f(s_1, \dots, s_n)) &:= f(\hat{\sigma}(s_1), \dots, \hat{\sigma}(s_n)) && \text{Si } f \in \Sigma^n \end{aligned}$$

Par la suite, pour faciliter la lecture des formules, nous ne ferons plus la distinction entre une $T(\Sigma, X)$ -substitution σ et son extension $\hat{\sigma}$ à tout $T(\Sigma, X)$ et on utilisera σ pour représenter les deux notions.

Définition 1.4.13 (Instance d'un terme par une substitution). Un terme t est une instance d'un terme s s'il existe une substitution σ telle que $t = \sigma(s)$.

Remarque 1.4.4. On peut définir un ordre partiel à partir de la relation « être une instance d'un terme ». En effet on note $t \gtrsim s$ si t est une instance de s et on note $t > s$ si t est une instance de s mais que la réciproque est fausse.

1.4.2 Σ -identités et relations de réduction associées

Les Σ -identités sont des relations binaires sur les termes. À partir de ces relations, nous présenterons de quelle façon on peut construire une relation de réduction sur l'ensemble des termes. Ceci nous servira à définir les règles de réécriture et définir la relation de réécriture sur les systèmes de réécritures.

Définition 1.4.14 (Σ -identité). Soit Σ une signature et X un ensemble infini mais dénombrable de variables disjointes de Σ . Une Σ identité – où identité quand il n'y a pas d'ambiguïté à propos de Σ – est une paire $(s, t) \in T(\Sigma, X)^2$. On note $s \approx t$ pour décrire que s et t sont en relation par une Σ -identité. On appelle s le membre gauche et t le membre droit de l'identité $s \approx t$.

À partir d'un ensemble de Σ -identités, on peut construire une relation de réduction. Deux termes s et t seront en relation si pour ces deux termes il existe une position p , il existe une Σ -identité $l \approx r \in E$ et une substitution σ telle que les sous termes de s et t à la position p sont respectivement les images de l et r par σ . De façon plus formelle :

Définition 1.4.15 (Relation de réduction). Soit E un ensemble de Σ -identités. La relation de réduction $\rightarrow_E \subseteq T(\Sigma, X) \times T(\Sigma, X)$ est définie de la façon suivante :

$$s \rightarrow_E t \text{ si et seulement si}$$

$$\exists (l, r) \in E, p \in Pos(s), \sigma \in Sub \text{ tels que } s|_p = \sigma(l) \text{ et } t = s[\sigma(r)]_p.$$

Remarque 1.4.5.

Si E est un ensemble de Σ identités sur $T(\Sigma, X)$ alors on peut étudier les propriétés de la relation de réduction \rightarrow_E en considèrent l'ensemble $(T(\Sigma, X), \rightarrow_E)$ comme une

instance particulière des systèmes abstraits de réduction. On appliquera par exemple les résultats concernant la terminaison de la relation de réduction des ARS pour éventuellement savoir si \rightarrow_E est une relation terminante.

Avant d'entamer l'étude des systèmes de réécriture, nous devons introduire quelques définitions concernant les relations binaires sur les algèbres de termes :

Définition 1.4.16. Soit \equiv une relation binaire sur $T(\Sigma, X)$.

1. La relation \equiv est dite close par substitution si et seulement si $s \equiv t \Rightarrow \sigma(s) \equiv \sigma(t)$, pour tout $s, t \in T(\Sigma, X)$ et toute substitution $\sigma \in \mathcal{S}ub$.
2. Une relation \equiv est close par Σ -opérations si et seulement si $s_1 \equiv t_1, \dots, s_n \equiv t_n$ implique $f(t_1, \dots, t_n) \equiv f(s_1, \dots, s_n)$, pour tout $f \in \Sigma^n$ et tout

$$(s_1, \dots, s_n, t_1, \dots, t_n) \in T(\Sigma, X)^{2n}.$$

3. La relation \equiv est compatible par Σ -opérations si et seulement si $s \equiv t$ implique $f(s_1, \dots, s_{i-1}, s, s_{i+1}, \dots, s_n) \equiv f(s_1, \dots, s_{i-1}, t, s_{i+1}, \dots, s_n)$.
4. La relation \equiv est compatible par Σ -contexte si et seulement si $s \equiv s'$ implique $t[s]_p \equiv t[s']_p$ pour tout Σ -terme t et toute position $p \in Pos(t)$.

Toute Σ -identité \rightarrow_E est par définition compatible par Σ -contextes et est compatible par Σ -opérations.

Lemme 1.4.2. Soit E un ensemble de Σ -identités. La relation de réduction \rightarrow_E est close par Σ -identités et compatible par Σ -opérations.

1.5 Les systèmes de réécriture

Nous disposons désormais de tous les éléments nécessaires pour décrire les systèmes de réécriture. De façon simple, un système de réécriture, notés TRS pour Term Rewrite System, est formé d'un ensemble fini de Σ -identités particulières appelées règles de réécriture. Ainsi, à partir d'un ensemble de règles de réécriture, on définit une relation de réduction appelée « relation de réécriture » suivant la définition 1.4.15.

Définition 1.5.1 (Règle de réécriture). Une règle de réécriture est une identité $l \approx r$ telle que l n'est pas une variable et telle que $Var(r) \subseteq Var(l)$. On note une telle règle $l \rightarrow r$ au lieu de $l \approx r$.

Maintenant qu'on a défini ce qu'est une règle de réécriture, on peut poser la définition des systèmes de réécriture :

Définition 1.5.2 (Système de réécriture). Un système de réécriture est un ensemble fini de règles de réécriture.

Introduisons un terme technique couramment employé,

Définition 1.5.3 (Redex [BN98]). Un redex est un terme qui est une instance du membre gauche d'une règle de réécriture.

En d'autres termes, si $R = \{l_1 \rightarrow r_1, \dots, l_n \rightarrow r_n\}$ est un système de réécriture et t un terme, alors t est un redex s'il existe une substitution σ et une règle $l_k \rightarrow r_k \in R$ telle que $t = \sigma(l_k)$.

Comme les règles de réécriture sont des cas particuliers d'identités, on définit de façon naturelle la relation de réduction associée à cet ensemble d'identités.

Définition 1.5.4 (Relation de réécriture associée à système de réécriture [BN98]). La relation de réduction associée à un système de réécriture est appelée relation de réécriture.

En d'autre termes, deux termes t_1 et t_2 sont en relation par la relation de réduction \rightarrow_R s'il existe une position $p \in \mathcal{Pos}(t_1)$, une substitution $\sigma \in \mathcal{Sub}$ et une règle de réécriture $l \rightarrow r$ telle que $t_1|_p = \sigma(l)$ et $t_2 = x[\sigma(r)]_p$.

Toutes les relations de réécriture vérifient l'équivalence suivante :

Propriété 1.5.1 ([BN98]). Une relation sur $T(\Sigma, X)$ est une relation de réécriture si et seulement si elle est compatible avec les Σ -opérations et close par substitution.

Il existe de nombreux problèmes soulevés par l'étude de la réécriture, tels les problèmes concernant le comportement de ces objets en tant qu'instances de systèmes de réduction abstraits. On peut également mentionner les problèmes de terminaison qui nous intéressent dans ce travail, le problème de la confluence de ces systèmes ainsi que le rapport entre l'étude du comportement des systèmes de réécriture et la satisfiabilité et la décidabilité des problèmes d'équation aux classes dans les théories équationnelles. Nous allons nous focaliser sur les aspects qui concernent la terminaison des systèmes de réécriture, pour pouvoir par la suite avoir un recul sur l'étude de ce problème quand nous travaillerons avec la généralisation au cas probabiliste des systèmes de réécriture.

1.6 Terminaison des systèmes de réécriture

On trouve dans [HL78, Dau89] des preuves du fait que les systèmes de réécriture sont en fait Turing-complets. Cela signifie que répondre à la question du problème de la terminaison des systèmes de réécriture dans le cas général est indécidable car il se réduit au problème de la halte des machines de Turing. Néanmoins, il existe des classes

de systèmes de réécriture sur lesquels le problème de la terminaison devient décidable. Nous allons ici présenter dans les grandes lignes ces cas de figure et présenter des méthodes permettant de prouver la terminaison des systèmes de réécriture quand cela est possible.

Propriété 1.6.1 (Indécidabilité du problème de la terminaison). Le problème suivant est en général indécidable : Étant donné un système de réécriture R , est-ce que toutes les réductions partant de tous les termes sont de longueur finie ?

Nous allons maintenant présenter des propriétés qui impliquent la terminaison des systèmes de réécriture.

1.6.1 Les ordres de réduction

Comme discuté dans la section concernant les systèmes abstraits de réduction, montrer que la relation de réécriture satisfait le principe d'induction bien fondée suffit à montrer que cette relation est terminante. Une première façon de montrer qu'un système de réécriture termine consiste à trouver un ordre strict $>$ sur $T(\Sigma, X)$ qui vérifie le principe d'induction bien fondée.

Soit une relation de réécriture R , supposons qu'il existe un ordre strict $>$ sur $T(\Sigma, X)$ qui satisfait le principe d'induction bien fondée, si pour tout $t \rightarrow_R s$ on a $t > s$ alors R termine. Cette première façon de prouver la terminaison présente l'énorme inconvénient de requérir le test pour l'ensemble des couples $t \rightarrow_R s$, potentiellement infini.

Heureusement, il existe d'autres façons de vérifier que le système de réécriture R termine, qui requièrent seulement la vérification de la propriété $l_k > r_k$ pour tout $l_k \rightarrow r_k \in R$. Une de ces façons consiste à mettre en évidence l'existence d'un ordre de réécriture.

Définition 1.6.1 (Ordre de réécriture). Soit Σ une signature et X un ensemble dénombrable de variables. Un ordre strict sur $T(\Sigma, X)$ est un ordre de réécriture si et seulement si cet ordre est :

- compatible avec les Σ -opérations,
- clos par substitution.

Définition 1.6.2 (Ordre de réduction). Un ordre $>$ de réécriture est un ordre de réduction si et seulement s'il est terminant.

La recherche d'un ordre de réduction est motivée par le résultat suivant,

Théorème 1.6.1. *Un système de réécriture R termine si et seulement si il existe un ordre de réduction $>$ satisfaisant $l > r$ pour toute règle $l \rightarrow r \in R$.*

1.6.2 Méthode d'interprétation

Les méthodes d'interprétation sont classiquement utilisées pour montrer la terminaison de système de réécriture. À la place de construire un ordre vérifiant le principe d'induction bien fondée sur l'ensemble des termes $T(\Sigma, X)$, ces méthodes étudient le comportement de l'image de ces termes par une famille de fonctions évaluant les termes vers un ensemble où il existe un ordre bien fondé.

Définition 1.6.3. Soit \mathcal{A} une Σ -algèbre non vide et soit $>$ un ordre strict et bien fondé sur A le support de \mathcal{A} . Soit $>_{\mathcal{A}}$ la relation binaire définie par

$$s >_{\mathcal{A}} t \text{ ssi } \pi(s) > \pi(t)$$

pour tout homomorphisme $\pi : \mathcal{T}(\Sigma, X) \rightarrow \mathcal{A}$.

On a la stabilité par substitution car $>_{\mathcal{A}}$ est définie pour toute les valuation possibles des variables. Afin d'être conforme avec la notion de stabilité par Σ -opérations, il est nécessaire que toutes les interprétations des symboles de fonction vérifient la propriété de monotonie suivante :

Définition 1.6.4 (Monotonie d'une fonction pour un ordre). Soit $>$ un ordre strict sur l'ensemble A . Une fonction $F : A^n \rightarrow A$ est dite monotone pour l'ordre $>$ si et seulement si l'implication

$$a > b \Rightarrow F(a_1, \dots, a_{i-1}, a, a_{i+1}, \dots, a_n) > F(a_1, \dots, a_{i-1}, b, a_{i+1}, \dots, a_n)$$

est vérifiée pour tout i , $1 \leq i \leq n$ et pour tout $a, b, a_1, \dots, a_{i-1}, a_{i+1}, \dots, a_n \in A^{n+1}$.

Théorème 1.6.2. Soit \mathcal{A} et $>$ satisfaisant les propriétés de la définition 1.6.3. Si l'interprétation $f^{\mathcal{A}}$ de toute fonction $f \in \Sigma$ est monotone pour l'ordre $>$ alors $>_{\mathcal{A}}$ est un ordre de réduction sur $T(\Sigma, X)$.

Ce théorème permet de trouver des conditions permettant de construire des interprétations telles que l'ordre induit par cette interprétation est un ordre d'interprétation. Au rang des méthodes d'interprétations, on peut citer les méthodes d'interprétations polynomiales.

Définition 1.6.5 (Interprétation polynomiale). Soit Σ une signature. Une interprétation polynomiale de Σ est une Σ -algèbre \mathcal{A} qui vérifie les propriétés :

- Le support de \mathcal{A} est inclus dans $\mathbb{N} - \{0\}$,
- Toute fonction $f \in \Sigma^n$ d'arité n est associée à un polynôme $P_f(X_1, \dots, X_n) \in \mathbb{N}[X_1, \dots, X_n]$. L'interprétation polynomiale de f dans \mathcal{A} est l'évaluation de la fonction P_f , c'est à dire $f^{\mathcal{A}}(a_1, \dots, a_n) := P_f(a_1, \dots, a_n)$.

L'ordre $>$ sur les entiers naturels, qui est bien fondé, est habituellement utilisé. Comme le support A d'une Σ -algèbre \mathcal{A} est clos sous Σ -opérations alors le support A d'une interprétation polynomiale est clos par l'évaluation des polynômes P_f . En d'autres termes pour tout $f \in \Sigma$ et $a_1, \dots, a_n \in A^n$ on a $P_f(a_1, \dots, a_n) \in A$. Il n'est pas possible de définir un ordre de simplification juste en associant à chaque symbole de fonction f un polynôme P_f et en fixant un support A d'entiers strictement positifs, il faut également pouvoir garantir que l'évaluation des polynômes est clos dans A .

Exemple 1.6.1 ([BN98]). Posons $\Sigma = \{\oplus, \odot\}$, une signature composée de deux fonctions d'arité 2. Soit $A = \mathbb{N} - \{0, 1\}$. Si on pose

$$\begin{aligned} P_{\oplus} &:= 2X + Y + 1 \\ P_{\odot} &:= XY \end{aligned}$$

et on associe P_{\oplus} à \oplus et P_{\odot} à \odot alors l'interprétation polynomiale sur \mathcal{A} satisfait

$$\begin{aligned} \oplus^{\mathcal{A}}(m, n) &:= 2m + n + 1 \\ \odot^{\mathcal{A}}(m, n) &:= m \times n \end{aligned}$$

L'application associant les symboles de fonctions aux polynômes s'étend de la manière suivante : Pour un t contenant n variables x_1, \dots, x_n , on va construire un polynôme P_t à n variables X_1, \dots, X_n . Par exemple, l'interprétation polynomiale présentée ci-dessus associe le terme $t = x \odot (x \oplus y)$ avec le polynôme suivant :

$$\begin{aligned} P_t &= P_{\odot}(X, P_{\oplus}(X, Y)) \\ &= X(2X + Y + 1) \\ &= 2X^2 + XY + X. \end{aligned}$$

Remarque 1.6.1. Dans le cas général, les interprétations polynomiales ne sont pas forcément monotones pour l'ordre strict $>$.

On va définir des conditions sur les polynômes qui nous permettront de construire des interprétations « polynomiales monotones », c'est-à-dire une manière d'associer à un terme une fonction polynomiale qui sera monotone. Cette démarche permet de construire des ordres dits polynomiaux utilisés pour effectuer la preuve de terminaison de certains systèmes de réécritures, tels que celui présenté dans l'exemple 1.6.1.

Définition 1.6.6 (Polynôme monotone). Un polynôme $P \in \mathbb{N}[X_1, \dots, X_n]$ est dit monotone s'il existe au moins une occurrence –non nulle– de chaque variable X_i dans ce polynôme avec X_i apparaissant à un exposant supérieur ou égal à 1.

Définition 1.6.7 (Interprétation polynomiale monotone). Une interprétation polynomiale monotone est une interprétation polynomiale pour laquelle tous les symboles de fonction sont des polynômes monotones.

Ces interprétation polynomiales ont une propriété qui nous intéresse particulièrement :

Lemme 1.6.1. *Toutes les fonctions f^A d'une interprétation polynomiale monotone sont monotone.*

À partir de ce dernier lemme et du théorème 1.6.2 on montre que l'ordre $>_A$ induit par une interprétation polynomiale monotone est un ordre de réduction.

Définition 1.6.8 (Ordre polynomial). Un ordre polynômial est un ordre induit par une interprétation polynomiale monotone.

Cet ordre permet bien entendu de comparer des termes. Soit $P, Q \in \mathbb{N}[X_1, \dots, X_n]$. On note $P >_A Q$ si $P(a_1, \dots, a_n) > Q(a_1, \dots, a_n)$ pour tout vecteur $a_1, \dots, a_n \in A^n$.

Lemme 1.6.2. $l >_A r$ si et seulement si $P_l >_A P_r$.

Nous allons utiliser cet ordre de réduction dans le cadre d'une application pratique.

Exemple 1.6.2. Montrons que le système de réécriture

$$R := \begin{cases} x \odot (y \oplus z) & \rightarrow (x \odot y) \oplus (x \odot z) \\ (x \oplus y) \oplus z & \rightarrow x \oplus (y \oplus z) \end{cases}$$

termine en utilisant l'interprétation polynomiale présentée dans l'exemple 1.6.1. Posons P_{l_1} le polynôme associé au membre gauche de la première règle $l_1 := x \odot (y \oplus z)$ et P_{r_1} le polynôme associé au membre droit de l_1 . Calculons P_{l_1} et P_{r_1} ,

$$\begin{aligned} P_{l_1} &= X(2Y + Z + 1) = 2XY + XZ + X \\ P_{r_1} &= 2XY + XZ + 1. \end{aligned}$$

Rappelons que nous avons fixé l'ensemble support A à $\mathbb{N} - 0, 1$ et que pour tout $(a_1, a_2) \in A^2$ nous avons $P_{l_1}(a_1, a_2) > P_{r_1}(a_1, a_2)$, ce implique $P_{l_1} >_A P_{r_1}$.

De même, considérons P_{l_2} et P_{r_2} les polynômes respectivement associés au membre gauche et au membre droit de la règle de réécriture $l_2 := x \oplus (y \oplus z)$. Le calcul de ces deux polynômes

$$\begin{aligned} P_{l_2} &= 4X + 2Y + Z + 3 \\ P_{r_2} &= 2X + 2Y + Z + 2, \end{aligned}$$

montre bien que sur A on a bien $P_{l_2} >_A P_{r_2}$ qui est vérifiée. Les deux résultats précédents montrent que $l_1 >_A r_1$ ainsi que $l_2 >_A r_2$. Cela est suffisant pour montrer que le système de réécriture R est terminant.

1.6.3 Les ordres de simplification

Une seconde méthode de preuve de terminaison consiste à construire ce qu'on va appeler des ordres de simplification. Parmi ceux-ci, certains sont utilisés pour prouver de façon automatique la terminaison de systèmes de réécriture, comme l'ordre « KBO, pour Knuth-Bendix Order » et « RPO, pour Recursive Path Order ».

Définition 1.6.9 (Ordre de simplification). Soit Σ une signature et X un ensemble infini mais dénombrable de variables. Un ordre strict $>$ sur $T(\Sigma, X)$ est un ordre de simplification si $>$ est d'une part un ordre de réécriture et d'autre part satisfait la propriété suivante :

Pour tout terme $t \in T(\Sigma, X)$ et toute position $p \in \mathcal{Pos}(t) - \{\epsilon\}$, on a $t > t|_p$.

On appelle cette dernière propriété la « propriété de sous-terme ».

Lemme 1.6.3. *Un ordre de réécriture est un ordre de simplification si pour tout $n \geq 1$, tout symbole de fonction $f \in \Sigma^n$, toutes variables $x_1, \dots, x_n \in X^n$ et tout $i \in \mathbb{N}$ on a $f(x_1, \dots, x_i, \dots, x_n) > x_i$.*

Nous ne détaillerons pas d'avantage la description des ordres de simplification car nous ne les avons pas utilisé dans cette thèse. Néanmoins, on ne pouvait omettre de les mentionner dans un travail dédié à l'étude de la terminaison de certains systèmes de réécriture.

1.7 Les stratégies de réécriture

1.7.1 Présentation de la notion de stratégie de réécriture

Les stratégies de réécriture peuvent être considérées comme des algorithmes servant à sélectionner une sous partie des dérivations possible d'un système de réécriture. Une définition des stratégies de réécriture est donnée dans [KKV95] et se base sur la notion de logique de réécriture introduite dans [Mes92]. Ces dernières stratégies permettent de sélectionner un sous ensemble des dérivations qui correspondent en fait à des preuves et où une étape de réécriture correspond à un séquant.

Nous allons définir les stratégies par rapport aux dérivations d'un système de réécriture.

Définition 1.7.1 (Stratégie). Soit \mathcal{R} un système de réécriture et $(T(\Sigma, X), \rightarrow_{\mathcal{R}})$ l'ARS associé. Soit $D_f := \{(t_i)_{i \in \{1, \dots, n\}} \in T(\Sigma, X) \mid \forall i \in \{1, \dots, n-1\} t_i \rightarrow_{\mathcal{R}} t_{i+1}\}$ l'ensemble des dérivations de longueur finie. Une stratégie ϕ est une fonction de D_f vers $\rightarrow_{\mathcal{R}}$ telle que si $t_1, \dots, t_n \in D_f$ et t_n n'est pas un terme terminal, alors $\phi(t_1, \dots, t_n) = \rightarrow_r \in \rightarrow_{\mathcal{R}}$ et $t_n \rightarrow_r t_{n+1}$.

Remarque 1.7.1. On ne pose pas de restriction sur l'algorithme de stratégie. Il peut être aussi bien déterministe que non-déterministe.

Définition 1.7.2 (Réécriture sous stratégie). Soit \mathcal{R} un système de réécriture, $(T(\Sigma, X), \rightarrow_{\mathcal{R}})$ le système de réduction associé et ϕ une stratégie. Une séquence de termes a_0, \dots, a_n est une dérivation de $(T(\Sigma, X), \rightarrow_{\mathcal{R}})$ sous la stratégie ϕ si et seulement si pour tout $i + 1 \leq n$, $a_i \phi(a_0, \dots, a_i) a_{i+1}$.

Remarque 1.7.2. On associe à toute stratégie déterministe une unique dérivation.

Maintenant, nous pouvons définir la notion de terminaison des systèmes de réécriture sous stratégie.

Définition 1.7.3. Soit \mathcal{R} un système de réécriture et soit ϕ une stratégie. Le système de réécriture \mathcal{R} termine sous la stratégie ϕ s'il n'existe pas de chaîne infinie de terme $(a_i)_{i \geq 0}$ telle que $a_0 \phi(a_0) a_1 \phi(a_0, a_1) a_2 \dots a_n \phi(a_0, \dots, a_n) a_{n+1} \dots$

Donnons maintenant un exemple de système de réécriture non terminant et qui termine sous une stratégie que nous allons expliciter.

Exemple 1.7.1 (Toyama). Le système de réécriture suivant n'est pas terminant. Cependant, il termine sous certaines stratégies. Ce système de réécriture a été présenté par Yoshihito Toyama dans [Toy87] pour montrer que la somme directe de deux systèmes de réécriture terminants n'est pas toujours un système de réécriture terminant.

Soit le système de réécriture :

$$\begin{aligned} r_1 &:= f(0, 1, x) \rightarrow f(x, x, x) \\ r_2 &:= g(x, y) \rightarrow x \\ r_3 &:= g(x, y) \rightarrow y. \end{aligned}$$

x avec $f, g \in \Sigma$, $x, y \in X$ avec Σ et X disjoints. Le système de réécriture de Toyama ne termine pas. Pour le voir, il suffit de considérer la dérivation suivante :

$$\begin{aligned} f(0, 1, g(0, 1)) &\xrightarrow{r_1} f(g(0, 1), g(0, 1), g(0, 1)) \xrightarrow{r_2} \\ &f(0, g(0, 1), g(0, 1)) \xrightarrow{r_3} f(0, 1, g(0, 1)) \end{aligned}$$

Bien que ce système de réécriture ne termine pas, il existe une stratégie sous laquelle le système de réécriture termine. Soit la stratégie « innermost » qui applique les règles de réécriture sur les radicaux les plus en « profondeur ». C'est à dire que si le terme t a deux sous termes réductible à la position p_1 et à la position p_2 alors la stratégie innermost réduit le sous terme à la position p_1 si $|p_1| > |p_2|$.

Propriété 1.7.1. Sous la stratégie innermost, le cycle qui montre que l'exemple de Toyama, l'exemple 1.7.1, ne termine pas ne peut pas être produit lorsque la stratégie innermost conditionne l'application des règles de réécriture.

Démonstration. La stratégie « innermost » applique la règle r_2 ou r_3 sur le sous terme à la position 3 du terme $f(0, 1, g(0, 1))$, qui est alors réécrit en $f(0, 1, 0)$ ou en $f(0, 1, 1)$, termes qui se réécivent alors en des termes terminaux. \square

L'étude de la terminaison des systèmes de réécriture sous stratégie est à l'origine de nombreux travaux tels [AG97] et [FGK02a, FGK02b].

2

Les modèles de systèmes probabilistes

2.1 Rappels sur la théorie des probabilités

Nous allons dans cette section rappeler quelques fondements de la théorie des probabilités et fixer les notations que nous utiliserons ultérieurement. On invite le lecteur qui n'est pas familier avec ces notions essentielles à consulter un ouvrage de référence en la matière tels que [Fel68, BL98] ou un document mis en ligne sur le site d'une université tel que les cours de Jean Bertoin [Ber01] et de Jean Jacod [Jac03] disponibles sur <http://proba.jussieu.fr>.

2.1.1 Phénomènes aléatoires

Un phénomène est aléatoire lorsqu'il se produit un grand nombre de fois, possède une certaine régularité sans toutefois pouvoir être déterminé à l'avance. Un certain nombre d'exemples nous viennent des salles de jeux, où roulettes, dés et autres machines à sous nous offrent la possibilité de gagner de l'argent et surtout d'en perdre. Dans le cas classique du dé non pipé, on sait pertinemment qu'au bout d'un certain nombre de lancers, on observe que la proportion du nombre de jets où l'on obtient un 6 tend vers un sixième, de même pour les cinq autres valeurs. Cependant on ne peut pas prévoir quelle sera la prochaine valeur obtenue lors du prochain lancer, sachant la valeur obtenue lors des tirages précédents.

La théorie des probabilités fournit un modèle pour décrire de tels phénomènes. Elle s'appuie sur trois concepts fondamentaux :

1) Un espace d'états, noté Ω , qui représente l'ensemble de toutes les valeurs possibles pour les réalisations de l'expérience.

2) Les événements qui sont une collection de partie de Ω .

Soient A et B deux événements alors la tribu \mathcal{A} des événements contenant A et

B doit vérifier les propriétés ensemblistes suivantes :

- A^C l'événement contraire de A est dans la tribu des événements,
- $A \cup B$ qui est l'événement A ou B est dans la tribu \mathcal{A} ,
- $A \cap B$ l'événement A et B est dans \mathcal{A} ,

On définit les événements particuliers suivants,

- $\emptyset \in \mathcal{A}$ est l'événement impossible,
- $\Omega \in \mathcal{A}$ est l'événement certain et enfin
- $\{\omega\}$ est un événement élémentaire si ω est un seul et unique point de Ω .

Clairement on a d'après ces propriétés que $\mathcal{A} \subseteq \mathcal{P}(\Omega)$ où $\mathcal{P}(\Omega)$ est l'ensemble des parties de Ω .

3) La probabilité : qui est en fait une mesure [Gir04] sur l'espace des événements qui à un événement $A \in \mathcal{A}$ associe la valeur réelle $P(A)$ avec $P(A)$ satisfaisant les conditions suivantes :

- (P0) $0 \leq P(A) \leq 1$,
- (P1) $P(\Omega) = 1$,
- (P2) $P(A \cup B) = P(A) + P(B)$ si $A \cap B = \emptyset$.

On en déduit

$$P(\emptyset) = 0, \text{ Appliquer P2 avec } A = \emptyset \text{ et } B = \emptyset, \quad (2.1)$$

$$P(A) + P(A^C) = 1, \quad (2.2)$$

$$P(\cup_{i=1}^n A_i) = \sum_{i=1}^n P(A_i) \text{ si les } A_i \text{ sont deux à deux disjoints,} \quad (2.3)$$

$$P(A \cup B) + P(A \cap B) = P(A) + P(B), \quad (2.4)$$

$$P(A) \leq P(B) \text{ si } A \subseteq B \quad (2.5)$$

On définit donc un espace probabiliste par un triplet (Ω, \mathcal{A}, P) constitué de l'espace Ω , de la tribu des événements \mathcal{A} et de la famille des $P(A)$ pour $A \in \mathcal{A}$. On peut de cette manière considérer P comme une application de \mathcal{A} dans $[0, 1]$, qui vérifie les propriétés (P1) et (P2). On peut à ce stade décrire à quoi correspond une variable aléatoire :

4) Variable aléatoire : Il s'agit d'une grandeur dépendant du résultat de l'expérience. Il s'agit en fait d'une application de Ω dans un espace E , en général \mathcal{R}^n . Soit X une telle variable qui applique Ω dans E . On peut alors « transporter » la structure probabiliste sur l'espace d'arrivée E , en posant

$$P_X(B) = P(X^{-1}(B)) \text{ pour } B \subset E, \quad (2.6)$$

où $X^{-1}(B)$ désigne l'image réciproque de B par X , c'est à dire les $\omega \in \Omega$ tels que $X(\omega) \in B$. Cette formule définit une nouvelle probabilité P_X sur l'espace E . Cette probabilité P_X s'appelle la loi de la variable X .

Notation 2 (Loi d'une variable aléatoire X , $\mathcal{L}(X)$). *On utilise la notation $\mathcal{L}(X)$ pour représenter la loi d'une variable aléatoire X .*

2.1.2 Propriétés importantes des probabilités

Définition 2.1.1 (Indépendance). Une famille d'événements $\{A_i\}_{i \in \mathbb{N}}$ est dite indépendante si

$$P(\cap_{i \in \mathbb{N}} A_i) = \prod_{i \in \mathbb{N}} P(A_i)$$

Définition 2.1.2 (Limite d'une suite monotone d'événements). Soit $\{A_i\}_{i \in \mathbb{N}}$ une suite croissante, respectivement décroissante d'événements, c'est à dire $\forall n \in \mathbb{N} A_n \subseteq A_{n+1}$, respectivement $\forall n \in \mathbb{N} A_{n+1} \subseteq A_n$.

On note

$$\lim_{n \rightarrow \infty} A_n = \cup_{i=1}^{\infty} A_i \text{ Si } \{A_i\}_{i \in \mathbb{N}} \text{ est croissante.}$$

On note

$$\lim_{n \rightarrow \infty} A_n = \cap_{i=1}^{\infty} A_i \text{ Si } \{A_i\}_{i \in \mathbb{N}} \text{ est décroissante.}$$

Proposition 2.1.1. Soit $\{A_i\}_{i \in \mathbb{N}}$ une famille croissante ou décroissante d'événements. On a alors l'égalité suivante :

$$P(\lim_{n \rightarrow \infty} A_n) = \lim_{n \rightarrow \infty} P(A_n)$$

Définition 2.1.3 (Limite supérieure d'une famille d'événements). La limite supérieure d'une famille d'événements est l'ensemble des événements qui se produisent infiniment souvent, c'est à dire :

$$\limsup_{i \rightarrow \infty} A_i = \cap_{n \geq 1} \cup_{i \geq n} A_i$$

Définition 2.1.4 (Limite inférieure d'une famille d'événements). La limite inférieure d'une famille d'événements est l'ensemble des événements qui se produisent toujours à partir d'un certain rang, formellement on écrit :

$$\liminf_{i \rightarrow \infty} A_i = \cup_{n \geq 1} \cap_{i \geq n} A_i$$

Lemme 2.1.1 (Borel-Cantelli). Soit $\{A_i\}_{i \in \mathbb{N}}$ une famille d'événements, Si

$$\sum_{i=1}^{\infty} P(A_i) < \infty$$

alors

$$P(\limsup_{i \rightarrow \infty} A_i) = 0.$$

Remarque 2.1.1. Ce lemme est à la base de l'étude du comportement asymptotique des processus stochastiques à temps discret. Une première interprétation consiste à dire que les éléments d'une famille de processus sont vérifiés infiniment souvent soit avec probabilité 1 soit avec probabilité 0.

Remarque 2.1.2. La réciproque du lemme de Borel-Cantelli est fautive dans le cas général mais vraie si on suppose que les A_i sont indépendants.

Lemme 2.1.2 (Réciproque du lemme de Borel-Cantelli). *Si $\{A_i\}_{i \in \mathbb{N}}$ est une famille d'événements indépendants vérifiant*

$$\sum_{i \geq 1} P(A_i) = \infty$$

alors

$$P(\limsup_{i \rightarrow \infty} A_i) = 1$$

2.1.3 Espérance d'une variable aléatoire

Nous allons définir ici une notion très importante en probabilité, l'espérance mathématique d'une variable aléatoire, aussi appelée la moyenne et parfois le « premier moment » d'une variable aléatoire.

Définition 2.1.5 (Espérance mathématique). L'espérance mathématique d'une variable aléatoire X à valeur dans l'espace $\Omega \subseteq \mathbb{R}$ et ayant pour loi de probabilité P est définie de la façon suivante :

$$\begin{aligned} E[X] &= \int_{x \in \Omega} x dP(x) \text{ quand cette valeur existe} \\ &= \begin{cases} \int_{x \in \Omega} x P(x) dx & \text{si } X \text{ variable aléatoire continue} \\ \sum_{x \in \Omega} x P(X = x) & \text{si } X \text{ variable aléatoire discrète} \end{cases} \end{aligned}$$

Définition 2.1.6. Une variable aléatoire est dite intégrable sur (Ω, \mathcal{A}, P) si $E[|X|] < \infty$

Propriété 2.1.1 (Linéarité de l'espérance). Soient X et Y deux variables aléatoires à valeur dans \mathbb{R} , λ une constante réelle. L'espérance satisfait la propriété suivante :

$$E[\lambda \times X + Y] = \lambda \times E[X] + E[Y].$$

Propriété 2.1.2 (Espérance du produit de deux variables aléatoires). Soient X et Y deux variables aléatoires indépendantes. Dans ce cas, on a l'égalité suivante :

$$E[X \times Y] = E[X] \times E[Y]$$

Propriété 2.1.3. Si X est une variable aléatoire sur (Ω, \mathcal{A}, P) et à valeur dans \mathbb{R} dont l'espérance est finie, pour tout $A \in \mathbb{R}^+$ on a l'inégalité suivante :

$$P(\{\omega \in \Omega | X(\omega) > A\}) \leq \frac{1}{A} \times |E[X]|$$

On en déduit que,

Corollaire 2.1.1. *Si X est une variable aléatoire intégrable de (Ω, \mathcal{A}, P) à valeur dans \mathbb{R} alors la mesure de événements $\omega_\infty = \{\omega \in \Omega | X(\omega) = \infty\}$ vaut zéro, c'est à dire que $P(X = \infty) = 0$.*

Quand une variable aléatoire X à une espérance finie, alors on a le lemme suivant :

Lemme 2.1.3 (lemme du télescope [Fel68]).

$$E[X] = \lim_{n \rightarrow \infty} \sum_{i=0}^n P(X > i)$$

2.1.4 Convergence d'une suite de variables aléatoires

On va énumérer dans cette section les différents types de convergences de suite de variables aléatoires. Nous allons notamment nous intéresser par la suite à la convergence presque sûre, que nous allons définir en premier.

La convergence presque sûre

Définition 2.1.7 (Convergence presque sûre). Une suite de variables aléatoires réelles $(X_n)_{n \in \mathbb{N}}$ définies sur (Ω, \mathcal{A}, P) , converge presque sûrement (p.s.) vers la variable aléatoire réelle X , définie sur (Ω, \mathcal{A}, P) , si

$$P(\{\omega \in \Omega | \lim_{n \rightarrow \infty} X_n(\omega) = X(\omega)\}) = 1.$$

On note dans ce cas $\lim_{n \rightarrow \infty} X_n = X$ p.s. ou $X_n \rightarrow X$ p.s. lorsque $n \rightarrow \infty$.

Pour prouver la convergence presque sûre d'une suite de variables aléatoires on peut utiliser le lemme de Borel-Cantelli.

Lemme 2.1.4 (Borel-Cantelli). *Soit $(X_n)_{n \in \mathbb{N}}$ une suite de variables aléatoires réelles définies sur (Ω, \mathcal{A}, P) et X également une variable aléatoire définie sur le même espace,*

- *Si pour tout $\epsilon > 0$, $\sum_{n \in \mathbb{N}} P(|X_n - X| \geq \epsilon) < \infty$, alors $X_n \rightarrow X$ p.s..*
- *Si les $(X_n)_{n \in \mathbb{N}}$ sont mutuellement indépendantes, alors $X_n \rightarrow 0$ p.s. si et seulement si $\sum_{i \in \mathbb{N}} P(|X_i| \geq \epsilon) < \infty$ pour tout ϵ , avec 0 représentant la distribution telle que si X_0 à pour loi 0 alors $P(X_0 = 0) = 1$.*

La convergence en probabilité

La convergence en probabilité, appelée également « convergence en mesure », ou convergence dans $L^0(\Omega, \mathcal{A}, P)$ se définit de la façon suivante :

Définition 2.1.8 (Convergence en probabilité). Soient $(X_n)_{n \in \mathbb{N}}$, X des variables aléatoires réelles sur (Ω, \mathcal{A}, P) . On dit que X_n converge en probabilité vers X si pour tout $\epsilon > 0$ on a

$$\lim_{n \rightarrow \infty} P(|X_n - X| \geq \epsilon) = 0$$

On note dans ce cas $X_n \xrightarrow{P} X$.

Remarque 2.1.3. La convergence presque sûre implique la convergence en probabilité mais la réciproque est fautive. En effet si à partir d'un certain rang n_0 les variables aléatoires $(X_n)_{n \in \mathbb{N}}$, mutuellement indépendantes, on a $P(|X_n - X| \geq \frac{1}{n}) = \frac{1}{n}$ alors on a bien la convergence en probabilité mais $\lim_{n \rightarrow \infty} \sum_{n_0}^n P(|X_n - X| > \frac{1}{n_0}) = \infty$, ce qui montre la non convergence presque sûre (d'après le lemme de Borel-Cantelli).

La convergence L^p

Rappelons que X est un élément de $L^p(\Omega, \mathcal{A}, P)$ pour $p > 0$ si et seulement si $E(|X|^p) < \infty$. L'espace de fonction $L^p(\Omega, \mathcal{A}, P)$ muni de la norme

$$\|X\|_p = (E(|X|^p))^{\frac{1}{p}},$$

est un espace complet ¹.

Définition 2.1.9 (Convergence dans L^p). Soient $(X_n)_{n \in \mathbb{N}}$, X des variables aléatoires réelles sur (Ω, \mathcal{A}, P) . On dit que X_n converge vers X dans $L^p(\Omega, \mathcal{A}, P)$, $0 < p < \infty$ si $\lim_{n \rightarrow \infty} \|X_n - X\|_p = 0$.

La convergence en loi

Définition 2.1.10 (Convergence en loi). Soient $(X_n)_{n \in \mathbb{N}}$, X des variables aléatoires réelles sur (Ω, \mathcal{A}, P) . On dit que X_n converge en loi vers X , ou que les lois P^{X_n} convergent faiblement* (lire « faiblement étoile ») vers la loi P^X si l'une des quatre conditions équivalentes suivante est vérifiée :

- $\lim_{n \rightarrow \infty} F^{X_n}(t) = F^X(t)$ en tout point de continuité de F^X .
- $\lim_{n \rightarrow \infty} \int \phi(X_n) dP = \int \phi(X) dP$ pour toute fonction continue et bornée $\phi : \mathbb{R} \rightarrow \mathbb{R}$.
- $\lim_{n \rightarrow \infty} \varphi^{X_n}(t) = \varphi^X(t)$ pour tout $t \in \mathbb{R}$ et $\varphi^X(t) = \int_{\mathbb{R}} \exp t(d) P_X(t)$ la fonction caractéristique de X , c'est à dire $\varphi^X(t) = E(e^{itX})$.
- Il existe un espace probabilisé $(\Omega', \mathcal{A}', P')$ sur lequel est définie une suite de variables aléatoires $(X'_n)_{n \in \mathbb{N}}$ et X' , telles que X_n et X'_n ont une même loi, X et X'_n on une même loi et $\lim_{n \rightarrow \infty} X'_n = X'$ p.s.

On note dans ce cas $X_n \xrightarrow{\mathcal{L}} X$.

¹Toute suite de cauchy converge.

La convergence en loi est la convergence la plus faible, dans le sens où toutes les autres convergences énoncées précédemment implique cette convergence. On peut résumer le rapport entre les différents types de convergence grâce aux implications suivantes :

$$\begin{array}{ccccc} \text{Presque sûre} & \Rightarrow & \text{En probabilité} & \Rightarrow & \text{En loi} \\ & & \uparrow & & \\ & & L^p & & \end{array}$$

2.1.5 Probabilité conditionnelle et espérance conditionnelle

La notion d'espérance conditionnelle est une notion très importante dans la théorie des probabilités. Elle est à la base de l'étude d'une grande famille de processus stochastiques telles les martingales, les chaînes de Markov, etc. La première chose déroutante avec l'appellation « espérance conditionnelle » est qu'il s'agit en fait d'une variable aléatoire. Intuitivement cette notions d'espérance conditionnelle permet d'étudier le comportement d'une famille de variables aléatoires en émettant certaines hypothèses sur les réalisation d'une sous partie de ces variables aléatoires. D'un point de vue géométrique, l'espérance conditionnelle est l'unique variable aléatoire dont la densité correspond à la projection de l'ensemble des variables aléatoires sur la tribu engendrée par la famille des variables contraintes.

Définition 2.1.11 (Fonction indicatrice). On note la I_B les fonctions indicatrices qui évaluent les événements de Ω dans $\{0, 1\}$ de la façon suivante :

$$I_B(\omega) = \begin{cases} 1 & \text{si } \omega \in B \\ 0 & \text{sinon} \end{cases}$$

Définition 2.1.12 (Probabilité conditionnelle). Soit (Ω, \mathcal{A}, P) un espace probabilisé, soit $A \subseteq \mathcal{A}$ et $B \subseteq \mathcal{A}$ telle que $P(B) > 0$. La probabilité conditionnelle de A sachant B est définie comme il suit

$$P(A|B) = \frac{P(A \cap B)}{P(B)}.$$

Définition 2.1.13 (Loi conditionnelle). Soit (Ω, \mathcal{A}, P) un espace probabilisé, et $B \subseteq \mathcal{A}$ telle que $P(B) > 0$. On appelle la loi conditionnelle sachant B la mesure de probabilité qui à tout $A \in \mathcal{A}$ associe $P(A|B)$. On note cette loi $P(\cdot|B)$.

Définition 2.1.14 (Système complet d'événements). Soit (Ω, \mathcal{A}, P) un espace probabilisé. Une famille d'événements $(B_i)_{i \in I} \subset \mathcal{A}$ forme un système complet d'événements si les B_i sont disjoints et $P(\cup_{i \in I} B_i) = \sum_{i \in I} P(B_i) = 1$.

Définition 2.1.15 (Atome). Soit \mathcal{B} une tribu. Un événement $B \in \mathcal{B}$ est appelé un atome de \mathcal{B} si pour tout événement $C \in \mathcal{B}$ on a soit $C = B$ soit $C = \emptyset$.

Définition 2.1.16 (Probabilité conditionnelle, conditionnellement à une tribu). Soit \mathcal{B} une sous-tribu dans (Ω, \mathcal{A}, P) , engendrée par un système complet d'événements $(B_i)_{i \in I}$. Soit $I^* = \{i \in I \mid P(B_i) > 0\}$. On appelle probabilité conditionnelle de $A \in \mathcal{A}$ sachant \mathcal{B} la variable aléatoire

$$\sum_{i \in I^*} P(A|B_i) I_{B_i},$$

également notée $P(A|\mathcal{B})$.

La probabilité conditionnelle de A sachant une sous-tribu est donc une variable aléatoire, constante sur les atomes de cette sous-tribu et donc mesurable par rapport à \mathcal{B} .

Définition 2.1.17 (Espérance conditionnelle). Soit X une variable aléatoire intégrable sur (Ω, \mathcal{A}, P) et \mathcal{B} une sous-tribu engendrée par un système complet d'événements $(B_i)_{i \in I}$, $I \subset \mathbb{N}$. Soit $I^* = \{i \in I \mid P(B_i) > 0\}$. On appelle l'espérance conditionnelle de X sachant \mathcal{B} , notée $E(X|\mathcal{B})$, la variable aléatoire \mathcal{B} -mesurable

$$\sum_{i \in I^*} \frac{1}{P(B_i)} \left(\int_{B_i} X dP \right) I_{B_i}$$

On va définir l'espérance conditionnelle dans le cas général, c'est à dire dans le cas où on conditionne par une tribu quelconque.

Définition 2.1.18 (Espérance conditionnelle). Soit (Ω, \mathcal{A}, P) un espace probabilisé, et soit \mathcal{B} une sous-tribu de \mathcal{A} . Soit de plus une variable aléatoire X réelle sur (Ω, \mathcal{A}, P) , intégrable. Alors il existe une unique (p.s.) variable aléatoire, appelée espérance conditionnelle de X sachant \mathcal{B} , notée $E(X|\mathcal{B})$ telle que :

- (i) $\omega \rightarrow E(X|\mathcal{B})(\omega)$,
- (ii) pour tout $B \in \mathcal{B}$, $E(X|\mathcal{B}) = \int_B X dP$

Proposition 2.1.2 (Propriétés de l'espérance conditionnelle). *L'espérance conditionnelle satisfait les propriétés suivantes :*

1. $E(aX + bY + c|\mathcal{B}) = aE(X|\mathcal{B}) + bE(Y|\mathcal{B}) + c$ p.s.
2. Si $X \leq Y$, alors $E(X|\mathcal{B}) \leq E(Y|\mathcal{B})$ p.s.
3. Si X_n converge p.s. vers X en croissant, alors $E(X_n|\mathcal{B})$ converge p.s. en croissant vers $E(X|\mathcal{B})$.
4. Si $\phi : \mathbb{R} \rightarrow \mathbb{R}$ est convexe et $\phi(X)$ est intégrable, alors $\phi(E(X|\mathcal{B})) \leq E(\phi(X)|\mathcal{B})$ p.s. (inégalité de Jensen).
5. Si $\mathcal{B} = \{\emptyset, \Omega\}$ $E(X|\mathcal{B}) = E(X)$ p.s.
6. Si $\mathcal{C} \subset \mathcal{B} \subset \mathcal{A}$, $E(E(X|\mathcal{B})|\mathcal{C}) = E(X|\mathcal{C})$.

7. Si \mathcal{B} est indépendante de $\sigma(X)$, $E(X|\mathcal{B}) = E(X)$ p.s.
8. Si Y est \mathcal{B} -mesurable et XY est intégrable, $E(XY|\mathcal{B}) = YE(X|\mathcal{B})$.
9. Si X est de carré intégrable, $E(X|\mathcal{B})$ est la projection orthogonale de X sur le sous espace $L^2(\Omega, \mathcal{B}, P)$ dans l'espace de Hilbert $L^2(\Omega, \mathcal{A}, P)$.

Notation 3. Si \mathcal{B} est la tribu engendrée par la variable aléatoire Y alors on note $E(X|Y) = E(X|\mathcal{B})$.

2.2 Martingales, surmartingales et sous martingales

2.2.1 Martingales

Nous allons présenter dans cette section une famille de processus appelés « martingales ». Ces objets, ainsi que leurs dérivés « surmartingales » et « sous martingales » furent étudiés par Descartes lorsque celui-ci cherchait à trouver une façon de formaliser le problème de la ruine pour étudier la viabilité des jeux de hasard. Nous avons utilisé ces outils dans cette thèse car il existe de nombreux résultats concernant la convergence de tels processus.

Tous d'abord, nous allons introduire la notion de filtration qui correspond à certaines familles croissantes pour l'inclusion de sous tribus.

Définition 2.2.1 (Filtration [LP05]). On appelle filtration un 4-uplet $(\Omega, \mathcal{F}_n, \mathcal{A}, P)$ où (Ω, \mathcal{A}, P) est un espace probabilisé et \mathcal{F}_n est une famille croissante pour l'inclusion de sous tribus de \mathcal{A} .

On introduit la notion de processus adapté,

Définition 2.2.2 (Processus adapté [LP05]). On appelle processus adapté à valeur dans $(\mathbb{R}, \mathcal{B})$ un 5-uplet $X = (\Omega, \mathcal{A}, \mathcal{F}_n, (X_n)_{n \geq 0}, P)$ où $(\Omega, \mathcal{F}_n, \mathcal{A}, P)$ est une filtration et où pour tout n , X_n est une variable \mathcal{F}_n mesurable à valeur dans $(\mathbb{R}, \mathcal{B})$, où \mathcal{B} est la tribu des Boréliens, c'est à dire la tribu engendrée par les ouverts de \mathcal{R} .

Remarque 2.2.1. Étant donnée une famille de variables aléatoires $(X_n)_{n \in \mathbb{N}}$ on considère généralement

$$\mathcal{F}_n = \sigma(X_1, \dots, X_n)$$

la tribu engendrée par les variables aléatoires X_1, \dots, X_n . C'est bien une filtration car

$$\sigma(X_1, \dots, X_n) \subset \sigma(X_1, \dots, X_{n+1}).$$

Définition 2.2.3 (Martingale). Une \mathcal{F}_n -martingale est une suite de variables aléatoires réelles $(X_n)_{n \in \mathbb{N}}$ \mathcal{F}_n adaptée telle que $\forall n \in \mathbb{N}$ X_n est intégrable et

$$E(X_{n+1}|\mathcal{F}_n) = X_n.$$

Quand il n'y a pas d'ambiguïté à propos de la filtration $(\mathcal{F}_n)_{n \in \mathbb{N}}$ on parle de martingale.

Si un jeu d'argent se modélise sous la forme d'une martingale et X_n est la somme que possède le joueur à la n ème partie, alors le joueur possède en moyenne autant d'argent à la $n + 1$ -ième partie. On appelle un tel jeu un jeu équitable.

Définition 2.2.4 (Sur-martingale). On appelle \mathcal{F}_n -surmartingale toute famille de variables aléatoires réelles $(X_n)_{n \in \mathbb{N}}$ \mathcal{F}_n adaptée et intégrable vérifiant l'inégalité suivante :

$$E(X_{n+1}|\mathcal{F}_n) \leq X_n.$$

Si un jeu d'argent se modélise sous la forme d'une surmartingale et X_n est la somme que possède le joueur à la n ème partie, alors le joueur possède en moyenne moins d'argent à la $n + 1$ -ième partie. On appelle un tel jeu un jeu défavorable –du point de vue du joueur.

Définition 2.2.5 (Sous-martingale). On appelle \mathcal{F}_n -sous-martingale toute famille de variables aléatoires réelles $(X_n)_{n \in \mathbb{N}}$ \mathcal{F}_n adaptée et intégrable vérifiant l'inégalité suivante :

$$E(X_{n+1}|\mathcal{F}_n) \geq X_n$$

Si un jeu d'argent se modélise sous la forme d'une sous-martingale et X_n est la somme que possède le joueur à la n ème partie, alors le joueur possède en moyenne plus d'argent à la $n + 1$ -ième partie. On appelle un tel jeu un jeu favorable du point de vue du joueur.

Définition 2.2.6 (Martingale L^1). Une martingale $(X_n)_{n \in \mathbb{N}}$ est dite L^1 si

$$\sup_{n \in \mathbb{N}} \|X_n\| = \sup_{n \in \mathbb{N}} E(|X_n|) < \infty.$$

Ce genre de martingales est également appelé « martingale finie ».

Théorème 2.2.1 (Convergence des martingales [BL98]). *Toute martingale L^1 converge presque sûrement vers une limite X_∞ .*

Remarque 2.2.2. La limite n'est pas forcément dans L^1 .

Dans l'étude du comportement de processus probabilistes, on s'intéresse souvent à comprendre comment se comporte le dit processus à certains indices dépendant d'une certaine variable aléatoire. Parmi ces « variables aléatoires d'indice » on prête une grande importance à ce qu'on appelle les « temps d'arrêt ».

Définition 2.2.7 (Temps d'arrêt). Soient (Ω, \mathcal{A}, P) un espace probablisé, muni d'une filtration $(\mathcal{F}_n)_{n \in \mathbb{N}}$ et $T : \Omega \rightarrow \mathbb{N}$ une variable aléatoire. On appelle T un \mathcal{F}_n temps d'arrêt si pour tout $n \in \mathbb{N}$ $\{T \leq n\} \in \mathcal{F}_n$.

2.3 Chaînes de Markov

2.3.1 Chaînes de Markov à espace discret et à temps discret

Les chaînes de Markov définissent une classe de processus probabilistes. Pour rester très simple dans un premier temps, une chaîne de Markov à temps discret est une suite de variables aléatoires à réalisation dans un espace d'état \mathbf{E} au plus dénombrable. À chaque élément de \mathbf{E} on associe une loi de probabilité fixée sur les éléments de \mathbf{E} de telle façon que si le processus $X_n = k$ alors on peut calculer la probabilité que la réalisation de X_{n+1} vaille j . L'évolution d'une chaîne de Markov ne dépend que de l'état courant et non du passé, c'est ce qu'on appelle l'effet « sans mémoire ». Plus formellement,

Définition 2.3.1 (Chaîne de Markov). Une chaîne de Markov est une suite de variables aléatoires $(X_n)_{n \in \mathbb{N}}$ à réalisation dans un espace $(\mathbf{E}, \mathcal{P}(\mathbf{E}))$ au plus dénombrable et satisfaisant la propriété suivante, dite de Markov :

$$P(X_{n+1} = k_{n+1} | \cap_{i=1}^n \{X_i = k_i\}) = P(X_{n+1} = k_{n+1} | X_n = k_n) \quad (2.7)$$

L'égalité 2.7, montre que la loi de probabilité de X_{n+1} ne dépend que de la valeur courante X_n et non des valeurs réalisées par le passé.

Définition 2.3.2 (Chaîne de Markov homogène). Une chaîne de Markov $(X_n)_{n \in \mathbb{N}}$ est dite homogène si pour tout $i, j \in \mathbf{E}$, pour tout $n \in \mathbb{N}$ tel que $X_n = i$ on a $P(X_{n+1} = j | X_n = i) = p_{i,j}$ avec $p_{i,j}$ une constante de $[0, 1]$.

Définition 2.3.3 (Matrice stochastiques, de Markov). Une matrice $M = (M)_{\{i,j\} \in \mathbf{E}}$ est dite stochastique si

- pour tout $i \in \mathbf{E}$, $\sum_{j \in \mathbf{E}} M_{i,j} = 1$,
- pour tout $i, j \in \mathbf{E}^2$, $M_{i,j} \geq 0$.

Propriété 2.3.1 (Représentation des chaînes de Markov homogènes). On représente par convention une chaîne de Markov homogène par matrice stochastique $(p_{i,j})_{i,j \in \mathbf{E}}$.

Propriété 2.3.2. Le produit de deux matrices stochastiques est une matrice stochastique.

Propriété 2.3.3. Soit $P = (p_{i,j})_{i,j \in \mathbf{E}}$ une matrice stochastique représentant une chaîne de Markov homogène à valeurs dans $(\mathbf{E}, \mathcal{P}(\mathbf{E}))$. On note

$$P^n = \underbrace{P \times \dots \times P}_{n \text{ fois}}$$

la matrice P élevée à la n ème puissance et $p_{i,j}^n$ l'élément de la i ème ligne et de la j ème colonne. Le réel $p_{i,j}^n$ représente la probabilité de transiter de i vers j en exactement n étapes, c'est à dire

$$P(X_{k+n} = j | X_k = i) = p_{i,j}^n.$$

Les chaînes de Markov vérifient toujours une propriété très importante, appelée la propriété de « Markov fort ». Cette propriété explique comment se comporte une chaîne de Markov en fonction des réalisations d'un temps d'arrêt fini.

Propriété 2.3.4 (Propriété de Markov fort). Soit $(X_n)_{n \in \mathbb{N}} \in \mathbf{E}$ une chaîne de Markov, T un \mathcal{F}_n temps d'arrêt pour la filtration $\mathcal{F}_n = \sigma(X_1, \dots, X_n)$, alors

$$P(X_{T+n} = y \mid T < \infty, X_T = k_T, \dots, X_0 = k_0) \quad (2.8)$$

$$= P(X_{T+n} = y | T < \infty, X_T = k_T) \quad (2.9)$$

$$= P(X_n = y | X_0 = k_T) \quad (2.10)$$

2.3.2 Comportement asymptotique des chaînes de Markov

Il existe certaines classes de chaînes de Markov pour lesquelles il existe des lois limites et des distributions stationnaires. Nous allons discuter des conditions nécessaires et suffisantes à l'existence de distributions stationnaires et de lois limites.

Dans la suite de cet exposé, nous désignerons par \mathbf{P} une matrice stochastique. Nous noterons par $(X_n)_{n \in \mathbb{N}}$ une chaîne de Markov homogène définie sur un espace probabilisé (Ω, \mathcal{A}, P) à valeur dans $(\mathbf{E}, \mathcal{P}(\mathbf{E}))$ de matrice de transition \mathbf{P} et de mesure initiale μ .

Définition 2.3.4 (Tribu cylindrique). Soit $\mathbf{E}^{\mathbb{N}}$ l'espace des suites sur \mathbf{E} , $\mathbf{E}^{\mathbb{N}} = \{x = (x_i)_{i \in \mathbb{N}} | x_n \in \mathbf{E}\}$. La tribu cylindrique \mathcal{B} de $\mathbf{E}^{\mathbb{N}}$ est formée par les parties de la forme (cylindres)

$$B_0 \times \dots \times B_n \times \mathbf{E} \times \mathbf{E} \dots, B_1, \dots, B_n \in \mathcal{P}(E), n \in \mathbb{N}$$

L'ensemble $\mathbf{E}^{\mathbb{N}}$ est muni de sa tribu cylindrique et de la probabilité image $P_{\mu, \mathbf{P}}$ de P par la chaîne de Markov X . Par la suite on considère \mathbf{P} fixée et $P_{\mu, \mathbf{P}}$ sera notée P_{μ} avec μ non fixée.

Notation 4. Si μ est une probabilité sur \mathbf{E} , pour tout $i \in \mathbf{E}$, on note $\mu_i = \mu(\{i\})$. On désignera par μ le vecteur de composantes $(\mu_i)_{i \in \mathbf{E}}$.

Remarque 2.3.1. Comme μ est vu tel un vecteur, alors ${}^t \mathbf{P} \mu$ est également un vecteur et il est associé à ce vecteur une mesure de probabilité notée ${}^t \mathbf{P} \mu$.

Définition 2.3.5 (Mesure asymptotique). On dit que μ , probabilité sur \mathbf{E} est une mesure asymptotique de la chaîne $(X_n)_{n \in \mathbb{N}}$ s'il existe une probabilité μ_o sur \mathbf{E} telle que si μ_0 est la loi de X_0 alors $(X_n)_{n \in \mathbb{N}}$ converge en loi vers μ .

Dans la suite de cette section on va discuter de l'existence des mesures asymptotiques et de l'impact de la loi initiale pour la convergence de la chaîne de Markov.

Définition 2.3.6 (Mesure invariante). On dit que μ , mesure positive sur \mathbf{E} , est une mesure invariante de la chaîne $(X_n)_{n \in \mathbb{N}}$ de matrice de transition \mathbf{P} si ${}^t\mathbf{P}\mu = \mu$.

Remarque 2.3.2. Il existe des mesures invariantes qui ne sont pas des mesures de probabilités.

Proposition 2.3.1. Soit μ une probabilité sur \mathbf{E} . Les assertions suivantes sont équivalentes :

- μ est une mesure asymptotique de la chaîne ;
- μ est une mesure invariante de la chaîne.

Si de plus μ est une mesure de probabilité, alors les deux propositions précédentes sont équivalentes à celle ci-dessous :

$$\mathcal{L}(X_0) = \mu \Rightarrow \mathcal{L}(X_n) = \mu \text{ pour tout } n \in \mathbb{N},$$

où $\mathcal{L}(X)$ dénote la loi de la variable aléatoire X .

Dans le cas où l'espace \mathbf{E} est fini,

Théorème 2.3.1. Toute chaîne de Markov homogène à valeur dans un ensemble fini admet une mesure invariante.

La question de l'unicité de la mesure invariante se pose naturellement et l'unicité de la mesure invariante n'est pas vérifiée dans le cas générale. Cela provient du fait que les mesures invariantes sont en fait les vecteurs propres de la matrice \mathbf{P} associés à la valeur propre 1 et que dans le cas général il peut exister plusieurs vecteurs propres indépendants associés à la valeur propre 1. Il faut garder à l'esprit que ces mesures invariantes ne sont pas nécessairement des mesures de probabilité.

Définition 2.3.7 (Composantes connexes). Soit i et j deux éléments de \mathbf{E} . On dit que j est accessible depuis i , noté $i \rightarrow j$ s'il existe $n > 0$ tel que $P_{i,j}^n > 0$. On dit que i et j communiquent, noté $i \leftrightarrow j$ si $i \rightarrow j$ et $j \rightarrow i$. La relation \leftrightarrow est symétrique et transitive. Elle est également réflexive sur le sous ensemble de \mathbf{E} , noté \mathbf{E}' , des éléments qui communiquent avec un autre état. On appelle composante connexe ou classe de la chaîne, soit un singleton de \mathbf{E} \mathbf{E}' , soit une classe d'équivalence de \leftrightarrow restreinte à \mathbf{E}' .

Définition 2.3.8 (Chaîne de Markov irréductible). Une chaîne de Markov est dite irréductible si elle n'admet qu'une seule classe. Dans ce cas on dit que la matrice stochastique de transition associée à cette chaîne de Markov est une matrice irréductible.

De la façon avec laquelle communiquent les états d'une chaîne de Markov va dépendre la manière avec laquelle la chaîne considérée convergera vers une éventuelle distribution asymptotique. Nous allons présenter trois conditions appelées récurrence, transience et récurrence positive qui vont nous permettre de dire s'il existe une loi asymptotique vers laquelle convergera la chaîne de Markov.

2.3.3 Récurrence et transience

Notation 5. Soit $X = (X_n)_{n \in \mathbb{N}}$ une chaîne de Markov à valeur dans $(\mathbf{E}, \mathcal{P}(\mathbf{E}))$. Soit i un élément de \mathbf{E} . On note

$$N_i = N_i(X) = \text{card}\{n \geq 0 | X_n = i\}$$

le nombre de passages en i ,

$$\tau_i = \tau_i^i = \tau_i^i(X) = \inf\{n > 0 | X_n = i\}$$

et pour $n > 1$,

$$\tau_i^n = \tau_i^n(X) = \inf\{k > \tau_i^{n-1} | X_k = i\}.$$

On note également P_i la loi de la chaîne conditionnée à débiter à l'état i . On note aussi E_i l'espérance sous P_i , c'est à dire l'espérance conditionnelle à $X_0 = i$.

Remarque 2.3.3. Les $\tau_i^n, i \in \mathbf{E}, n \geq 1$ sont des temps d'arrêt relativement à toute filtration à laquelle la chaîne de Markov est adaptée.

Définition 2.3.9 (Etat récurrent, état transient). Un état i de \mathbf{E} est dit récurrent pour la chaîne de Markov $(X_n)_{n \in \mathbb{N}}$ si $P_i\{\tau_i < \infty\} = 1$. Il est appelé transient dans le cas contraire.

En d'autres termes, une chaîne de Markov passe infiniment souvent sur chacun de ses états récurrents avec probabilité un. De même, presque sûrement, elle ne passe qu'un nombre fini de fois par chacun de ses états transient.

Lemme 2.3.1. Soit $(X_n)_{n \in \mathbb{N}}$ une chaîne de Markov définie sur (Ω, \mathcal{A}, P) à valeur dans $(\mathbf{E}, \mathcal{P}(\mathbf{E}))$. Si i est un état récurrent, les τ_i^n sont des temps d'arrêt P_i -p.s. finis.

Théorème 2.3.2. Soit $(X_n)_{n \in \mathbb{N}}$ une chaîne de Markov définie sur (Ω, \mathcal{A}, P) à valeur dans $(\mathbf{E}, \mathcal{P}(\mathbf{E}))$. Un état de i de \mathbf{E} est récurrent si et seulement si

$$P_i\{N_i = \infty\} = 1.$$

Proposition 2.3.2. Soit i un point de \mathbf{E} . Alors

$$P_i\{N_i = \infty\} = 1 \Leftrightarrow P_i\{N_i = \infty\} > 0.$$

Proposition 2.3.3. *La variable aléatoire N_i est P_i -intégrable si et seulement si i est un point transient de \mathbf{E} .*

De même on a,

Proposition 2.3.4. *L'état i est récurrent si et seulement si la série $\sum_{n \geq 0} P_{i,i}^n$ est divergente.*

Voici un résultat important concernant les propriétés de transience et de récurrence,

Théorème 2.3.3. *Les propriétés de récurrence et de transience sont des propriétés de classe (pour la relation \leftrightarrow).*

C'est à dire que si i est récurrent –resp transient–, alors tout les états j tels que $i \leftrightarrow j$ sont récurrents –resp transients–.

Corollaire 2.3.1 (Chaîne de Markov irréductible). *Une chaîne de Markov est irréductible s'il n'existe qu'une seule classe d'états récurrents.*

L'étude du comportement asymptotique des chaînes de Markov irréductibles est amplement simplifiée car d'une part il n'existe qu'une seule classe d'états irréductible et d'autre part grâce au théorème suivant :

Théorème 2.3.4. *Une mesure asymptotique ne charge pas les points transients, c'est à dire que si μ est une mesure asymptotique et i un état transient alors $\mu(\{i\}) = 0$.*

Deux autres conditions suffisent à garantir qu'une chaîne de Markov irréductible converge en loi vers une distribution de probabilité, ces deux conditions s'appellent l'apériodicité et la récurrence positive.

Définition 2.3.10 (Période d'un point). On dit qu'un point $i \in \mathbf{E}$ est de période d pour la chaîne $(X_n)_{n \in \mathbb{N}}$ si $d = \text{pgcd}\{n \geq 1 | P_{i,i}^n > 0\}$. Un état de \mathbf{E} est dit apériodique pour X si $d = 1$.

Propriété 2.3.5. La période d'un état est en fait une propriété de classe.

Définition 2.3.11 (Chaîne de Markov apériodique). Une chaîne de Markov apériodique a tout ses état de période 1.

Définissons maintenant la récurrence positive :

Définition 2.3.12 (Etat récurrent positif). Un état $i \in \mathbf{E}$ est dit récurrent positif s'il est récurrent et que $E_i(\tau_i) = E(\tau_i | X_0 = i) < \infty$.

En d'autres termes, un état est récurrent positif si le nombre moyen de transitions nécessaires pour partir de cet état et y revenir est fini.

2.3.4 Convergence en loi des chaînes de Markov

Définition 2.3.13 (Ergodicité). Une chaîne de Markov $(X_n)_{n \in \mathbb{N}}$ est dite ergodique s'il existe une probabilité μ vers laquelle la chaîne $(X_n)_{n \in \mathbb{N}}$ converge en loi quelle que soit $X_0 = i \in \mathbf{E}$.

Théorème 2.3.5 (Ergodicité d'une chaîne de Markov irréductible). *Une chaîne de Markov est ergodique si elle est irréductible, apériodique, et récurrente positive.*

2.3.5 Exemples d'applications

Donnons quelques exemple d'applications de la théorie des chaînes de Markov pour étudier une catégorie de processus appelés « marches aléatoires ».

Définition 2.3.14 (Marche aléatoire sur \mathbb{Z}). Une marche aléatoire sur \mathbb{Z} est une chaîne de Markov sur l'espace d'état $(\mathbb{Z}, \mathcal{P}(\mathbb{Z}))$ et les transitions sont décrites de la façon suivante :

$$\begin{aligned} P(X_{n+1} = k + 1 | X_n = k) &= p \\ P(X_{n+1} = k - 1 | X_n = k) &= 1 - p \\ p &\in [0, 1]. \end{aligned}$$

On montre que tous les états de la marche aléatoire sur \mathbb{Z} sont transients si $p \neq \frac{1}{2}$. Pour voir cela, il suffit de voir que la série $\sum_{n \geq 1} P_{i,i}^n$ est convergente pour tout i . Cependant cette même série est divergente si jamais $p = \frac{1}{2}$, ce qui signifie que tout les états sont récurrents. On montre également que cette chaîne de Markov est apériodique et irréductible. Cependant on montre que 0 n'est pas récurrent positif, donc aucun état n'est récurrent positif. Cette chaîne de Markov n'est pas ergodique. Pour résumer, la probabilité que la chaîne de Markov atteigne un état $k \in \mathbb{Z}$ quelconque vaut 1 mais le nombre moyen de transitions nécessaire à cette chaîne de Markov pour retourner à un état déjà atteint est infini.

Définition 2.3.15 (Marche aléatoire symétrique sur \mathbb{Z}^2). Une marche aléatoire symétrique sur \mathbb{Z}^2 est une chaîne de Markov dont les transitions sont décrites de la façon suivante :

$$\begin{aligned} P(X_{n+1} = (k + 1, l) | X_n = (k, l)) &= \frac{1}{4} \\ P(X_{n+1} = (k - 1, l) | X_n = (k, l)) &= \frac{1}{4} \\ P(X_{n+1} = (k, l + 1) | X_n = (k, l)) &= \frac{1}{4} \\ P(X_{n+1} = (k, l - 1) | X_n = (k, l)) &= \frac{1}{4} \end{aligned}$$

On montre que tous les états de cette chaîne de Markov sont récurrents mais non récurrents positifs.

Définition 2.3.16 (Marche aléatoire symétrique sur \mathbb{Z}^n). Une marche aléatoire symétrique sur \mathbb{Z}^2 est une chaîne de Markov dont les transitions sont décrites de la

façon suivante :

$$\begin{aligned}
 P(X_{n+1} = (k_1 + 1, \dots, k_n) | X_n = (k_1, \dots, k_n)) &= \frac{1}{2^n} \\
 P(X_{n+1} = (k_1 - 1, \dots, k_n) | X_n = (k_1, \dots, k_n)) &= \frac{1}{2^n} \\
 &\vdots \\
 P(X_{n+1} = (k_1, \dots, k_l + 1, \dots, k_n) | X_n = (k_1, \dots, k_n)) &= \frac{1}{2^n} \\
 P(X_{n+1} = (k_1, \dots, k_l - 1, \dots, k_n) | X_n = (k_1, \dots, k_n)) &= \frac{1}{2^n} \\
 &\vdots \\
 P(X_{n+1} = (k_1, \dots, k_n + 1) | X_n = (k_1, \dots, k_n)) &= \frac{1}{2^n} \\
 P(X_{n+1} = (k_1, \dots, k_n - 1) | X_n = (k_1, \dots, k_n)) &= \frac{1}{2^n}
 \end{aligned}$$

On montre que tous les états de cette chaîne de Markov sont transients dès que $n \geq 3$.

2.3.6 Temps moyen d'atteinte d'un intervalle pour une surmartingale

Pour prouver la terminaison en temps moyen fini des programmes à base de règles probabilistes que nous allons définir dans cette thèse, nous utiliserons ce résultat clef, présenté dans [VM98] :

Soit (Ω, \mathcal{F}, P) un espace probabiliste, $(\mathcal{F}_n)_{n \geq 0}$ une filtration. Soit $\{S_i, i \geq 0\}$ un processus \mathcal{F}_n -adapté, à valeur dans \mathbb{R}^+ . Sans perte de généralité on suppose que S_0 est constant. Notons par τ le \mathcal{F}_n temps d'arrêt correspondant à la première entrée dans l'intervalle $[0, C]$, c'est à dire $\tau = \inf\{n \geq 0 | S_n \leq C\}$.

Notons par $\tilde{S}_n = S_{n \wedge \tau}$ où $n \wedge \tau$ se définit de la façon suivante :

$$n \wedge \tau = \begin{cases} n & \text{si } n \leq \tau \\ \tau & \text{si } n > \tau \end{cases}$$

Théorème 2.3.6 (Critère d'atteignabilité en temps moyen fini d'un intervalle par une surmartingale positive [VM98]). *Supposons que $S_0 > C$ et qu'il existe $\epsilon > 0$ et que pour tout $n \geq 0$,*

$$E[\tilde{S}_{n+1} | \mathcal{F}_n] \leq \tilde{S}_n - \epsilon 1_{\{\tau > n\}} \quad (2.11)$$

alors,

$$E[\tau] \leq \frac{S_0}{\epsilon} < \infty \quad (2.12)$$

Démonstration. En passant l'inégalité 2.11 à l'espérance, on obtient

$$E[E[\tilde{S}_{n+1} | \mathcal{F}_n] - \tilde{S}_n] = -\epsilon E[1_{\{\tau > n\}}]$$

et comme S_n est \mathcal{F}_n -mesurable alors on a :

$$E[E[\tilde{S}_{n+1} - \tilde{S}_n | \mathcal{F}_n]] = -\epsilon P(\tau > n).$$

En sommant sur n ,

$$0 \leq \sum_{i=1}^n E[E[\tilde{S}_{i+1} - \tilde{S}_i | \mathcal{F}_i]] \leq -\epsilon \sum_{i=1}^n P(\tau > i)$$

$$0 \leq E[E[\sum_{i=1}^n \tilde{S}_{i+1} - \tilde{S}_i | \mathcal{F}_n]] \leq -\epsilon \sum_{i=1}^n P(\tau > i)$$

$$0 \leq E[\tilde{S}_{n+1}] - S_0 \leq -\epsilon \sum_{i=1}^n P(\tau > i)$$

$$0 \leq C < S_0 \leq E[\tilde{S}_{n+1}] \leq -\epsilon \sum_{i=0}^n P(\tau > i) + S_0$$

ce donne d'après le lemme du télescope :

$$E[\tau] \leq \frac{S_0}{\epsilon} < \infty.$$

□

2.4 Processus de décision Markovien

Le modèle des processus markoviens est une généralisation du modèle des chaîne de Markov classique. Comme on l'a vu précédemment, dans une chaîne de Markov, on associe à chaque état une distribution de probabilité sur les autres états. Les processus de décision markoviens modélisent des systèmes où on veut pouvoir spécifier des époques de transitions, un ensemble d'états, des actions en relation avec ces états, des transitions probabilistes ainsi qu'une fonction de coût. Ainsi à chaque état on peut associer plusieurs actions déclenchant chacune une transition probabiliste. Pour étudier le comportement de tels objets, on utilise des algorithmes appelés stratégies, « scheduler » ou encore « ordonnanceurs ». Nous nous sommes intéressés à ce type d'objet car ils permettent de modéliser des systèmes où des comportements non-déterministes et probabilistes se combinent. Nous nous sommes d'ailleurs inspirés de la notion de stratégie pour notre modèle de réécriture probabiliste. Les résultats qui nous intéressaient le plus dans cadre de ce travail de thèse concernaient les problèmes de terminaison, qui peut être étudié tel un problème d'accessibilité d'états terminaux. En effet le but de notre étude est de savoir si on atteint un ensemble d'états considérés comme terminaux sous certains critères au bout d'un temps moyen fini. Dans le cas où l'espace d'états est fini il a été montré dans [dA97] que la résolution du problème d'accessibilité revenait à résoudre le problème du plus court chemin stochastique [BT91]. Les notations utilisées dans cette introduction sont tirées de [Put94, dA97]. Nous avons suivi la description du domaine présenté dans [Mes04].

Définissons de manière formelle ce qu'est un processus de décision Markovien.

Définition 2.4.1 (Processus de décision markovien (PDM)). Un processus de décision Markovien est un triplet (S, A, p) tel que :

- S est un ensemble fini d'états.
- pour tout $s \in S$, $A(s) \subseteq Acts$ est un ensemble non vide d'actions disponibles dans l'état s , inclus dans l'ensemble de toutes les actions $Acts$.
- Pour tout $s, t \in S$ et $a \in A(s)$, $p_{st}(a)$ est la probabilité de la transition de s vers t quand l'action a a été sélectionnée. Pour tout $s, t \in S$ et $a \in A(s)$, on a $0 \leq p_{st}(a) \leq 1$ et $\sum_{t \in S} p_{st}(a) = 1$.

Exemple 2.4.1. Considérons le processus de décision Markovien formé par l'ensemble des entiers naturels, des actions $Acts = \{a, b\}$ en relation avec tout les éléments de \mathbb{N} sauf 0 tels que :

$$\begin{aligned} p_{n,n+1}(a) &= p_1 \\ p_{n+1,n}(a) &= 1 - p_1 \\ p_{n,n+1}(b) &= p_2 \\ p_{n+1,n}(b) &= 1 - p_2 \\ p_{0,0}(a) = p_{0,0}(b) &= 1. \end{aligned}$$

Ce processus de décision Markovien modélise la marche aléatoire représentée par la figure 2.1, où les distributions qui chargent les états $n + 1$ et $n - 1$ avec probabilité p_1 et $1 - p_1$ à partir de l'état n sont celles choisies par le déclenchement de l'action a et les actions chargeant les états $n + 1$ et $n - 1$ avec probabilité p_2 et $1 - p_2$ sont celles déclenchées par l'action b .

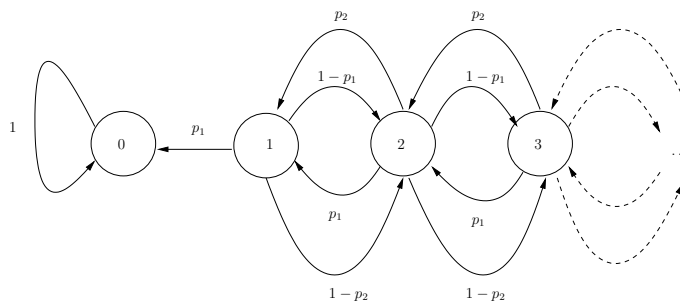


FIG. 2.1 – Une marche aléatoire avec plusieurs choix de successeurs

2.4.1 Exécutions

Comme le montre l'exemple de la section précédente, un PDM peut connaître plusieurs exécutions possibles, dépendantes de la séquence des actions choisies ainsi que des états atteints. En effet, fixer le choix des actions ou donner un algorithme calculant la prochaine action revient à lever un choix non déterministe. On va définir

une exécution comme un « enregistrement » de tous les couples états-actions atteints par le PDM.

Définition 2.4.2 (Exécution d'un PDM). Une exécution d'un PDM est une séquence infinie $\omega = s_0 a_0 s_1 a_1 \dots$ telle que $s_i \in S$, $a_i \in A(s_i)$ et $p_{s_i, s_{i+1}}(a_i) > 0$ pour tout $i \geq 0$.

De même que pour les chaînes de Markov, on notera par la suite $(X_n)_{n \in \mathbb{N}}$ la suite des états atteints et $(Y_n)_{n \in \mathbb{N}}$ la suite des actions du processus Markovien. On définit maintenant la notion de couple état-action :

Définition 2.4.3 (Couple état-action). Un couple état-action est un couple s, a tel que $s \in S$ et $a \in A(s)$. On notera $\chi_\Pi = \{(s, a) | X_0 = s_0 \wedge Y_0 = a_0 \wedge \dots \wedge X_n = s_n \wedge Y_n = a_n\}$ l'ensemble des paires états-actions du processus Markovien Π . Pour tout couple état-action (s, a) on note $Succ(s, a) = \{t | p_{s,t}(a) > 0\}$ l'ensemble des successeurs de s quand a est choisi.

Afin de pouvoir étudier de façon « probabiliste » le comportement des processus de décision markoviens, on étend la notion de tribu cylindrique des chaînes de Markov. Ainsi on va pouvoir rendre mesurables certains ensembles d'exécutions.

2.4.2 Ensembles mesurables d'exécutions

Pour tout état $s \in S$, soit $\mathcal{B}_s \subseteq 2^{\Omega_s}$ la plus petite algèbre de sous ensembles de Ω_s qui contient tous les cylindres de base :

$$\{\omega \in \Omega_s | X_0 = s_0 = s \wedge Y_0 = a_0 \wedge \dots \wedge X_n = s_n \wedge Y_n = a_n\}$$

pour tout $n \geq 0$, $s_0, \dots, s_n \in S$, $a_0 \in A(s_0), \dots, a_n \in A(s_n)$ et qui est close par complémentaire et par union et intersection dénombrable. Il s'agit bien d'une tribu σ -algèbre, comme nous l'avons définie dans la partie introductive aux probabilités.

2.4.3 Politiques

On entend par politique une manière de lever les choix non-déterministes liés à l'absence de description du choix des actions. Plusieurs versions très proches ont été introduites, telles celles présentées dans [Der70] et proches des adversaires de Segala et Lynch [SL95] et les ordonnanceurs de Lehman et Rabin [LR81], Vardi [Var85] et Pnueli et Zuck dans [PZ86].

Définition 2.4.4 (Politique). Une politique ϕ est un ensemble de probabilités conditionnelles $\mathcal{Q}_\phi(a | s_0 s_1 \dots s_n)$, définies pour tout $n \geq 0$, pour toute séquence d'états et pour tout $a \in A(s_n)$ et tel qu'on ait

$$0 \leq \mathcal{Q}_\phi(a | s_0 s_1 \dots s_n) \leq 1$$

et

$$\sum_{a \in A(s_n)} \mathcal{Q}_\phi(a | s_0 s_1 \dots s_n) = 1.$$

On peut ainsi calculer la probabilité d'une transition vers t sous une politique ϕ sachant le début d'une exécution $s_0 s_1 \dots s_n$:

$$P_{s_0}^\phi(t | s_0 s_1 \dots s_n) = \sum_{a \in A(s_n)} p_{s_n, t}(a) \mathcal{Q}(a | s_0 s_1 \dots s_n).$$

On mesure de cette façon la probabilité d'un début d'exécution sous la politique ϕ :

$$P_{s_0}(s_0 s_1 \dots s_n) = \prod_{i=0}^{n-1} p_{s_i, s_{i+1}}(a_i) \mathcal{Q}(a | s_0 s_1 \dots s_n).$$

Sous une politique ϕ fixée on définit ainsi une mesure de probabilité sur \mathcal{B}_s .

Le formalisme des processus de décision markoviens nous a intéressé grâce à la manière dont le non déterminisme est levé par les politiques. Dans cette thèse nous nous sommes également intéressés au temps moyen d'atteinte de certaines classes d'états et nous voulions être capable de calculer le pire temps moyen d'atteinte. Lucas de Alfaro a montré [dA97] que résoudre ce problème du calcul du pire temps moyen d'atteinte revenait à résoudre une instance du problème de plus court chemin stochastique et étudié dans [BT91]. C'est pourquoi par la suite nous allons brièvement parler du problème SSP et expliquer comment réduire un problème d'accessibilité à la résolution du problème SSP a donné naissance à une nouvelle génération de Model-checkers. Les résultats de Lucas De Alfaro résolvent en fait la question de la terminaison en temps moyen fini dans le cas où l'espace d'états est fini.

2.5 Problèmes classiques

2.5.1 Le problème du plus court chemin stochastique (SSP)

Le problème du plus court chemin stochastique est la version probabilisée du classique problème du plus court chemin. Résoudre le problème SSP consiste à déterminer la politique minimisant le coût moyen d'atteinte d'un ensemble d'états cibles R . Le coût total est calculé comme étant égal à la somme des coûts de chaque transition, qui est déterminé dans le cas général par une fonction $c : S \times Acts \rightarrow \mathbb{R}$.

Définition 2.5.1 (Fonction de coût). La fonction $c : S \times Acts \rightarrow \mathbb{R}$ est appelée fonction de coût des transitions, elle associe à chaque état $s \in S - R$ et à chaque action $a \in A(s)$ le coût $c(s, a)$.

Dans le cas général on définit une fonction de coût sur l'ensemble des états cibles,

Définition 2.5.2 (Fonction de coût terminale). La fonction $g : R \rightarrow \mathbb{R}$ est appelée fonction de coût terminale, elle associe à chaque $s \in R$ son coût terminal $g(s)$.

Définition 2.5.3 (Problème du plus court chemin stochastique (SSP)). Une instance de problème du plus court chemin stochastique (SSP) se représente par un 6-uplet (S, A, p, R, c, g) où (S, A, p) est un processus de décision Markovien, R un ensemble d'états cibles, c une fonction de coût et g une fonction de coût terminale.

Définition 2.5.4. On appelle instance positive –respectivement négative– du problème SSP une instance pour laquelle quelle que soit $s \in S$ et quel que soit $a \in A(s)$ on a $c(s, a) \geq 0$ –respectivement $c(s, a) \leq 0$.

Résoudre SSP revient à trouver une politique qui minimise l'espérance du coût d'atteinte de l'ensemble cible R , lorsqu'il existe des politiques permettant d'atteindre cet ensemble cible au bout d'un temps fini.

On appelle de telles politiques des politiques adéquates,

Définition 2.5.5 (Politique adéquate). Soit $T_R(\omega) = \min\{k | X_k(\omega) \in R\}$ le temps de la première visite dans R . Pour tout $s \in S$ on peut définir l'ensemble des politiques adéquates pour s par $Ad(s) = \{\phi | P_s^\phi(T_R < \infty) = 1\}$.

Le coût d'une politique adéquate est définie de la manière ci dessous,

$$v_s^\phi = E_s^\phi \left(g(X_{T_R}) + \sum_{k=0}^{T_R-1} c(X_k, Y_k) \right).$$

Résoudre SSP se résume à remplir les deux tâches, la première consiste à déterminer l'ensemble des états pour lesquels il existe une politique adéquate et calculer le coût minimum $v_s^* = \inf_{\phi \in Ad(s)} v_s^\phi$ d'une stratégie adéquate sur ces états.

Définition 2.5.6 (Politique optimale). Une politique ϕ est optimale si $v_s^\phi = v_s^*$.

2.5.2 Méthode de résolution de SSP

Les principales méthodes de résolution du problème SSP sont décrites dans [dA97, BT91] et [dA99]. La méthode classique consiste à utiliser les opérateurs de Bellman sous les hypothèses

- SSP1 : Il existe toujours une politique adéquate qui est Markovienne.
- SSP2 : Si ϕ est une politique Markovienne non adéquate alors $v_s^\phi = \infty$ pour au moins un état $s \in S - R$.

Théorème 2.5.1 (Equations de Bellman). Soit (S, A, p, R, c, g) une instance du problème SSP. Notons $[v_s]_{s \in S}$ un vecteur de réels et définissons la fonctionnelle L –opérateur de Bellman– par :

$$[Lv]_s = \min_{a \in A(s)} [c(s, a) + \sum_{t \in S} p_{s,t}(a)v_t + \sum_{t \in R} p_{s,t}(a)g(t)] \quad (2.13)$$

Si SSP1 et SSP2 sont vérifiées alors toutes les propositions suivantes sont vraies :

- La fonctionnelle L admet un point fixe v^\diamond telle que $Lv^\diamond = v^\diamond$.
- Le point fixe v^\diamond est la solution unique du problème de programmation linéaire suivant : Maximiser $\sum_{s \in R-S} v_s$ sous les contraintes :

$$v_s \leq c(s, a) + \sum_{t \in S-R} p_{s,t}(a)v_t + \sum_{t \in R} p_{s,t}(a)g(t) \quad (2.14)$$

- On a $v^* = v^\diamond$ pour tout $s \in S - R$.
- Si on considère une politique Markovienne n'autorisant pas que les actions qui réalisent le minimum de l'opérateur de Bellman, soit :

$$\mathcal{Q}_\phi(a, s) > 0 \Leftrightarrow a \in \arg \min_{a \in A(s)} \left\{ c(s, a) + \sum_{t \in S-R} p_{s,t}(a)v_t + \sum_{t \in R} p_{s,t}(a)g(t) \right\}$$

pour tout $s \in S - R$. Alors, cette politique ϕ est adéquate et on a $v_s^\phi = v_s^* = v_s^\diamond$.

En d'autres termes ce théorème explique de quelle façon résoudre le problème SSP, c'est-à-dire calculant un point fixe de L par une méthode itérative ou en résolvant le problème de programmation linéaire sous contrainte formulé par l'équation 2.14.

2.6 Vérification de propriétés sur des modèles probabilistes

2.6.1 Méthodes formelles

On appelle « méthodes formelles » l'ensemble des outils mathématiques et des critères permettant de savoir si certains modèles satisfont certaines propriétés. Parmi les méthodes formelles, on peut faire la distinction entre les méthodes automatiques de preuve de propriétés et celle qui ne sont pas automatisables ou qu'on ne sait pas automatiser.

En décrivant les processus de décision Markoviens, nous avons mentionné l'existence d'une méthode permettant de ramener le calcul des probabilités minimales et maximales d'accès à certaines classes d'états. On peut considérer ces critères comme des méthodes formelle pour la preuve de la propriété de récurrence.

Dans cette thèse nous fournissons un ensemble de méthodes formelles pour prouver la terminaison en temps moyen fini de programmes à base de règle de réécriture probabilistes, mais nous n'avons pas étudié l'automatisation de nos méthodes. Avant de résumer ces algorithmes et de parler de logiciels qui les implémentent, nous allons présenter un formalisme d'expression de propriétés sur les traces d'exécutions de systèmes modélisés par des processus de décision Markoviens.

L'aspect vérification de propriétés sur des systèmes probabilistes est un domaine très étendu. Cela englobe entre autres les notions d'accessibilité, le calcul de probabilité minimum et maximum de certaines classes d'évènements ainsi que le calcul du

nombre de transitions moyen nécessaire pour atteindre certaines classes d'états. On peut distinguer deux classes de méthodes de vérification : les méthodes déductives et les méthodes algorithmiques. Les méthodes algorithmiques ont connu un grand essor par le passé et ont permis, entre autres, le développement de ce qu'on appelle le model-checking ainsi que le développement d'outils et de logiciels de vérification de systèmes tels qu'APMC et PRISM [KNP02]. Les méthodes formelles, dans le cas général et les méthodes déductives en particulier permettent d'obtenir des résultats en prenant en considération une modélisation formelle du système étudié ou d'une classe de modèles. On peut alors obtenir des résultats sur des instances de systèmes où le nombre d'états est infini, comme c'est le cas par exemple avec l'application des résultats sur les chaînes de Markov pour l'étude de la théorie des files d'attente. Pour faire cela on considère une abstraction du modèle étudié pour rendre la modélisation plus aisée.

2.6.2 La vérification dans le cas général

La vérification a pour rôle principal de s'assurer que certains systèmes étudiés satisfont certaines propriétés. En d'autres termes, pour une modélisation \mathcal{A} d'un système et certaines propriétés ϕ , on cherche à savoir si la propriété ϕ est satisfaite dans le modèle \mathcal{A} . Lorsque c'est le cas, on note $\mathcal{A} \models \phi$ et éventuellement $\mathcal{A}, C \models \phi$ avec C un ensemble de conditions supplémentaires.

Le formalisme de modélisation, utilisé pour décrire les systèmes est choisi en fonction de la nature des systèmes modélisés. Dans le cas des systèmes probabilistes, on peut utiliser les chaînes de Markov et les Processus de Décision Markoviens pour modéliser des systèmes de transitions probabilistes. Il existe d'autres formalismes, tels les réseaux de Pétri stochastiques [FN85, Mol81, Bal01] et les chaînes de Markov à temps continu [Put94, Ros83] permettant de modéliser une grande variété de systèmes.

Il faut de même choisir ou définir une logique, pour exprimer de façon formelle les propriétés que l'on veut vérifier. Ainsi, pour spécifier des propriétés temporelles liées à l'exécution d'une chaîne de Markov à temps continu, on peut utiliser la logique CSL [BHK03], de même les logiques « pCTL » [BDA95] et « PTCTL » [KGSS02] sont utilisées pour énoncer des propriétés sur des processus de décision Markoviens.

Nous allons principalement nous intéresser au cas de la vérification de propriétés sur les chaînes de Markov et les processus de décision Markoviens.

2.6.3 Quelques propriétés importantes

Lorsqu'on étudie le comportement des systèmes probabilistes, tels les chaînes de Markov ou les processus de décision Markoviens, on cherche à savoir quels sont les états qui peuvent être atteints et parmi ces états on cherche à calculer la probabilité

qu'ils le soient. De même on peut être amené à avoir besoin de connaître le temps moyen d'atteinte d'une famille d'états, pourvu que celui-ci existe. On s'intéresse également aux propriétés classiques du model checking présentées dans [99].

De même, on peut s'intéresser à l'évaluation de certaines propriétés ϕ en fonction du temps, grâce à une formule en PTCTL par exemple. Dans ce dernier cas, on cherche à étudier les propriétés que vérifient l'arbre des exécutions possibles telles que :

- L'accès probabiliste : le système modélisé peut atteindre un ensemble d'états avec une probabilité maximale ou minimale.
- Temps d'accès probabilistiquement borné : le système atteint une certaine classe d'états en moins d'un certain temps avec probabilité supérieure à une constante supérieure à zéro.
- Coût d'atteinte probabilistiquement borné : le système atteint un ensemble d'états avec un coût inférieur à une constante avec probabilité supérieure à une autre constante.
- Invariance : le système ne quitte pas certains états avec une certaine probabilité minimale.
- Réponse bornée : le système atteint un certain ensemble d'états en moins d'une durée déterminée avec probabilité supérieure ou égale à une constante.
- Accessibilité en temps moyen fini.
- La terminaison en temps moyen fini, sujet d'étude principal de cette thèse. Il existe un rapport entre la terminaison en temps moyen fini et l'accessibilité en temps moyen fini des états terminaux à partir de tous les états du système modélisé. En effet, prouver qu'à partir de tout état d'un système on atteint un état terminal au bout d'un temps moyen fini revient à prouver la terminaison en temps moyen fini du système considéré.

En ce qui concerne l'évaluation de propriétés sur l'exécution d'un système :

- Probabilité qu'un événement ait lieu est supérieur, égale ou inférieur à une constante $a \in]0, 1[$.
- Une propriété ϕ est vérifiée jusqu'à ce qu'une propriété ψ le devienne.
- Vivacité : le fait qu'une propriété s'évalue à vrai au moins une fois.
- Le fait qu'il existe une branche pour laquelle une propriété soit satisfaite à partir d'un certain rang.
- Équité : le fait qu'il existe une branche pour laquelle une propriété est vérifiée infiniment souvent.

2.6.4 Aperçus des méthodes de vérifications de propriétés sur les Processus de décision Markoviens

Les premières méthodes de vérifications de formules sur les processus de décision Markoviens ont été présentés dans [CY95], où sont établies des bornes de complexité d'algorithmes de vérification de propriétés probabilistes du point de vue de la logique

LTL.

Cette approche a été étendue à d'autres logiques arborescentes, telles pCTL, pCTL* [BDA95], pTL et pTL*, obtenue en introduisant des horloges [dA97]. Ul-
térieurement, la notion d'équité a été introduite dans certaines logiques, avec des algorithmes permettant de vérifier ces propriétés [BK98, DA00].

Les résultats sur la vérification des processus de décision Markoviens peuvent être utilisés pour vérifier des propriétés sur des formalismes plus expressifs que les PDM. On peut mentionner le cas des automates temporisés probabilistes utilisés dans [KGSS02, KNS].

2.6.5 Logique probabiliste arborescente

Nous allons présenter de façon succincte, la définition de la logique pCTL (Probabilistic Computational Tree Logic), qui permet d'exprimer des propriétés sur les traces d'exécution d'un processus de décision Markovien.

On rappelle qu'un processus de décision Markovien est représenté par un triplet (S, A, p) où

- S est l'ensemble dénombrable des états,
- A est une fonction de S dans $Acts$, l'ensemble des actions possible. On peut considérer ces actions telles des propositions atomiques, en écrivant $a \in Acts$ s'évalue à vrai sur $s \in S$ si $a \in A(s)$.
- p qui associe à chaque couple état-action $(s, A(s))$ une distribution de probabilité sur les successeurs de s .

On considère les actions telles des proposition atomiques. La logique pCTL est obtenue à partir de la logique CTL en remplaçant les quantificateurs \forall et \exists par un opérateur P_{\bowtie} .

Définition 2.6.1 (formule pCTL). On définit la syntaxe des formules pCTL par induction :

- Si $\phi \in Acts$ alors ϕ est une formule de pCTL.
- Si ϕ et ψ sont des formules pCTL, alors $\neg\psi$ et $\phi \wedge \psi$ sont des formules de pCTL.
- Si ϕ et ψ sont des formules de pCTL, $a \in [0, 1]$ et $\bowtie \in \{<, \leq, >, \geq\}$ alors $A\phi U\psi$, $E\phi U\psi$ et $P_{\bowtie a}\phi U\psi$ sont des formules de pCTL.

On peut combiner les opérateurs A , E et U issus de la logique CTL*. Par exemple, l'opérateur $A\phi U\psi$ s'évalue à vrai dans un état $s \in S$ si tout chemin d'exécution issu de s comprend un préfixe constitué d'états qui satisfont ϕ et suivi d'au moins un état satisfaisant ψ .

L'opérateur $E\phi U\psi$ s'évalue à vrai dans un états s s'il existe un chemin d'exécution issu de s comprenant un préfixe dont les états satisfont ϕ suivi d'au moins un état satisfaisant ψ . Le troisième opérateur, $P_{\bowtie}U$. fait intervenir les politiques de la manière suivante : $P_{\bowtie a}\phi U\psi$ est vrai si pour toute politique la probabilité p que le processus

stochastique associé à un chemin d'exécution issu de s comprend un préfixe dont les états satisfont ϕ et suivi d'un état qui satisfait ψ avec $p \bowtie a$.

Cette logique permet d'exprimer des propriétés fines sur le comportement de processus stochastiques sous politiques, notamment des comportements asymptotiques.

2.6.6 Model checking de pCTL

Le modèle checking de formule pCTL, sur un modèle décrit à l'aide de processus de décision Markoviens, s'effectue à l'aide d'un algorithme polynomial en la taille du modèle. En effet, pour vérifier si ϕ est satisfaite sur le modèle, l'algorithme de vérification procède par évaluation successive des sous formules de ϕ en remontant l'arbre syntaxique de la formule ϕ des feuilles vers la racine, en étiquetant chaque état avec les sous-formules qu'il vérifie. Cette procédure de marquage est linéaire en la taille du modèle, exactement comme dans le cas de la vérification des formule CTL. Cependant, pour vérifier que les opérateurs P_{\bowtie} s'évaluent à vrai ou à faux, il faut résoudre un problème de programmation linéaire, qui se résout grâce à un algorithme polynomial en la taille du modèle. On trouve une description de cet algorithme dans [KNS03].

2.7 Logiciels de model-checking probabiliste

L'approche proposée par la communauté model-checking consiste à fournir un formalisme de modélisation de systèmes, un logiciel de « model-checking » qui vérifie si certaines propriétés ou formules sont satisfaites par le modèle donné en entrée. On peut citer par exemple deux « models checkers » APMC [HLMP03] et PRISM [KNP02, HKNP06, PRI], le premier permettant de vérifier de façon approchée la probabilité qu'une formule LTL monotone soit vérifiée sur une chaîne de Markov et le second permettant de vérifier des propriétés probabilistes exprimées sous la forme de formule PCTL ou CSL sur des systèmes représentés grâce à des automates temporisés probabilistes. Le premier utilise des méthodes de Monte-Carlo pour approcher la probabilité de satisfaction de propriétés monotone alors que le second ramène le problème de satisfaction de formules pCTL à la résolution d'une instance du problème du « Stochastic Shortest Path » [BT91]. Ces deux méthodes ne peuvent s'appliquer que sur des systèmes où l'espace des états est fini. Cependant, les formalisme de description de modèle utilisés par les « model checkers » ont atteint une certaine maturité depuis leur introduction [CE81, Eme81, QS82] et les logiciels développés permettent de vérifier des propriétés sur des systèmes concrets et complexes. Néanmoins, de tels outils butent toujours sur le problème de l'explosion combinatoire quand la taille des modèles augmentent. En effet, un logiciel de model-checking génère de façon exhaustive l'ensemble des états potentiellement accessible par le système modélisé.

Pour diminuer l'impact du problème de « l'explosion combinatoire du nombre d'états » des systèmes modélisés, de nombreuses méthodes ont été développées, telles les abstractions qui permettent de regrouper l'ensemble de toutes les configurations dans certains sous ensembles et de ne travailler que sur les représentants de ces ensembles.

PRISM représente les systèmes donnés en entrée sous la forme d'un processus de décision Markovien, représentant toutes les exécutions possibles du modèle donné en entrée. Le nombre d'états de ce processus de décision Markovien croît exponentiellement avec la taille du modèle étudié. Malgré les excellentes optimisations utilisées par ce model checker, le problème de l'explosion combinatoire du nombre d'états générés en fonction de la taille du modèle demeure inévitable et empêche la validation des systèmes au delà d'une certaine taille en utilisant ces méthodes.

Le logiciel APMC, en utilisant des méthodes de Monte Carlo ne permet que d'obtenir des solutions approchées de propriétés quantitatives mais avec un coût algorithmique moindre qu'avec une génération exhaustive de toutes les configurations possibles du système. Ces optimisations ont été obtenues entre autres grâce à l'utilisation d' ϵ -abstractions probabilistes [Pey03]. Ces deux approches complémentaire ont mené à la fusion des outils APMC et PRISM.

Deuxième partie

Réécriture probabiliste

3

Les Systèmes Abstrait de Réduction Probabilistes

3.1 Présentation du modèle

Nous allons dans ce chapitre définir une extension probabiliste des systèmes abstraits de réduction, qui va nous permettre de modéliser le comportement de systèmes où se combinent des comportements non déterministes et probabilistes. Ces objets vont nous permettre de définir les systèmes de réécriture probabilistes, de la même manière que les systèmes abstraits de réduction servent à décrire les systèmes de réécriture classique. Comme par la suite nous allons modéliser les systèmes de réécriture probabiliste comme étant une instance des systèmes abstraits de réduction probabiliste, l'étude des différentes classes de terminaison de ces systèmes va nous apporter les principaux critères permettant de certifier que les systèmes de réécriture probabiliste vérifient certaines propriétés de terminaison.

De la même manière que les systèmes abstraits de réduction peuvent se décrire sous la forme de systèmes de transition, on peut établir une correspondance entre les systèmes abstraits de réduction probabilistes et les Processus de Décision Markoviens [Put94]. Nous allons voir, que l'on peut en effet construire une correspondance entre les systèmes abstraits de réduction probabilités et les processus de décision markoviens. L'une des différence qu'il y a entre les deux systèmes, est que l'on ne nomme pas d'actions sur les systèmes de réduction probabilités et qu'il existe des états qui ne sont en relation avec aucun autre, y compris eux-mêmes. L'idée générale des systèmes abstraits de réduction probabilistes est que l'on met en relation certains éléments d'un ensemble A avec une distribution de probabilité sur les éléments de A , permettant de tirer un successeur suivant cette loi de probabilité.

Définition 3.1.1 (Système Abstrait de Réduction Probabiliste, noté PARS). Soit un ensemble dénombrable S , on note $Dist(S)$ l'ensemble des distributions de probabilité sur S , c'est à dire l'ensemble des mesures μ sur S satisfaisant $\sum_{i \in S} \mu(i) = 1$.

Un *système abstrait de réduction probabiliste* aussi noté *PARS* pour *Probabilistic Abstract Reduction System* est un couple $\mathcal{A} = (A, \rightarrow)$ où A est un ensemble dénombrable et \rightarrow une relation $\subset A \times \text{Dist}(A)$.

Un PARS est dit *déterministe* si pour tout élément a de A , il existe au plus une distribution μ telle que $a \rightarrow \mu$.

Un état $a \in A$ tel qu'il n'y ait aucune distribution de probabilité μ vérifiant $a \rightarrow \mu$ est appelé *terminal*.

Avant de présenter un premier exemple, présentons avec quelle manière on représente les distributions sur un espace d'états.

Notation 6 (Distribution). *Soit μ la distribution de probabilité sur un espace A qui associe chaque élément a_i de A la probabilité p_i . On écrit cette distribution de la façon suivante :*

$$\mu = \left\{ \begin{array}{l} a_1 : p_1 \\ \vdots \\ a_k : p_k \\ \vdots \\ a_n : p_n \end{array} \right. ,$$

ou de la façon suivante :

$$\mu = \{p_1 : a_1 | \dots | a_k : p_k | \dots | a_n : p_n\}.$$

La seconde notation permet de décrire de façons concise les distributions quand elle chargent peu d'éléments de l'espace.

Exemple 3.1.1 (Un PARS). Considérons le PARS similaire à l'exemple 2.1 du chapitre précédent formé par l'ensemble des entiers naturels et la relation de transition mettant en relation chaque entiers n non nul avec les autres avec la distribution $\mu_{1,n}$ et $\mu_{2,n}$, où

$$\begin{aligned} \mu_{1,n} &= \left\{ \begin{array}{l} n-1 : p_1 \\ n+1 : 1-p_1 \end{array} \right. \\ \mu_{2,n} &= \left\{ \begin{array}{l} n-1 : p_2 \\ n+1 : 1-p_2 \end{array} \right. \end{aligned}$$

et pour tout $n \geq 1$ on $n \rightarrow \mu_{1,n}$ et $n \rightarrow \mu_{2,n}$. L'état zéro, n'est en relation avec aucune distribution de probabilité, il est par définition terminal.

Le PARS décrit dans exemple 3.1.1 modélise l'ensemble des systèmes de transitions où de chaque entiers n on peut choisir de transiter vers un successeur suivant soit la distribution $\mu_{1,n}$ ou la distribution $\mu_{2,n}$. La description de ce modèle ne donne aucune contrainte quand au choix de la distribution $\mu_{1,n}$ ou $\mu_{2,n}$ que doit suivre le tirage du

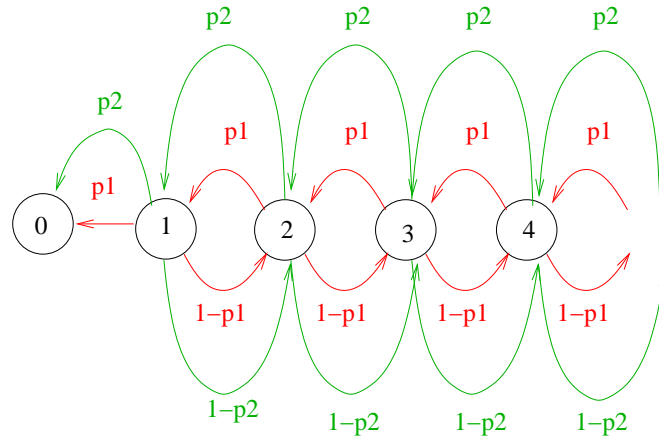


FIG. 3.1 – Une marche aléatoire non déterministe.

successeur, c'est à ce niveau que l'on trouve l'expression du non déterminisme dans le modèle des PARS. Nous devons à présent expliquer comment de tels systèmes se comportent, et pour cela nous devons dans un premier temps introduire quelques notions,

Définition 3.1.2 (Histoire). Une histoire de longueur $n + 1$ est une séquence de $n + 1$ états de A , noté a_0, \dots, a_n . Une telle histoire est dite non terminale si le dernier état atteint a_n n'est pas terminal.

Définition 3.1.3 (Stratégie déterministe). Une stratégie ϕ est une fonction de l'espace des histoires non terminales de longueur finies vers l'ensemble des distributions $Dist(A)$ sur l'espace A telle que pour toute histoire a_0, \dots, a_n , si $\phi(a_0, \dots, a_n) = \mu$ alors $a_n \rightarrow \mu$. Une histoire a_0, \dots, a_n est dite réalisable sous la stratégie ϕ si pour tout $i < n$ on a $\phi(a_0, \dots, a_i)[a_{i+1}] > 0$, c'est à dire que la probabilité qu'on a de choisir a_{i+1} en connaissant les i premières étapes et en suivant la stratégie ϕ est positive. Lorsque nous définirons et étudierons les systèmes de réécriture probabiliste, nous montrerons que cette notion de stratégie est très proche de celle de stratégie de réécriture.

De même, si on désire pouvoir choisir aléatoirement une distribution en relation avec l'état courant à l'instar des politiques pour les processus de décision Markovien, alors il convient d'étendre la précédente définition de stratégie de la manière suivante :

Définition 3.1.4 (Stratégie aléatoire). Une stratégie aléatoire ϕ est une fonction de l'espace des histoires non terminales de longueur finie vers l'ensemble des distributions $Dist(Dist(A))$ sur l'espace $Dist(A)$ telle que pour toute a_0, \dots, a_n , $\phi(a_0, \dots, a_n) = Q$ et Q est une distribution sur l'espace des distributions μ telle que $a_n \rightarrow \mu$. Une histoire a_0, \dots, a_n est dite réalisable sous la stratégie aléatoire ϕ si pour tout $i < n$ il existe d'une part une distribution μ telle que $a_i \rightarrow \mu$ et $\phi(a_0, \dots, a_i)[\mu] > 0$ et d'autre part $\mu[a_{i+1}] > 0$.

Remarque 3.1.1. Pour exprimer les stratégies déterministes avec les stratégies aléatoire, il suffit de ne travailler qu'avec des stratégies aléatoires ϕ telles que pour toute histoire réalisable sous ϕ a_1, \dots, a_n et pour tous $a_n \rightarrow \mu$ on ait $\phi(a_0, \dots, a_n)[\mu] = 0$ ou $\phi(a_0, \dots, a_n)[\mu] = 1$.

Exemple 3.1.2 (Quelques stratégies simples). Énumérons quelques exemples très simples de stratégies s'appliquant au PARS présenté dans l'exemple 3.1.1 :

1. Soit ϕ_1 la stratégie telle que, pour toute histoire a_0, \dots, a_n on ait $\phi_1(a_0 \dots a_n) = \mu_{1,a_n}$,
2. ϕ_2 la stratégie telle que pour toute histoire $h = a_0 \dots a_{2n+1}$ on ait $\phi_2(h) = \mu_{1,a_{2n+1}}$ et $h = a_0 \dots a_{2n}$ $\phi_2(h) = \mu_{2,a_{2n}}$

Remarque 3.1.2. La notion de stratégie subvient au besoin que l'on a de faire des choix là où se tient le non-déterminisme pour étudier le comportement des dérivations en fonction de ces choix. On peut classer ces stratégies en fonction de la nature de l'algorithme de choix :

- Markovien, si le choix de la distribution en relation avec l'état courant et décrivant le choix du successeur ne dépend pas des états atteints avant l'état courant,
- Déterministe si la stratégie l'est,
- Aléatoire si la stratégie l'est.

Étudier le comportement d'un PARS, consiste à étudier les propriétés de ses traces d'exécution. Par exemple, savoir si un PARS termine, revient à savoir que toutes les traces d'exécution possibles de ce PARS mènent à un état terminal. De même, savoir s'il existe des stratégies permettant à certaines traces d'atteindre une certaine classe d'états, en particulier d'atteindre des états terminaux est une question essentielle. Pour répondre à ce type de problème, on étudie un objet que l'on nomme *dérivation*, où une dérivation est une séquence stochastique où les choix non déterministes sont fait grâce à une stratégie ϕ fixée et les choix probabilistes suivent les distributions choisis par la stratégie ϕ . D'autre part, une dérivation qui arrive sur un état terminal ne progresse plus. Formellement, pour pouvoir parler de suite stochastique, nous allons supposer qu'au temps ultérieurs, le dérivation sera sur un état \perp stable.

Définition 3.1.5 (Dérivation). Une *dérivation* π du PARS \mathcal{A} sous une stratégie déterministe ϕ est une suite de variables aléatoires $\pi = (\pi_i)_{i \in \mathbb{N}}$ telle que,

$$\begin{aligned} P(\pi_{n+1} = \perp | \pi_n = \perp) &= 1, \\ P(\pi_{n+1} = \perp | \pi_n = s) &= 1 \quad \text{si } s \in A \text{ est terminal,} \\ P(\pi_{n+1} = \perp | \pi_n = s) &= 0 \quad \text{si } s \in A \text{ est non-terminal,} \end{aligned}$$

Et quel que soit $t \in A$

$$P(\pi_{n+1} = t | \pi_n = a_n, \pi_{n-1} = a_{n-1}, \dots, \pi_0 = a_0) = \mu(t)$$

dès lors que $a_0 a_1 \cdots a_n$ est une histoire non terminale et réalisable et $\mu = \phi(a_0 \dots a_n)$

Remarque 3.1.3. Les dérivations sont homogènes et Markovienne lorsque la stratégie ϕ suivie est Markovienne, c'est à dire que la valeur $\phi(a_0 \dots a_n)$ ne dépends que de a_n .

On peut étendre la notion de dérivation au cas des stratégies probabilistes. Nous ne le feront pas dans ce chapitre et nous nous focaliserons dans ce chapitre sur les stratégies déterministes.

3.2 Terminaison des Systèmes Abstrait de Réduction Probabiliste

Si une dérivation π vérifie à un rang n fini $\pi_n = \perp$, alors presque sûrement pour tout $m \geq n$ $\pi_m = \perp$. On qualifie une telle dérivation de *terminante*. Une dérivation qui ne termine pas vérifie la propriété suivante, $\pi_n \neq \perp$ pour tout entier n , presque sûrement.

Définition 3.2.1 (Terminaison presque sûre). Un PARS $\mathcal{A} = (A, \rightarrow)$ est dit presque sûrement terminant si et seulement si pour toute stratégie ϕ , la probabilité que toute dérivation $\pi = (\pi_i)_{i \in \mathbb{N}}$ termine sous la stratégie ϕ vaut 1. Formellement, quelque soit la stratégie ϕ suivie par π $P(\exists n | \pi_n = \perp) = 1$. On notera qu'un tel PARS est *a.s* terminant, pour *almost surely*.

La notion de terminaison presque sûre garantit qu'à partir de toute dérivation π commençant à partir d'un terme a arbitrairement choisi et suivant une stratégie quelconque ϕ , la variable aléatoire $\tau[a, \phi]$ associée à π -où $\pi_0 = a$ - à valeur dans $\mathbb{N} \cup \{+\infty\}$, telle que $\tau[a, \phi] = \infty$ si $\forall n \in \mathbb{N} \pi_n \neq \perp$ et $\tau[a, \phi] = \min\{n | \pi_n = \perp\}$ sinon est finie avec probabilité 1.

Proposition 3.2.1. *Un PARS $\mathcal{A} = (A, \rightarrow)$ est presque sûrement terminant si et seulement si pour toute les stratégies ϕ et tout état $a \in A$, $P(\tau[a, \phi] = +\infty) = 0$.*

Remarque 3.2.1. Cette notion, bien qu'intéressante en théorie, s'avère être trop faible dans des cas d'étude pratique. En effet, dans de nombreux domaines, notamment en qualité de service, on attend des systèmes probabilistes qu'ils satisfassent certaines propriétés en un temps moyen fini. C'est en particulier le cas lorsque l'on étudie la terminaison d'algorithmes probabilistes. Savoir qu'ils terminent avec probabilité 1 est certes intéressant, mais cette terminaison presque sûre n'entraîne pas la finitude du temps moyen d'exécution d'un tel algorithme. C'est pour cela que nous introduisons la notion de *terminaison presque sûre positive*, ou en d'autres termes la finitude du temps moyen de terminaison.

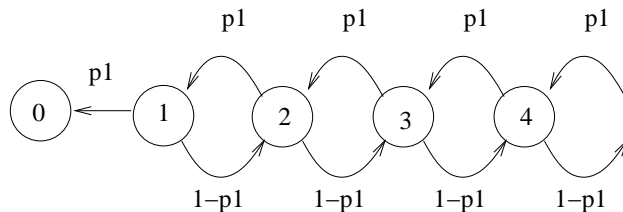


FIG. 3.2 – Pourquoi introduire la notion de terminaison positive et presque sûre.

Considérons pour cela un exemple élémentaire et associons y la question suivante, *Combien de transitions faut il en moyenne pour atteindre l'état 0 à partir d'un autre état ?* L'étude des chaînes de Markov nous enseigne que la réponse dépend de la valeur du paramètre p_1 , et qu'en fonction de sa valeur, la moyenne existe ou non (voire la section 2.3.3),

p_1	$P(\text{termine})$	$E[\text{nb_transition}]$
$< \frac{1}{2}$	< 1	∞
$= \frac{1}{2}$	$= 1$	∞
$> \frac{1}{2}$	$= 1$	$< \infty$

Si on considère le cas où la valeur de p_1 égale $\frac{1}{2}$, on constate que la probabilité d'atteindre l'état 0 en un certain nombre de transitions vaut 1, mais le nombre de transitions moyen requis pour atteindre 0 est infini. Cependant, dès que la valeur de $p_1 > \frac{1}{2}$ alors avec probabilité 1 on atteint l'état terminal et d'autre part le nombre moyen de transitions nécessaires pour atteindre 0 est fini. Ce cas particulier de terminaison en temps moyen fini est beaucoup plus intéressant que le précédent, car on a une information sur le temps moyen requis pour terminer. Passons maintenant à la définition formelle de la terminaison presque sûre positive :

Définition 3.2.2 (Terminaison presque sûre positive). Un PARS $=(A, \rightarrow)$ est positivement presque sûrement terminant (+a.s) si pour toute stratégie ϕ , pour tout état $a \in A$, $T[a, \phi]$ existe et est fini, où

$$T[a, \phi] = E[\tau(a, \phi)]$$

est le temps moyen de terminaison à partir de l'état a et suivant la stratégie ϕ .

Comme expliqué dans la partie introductive aux probabilités, si $P(\tau[a, \phi] = \infty) > 0$ alors l'espérance de $\tau[a, \phi]$ n'existe pas, c'est à dire n'est pas une valeur finie. Le fait que $T[a, \phi]$ existe implique que $P(\tau[a, \phi] = \infty) = 0$, ce qui nous amène à énoncer la proposition suivante :

Proposition 3.2.2. *Un PARS positivement presque sûrement terminant est presque sûrement terminant .*

Démonstration. Si $E[\tau[a, \phi]] < \infty$ alors $P(\tau[a, \phi] < \infty) = 1$. Cela signifie bien que la probabilité qu'une dérivation quelconque ait une longueur finie vaut 1. \square

Proposition 3.2.3. *Les systèmes abstraits de réduction probabiliste sont une généralisation des systèmes abstraits de réduction. Ils permettent en effet de décrire de façon simple les systèmes abstraits de réduction. Pour faire cela, il suffit de décrire le choix des successeurs à l'aide de distributions de Dirac, qui chargent un et un seul état avec probabilité 1.*

Démonstration. Pour montrer cela, nous allons construire un PARS exprimant un ARS. Soit $\mathcal{A} = (A, \rightarrow)$ un ARS. Pour tout a et b états de A , tels que $a \rightarrow b$, on construit la relation $\overset{\prime}{\rightarrow}$ satisfaisant $a \overset{\prime}{\rightarrow} \{b : 1\}$. On met ainsi en relation l'état a avec la distribution de probabilité qui charge l'état b avec probabilité 1. Le PARS $\mathcal{A}' = (A, \overset{\prime}{\rightarrow})$ simule bien toutes les règles de \mathcal{A} et seulement celles là. \square

Exemple 3.2.1. Soit l'ARS $\mathcal{A} = (\mathbb{N}, \rightarrow)$ avec \rightarrow défini de la façon suivante :

$$\begin{array}{lcl} n + 1 & \rightarrow & n \\ n & \rightarrow & n + 1 \\ n & \rightarrow & 5 \end{array}$$

L'ARS \mathcal{A} s'exprime sous la forme du PARS $\mathcal{A}' = (\mathbb{N}, \overset{\prime}{\rightarrow})$ avec $\overset{\prime}{\rightarrow}$:

$$\begin{array}{lcl} n + 1 & \overset{\prime}{\rightarrow} & \{n : 1\} \\ n & \overset{\prime}{\rightarrow} & \{n + 1 : 1\} \\ n & \overset{\prime}{\rightarrow} & \{5 : 1\} \end{array}$$

Corollaire 3.2.1 (Indécidabilité de la terminaison des PARS). *La terminaison des PARS est indécidable.*

Démonstration. Puisque les PARS expriment les ARS, et que le problème de la terminaison des ARS est indécidable [BN98], alors le problème de la terminaison des PARS est indécidable. \square

3.3 Prouver la terminaison presque sûre positive

Dans cette partie nous présentons les techniques de preuves que nous avons mise au point et présentées dans [BG05], pour montrer la terminaison presque sûre positive des PARS. Ces techniques s'inspirent très fortement de la proposition 1.3.1. La proposition 3.2.3 permet de donner l'intuition que nous avons suivie pour établir nos résultats. En effet, si la méthode de preuve fonctionne sur les PARS, elle doit à priori s'appliquer sur les ARS, puisque les ARS sont une instance des PARS.

Théorème 3.3.1 (Correction). *Un PARS $\mathcal{A} = (A, \rightarrow)$ est +a.s terminant s'il existe une fonction $V : A \rightarrow \mathbb{R}$, vérifiant $\inf_{i \in A} V(i) > -\infty$, un $\epsilon > 0$, tel que pour tout état $a \in A$, tout μ avec $a \rightarrow \mu$, le drift évalué en a suivant μ défini par*

$$\Delta_\mu V(a) = \sum_i \mu(i)V(i) - V(a)$$

vérifie

$$\Delta_\mu V(a) \leq -\epsilon.$$

Démonstration. Nous allons appliquer le théorème 2.3.6 pour montrer que sous les hypothèses du théorème, le PARS atteint un état terminal au bout d'un nombre de transition fini, quelque soit la stratégie choisissant de choix du successeur.

On étend V à $A \cup \{\perp\}$ avec $V(\perp) = 0$, et sans perte de généralités, on suppose que $\inf_{a \in A} [V(a)] = C > \epsilon$. En effet, si C est inférieur à ϵ , il suffit de considérer la fonction $V'(t) := V(t) + K$ avec $K > -C + \epsilon$ pour se ramener à ce cas. Pour tout $a \in A$ et pour tout $a \rightarrow \mu$ le drift de V' évalué en a suivant μ est égal au drift de V évalué en a et suivant μ , en effet :

$$\begin{aligned} \Delta_\mu V'(a) &= \sum_{t \in A} \mu(t)[V(t) + K] - [V(a) + K] \\ \Delta_\mu V'(a) &= \sum_{t \in A} \mu(t)[V(t)] - V(a) + K - K \\ \Delta_\mu V'(a) &= \sum_{t \in A} \mu(t)[V(t)] - V(a) = \Delta_\mu V(a). \end{aligned}$$

Soit $a_0 \in A$ un élément de A . Posons $S_0 = V(a_0)$ et soit ϕ une stratégie. On pose $S_n = V(\pi_n)$ avec π_n le n -ième successeur de a_0 de la dérivation $(\pi)_{n \in \mathbb{N}}$ partant de a_0 et suivant ϕ . On note τ le premier temps d'entrée de $(S_n)_{n \in \mathbb{N}}$ dans l'intervalle $[0, \epsilon]$ qui correspond également au plus petit entier $n + 1$ tel que $\pi_n = s$ avec s un état terminal. Si $\pi_n = s$ est terminal alors le drift de V en π_n est inférieur ou égal à $-C$. La variable aléatoire τ satisfait la propriété suivante $\forall n > \tau \pi_n = \perp$ et $V(\pi_n) = 0$.

Comme pour toute stratégie ϕ , pour tout $a \in A$ non terminal on a $\Delta_{\phi(a)} V(a) \leq -\epsilon$, alors si $\pi_n = a_n$ est non terminal on a

$$E[V(\pi_{n+1}) | \pi_n = a_n, \dots, \pi_0 = a_0] \leq V(a_n) - \epsilon,$$

c'est à dire

$$E[S_{n+1} | \pi_n = a_n, \dots, \pi_0 = a_0] \leq S_n - \epsilon \times 1_{\{\tau > n\}}.$$

En rappelant qu'on note $\tilde{S}_n = S_{n \wedge \tau}$, et en considérant la filtration $\mathcal{F}_n = \sigma(\pi_0, \dots, \pi_n)$ on a l'inégalité suivante

$$E[\tilde{S}_{n+1} | \mathcal{F}_n] \leq \tilde{S}_n - \epsilon \times 1_{\{\tau > n\}},$$

car si $\tau > n$ on a $S_n = \tilde{S}_n$ et si $\tau \leq n$ on a $\tilde{S}_{n+1} = \tilde{S}_n = S_\tau = 0$.

Donc d'après le théorème 2.3.6

$$E[\tau] \leq \frac{S_0}{\epsilon}$$

ce qui montre bien que sous ces hypothèses les dérivation d'un PARS ont longueur moyenne bornée.

□

Ce résultat peut se voir comme une conséquence directe du théorème 5.3.1, plus général, que nous énonçons et démontrerons dans le chapitre 5 traitant de la terminaison sous stratégies. Nous allons maintenant montrer que la réciproque de ce résultat est vrai dans certains cas particuliers.

Définition 3.3.1. Un système abstrait de réduction probabiliste $\mathcal{A} = (A, \rightarrow)$ est dit finiment branchant si pour tout état $a \in A$ il existe au plus un nombre fini de distributions μ telles que $a \rightarrow \mu$.

Théorème 3.3.2 (Complétude des systèmes finiments branchants). *Si un PARS finiment branchant $\mathcal{A} = (A, \rightarrow)$ est +a.s terminant, et que A est fini, alors il existe une fonction $V : A \rightarrow \mathbb{R}$ telle que $\inf_{i \in A} V(i) > -\infty$ et il existe $\epsilon > 0$ tel que $a \in A$, pour tout μ tel que $a \rightarrow \mu$, le drift évalué en a suivant μ défini par*

$$\Delta_\mu V(a) = \sum_i \mu(i)V(i) - V(a)$$

vérifie

$$\Delta_\mu V(a) \leq -\epsilon.$$

Nous allons prouver ce théorème en procédant en plusieurs étapes. En premier lieu, remarquons que par hypothèse, étant donné tout état a , toute stratégie ϕ , le temps moyen pour atteindre \perp en partant de a et en suivant la stratégie ϕ , vérifie $T[a, \phi] < \infty$.

De même si h est une histoire réalisable, et que l'on note par $T[h, \phi]$ le temps moyen pour atteindre un état terminal depuis le dernier état de l'histoire h , alors on a la propriété suivante

$$T[h, \phi] = 1 + \sum_{x \in A} \phi(h)(x)T[hx, \phi] \tag{3.1}$$

Toujours pour écrire la preuve du théorème 3.3.2, nous allons utiliser le lemme suivant :

Lemme 3.3.1. *Lorsque la stratégie ϕ est Markovienne, on a $T[hx, \phi] = T[x, \phi]$ ce qui implique grâce à l'équation 3.1*

$$T[x, \phi] = 1 + \sum_{x' \in A} \phi(x)(x')T[x', \phi] \quad \text{pour tout } x \text{ non terminal.} \tag{3.2}$$

$$T[x, \phi] = 0 \text{ si } x \text{ est terminal.} \tag{3.3}$$

Pour alléger l'écriture des équations, on notera dorénavant ϕ_x au lieu de $\phi(x)$. L'équation 3.2 s'écrit de la façon suivante :

$$T[x, \phi] = 1 + \sum_{x' \in A} \phi_x(x')T[x', \phi]. \quad (3.4)$$

Maintenant, on peut énoncer le resultat sur lequel s'articule la preuve :

Lemme 3.3.2. *S'il existe une stratégie Φ Markovienne telle que pour tout $a \rightarrow \mu$,*

$$\sum_{x'} \mu(x')T[x', \Phi] \leq \sum_{x'} \Phi_a(x')T[x', \Phi]$$

alors on a les conclusions du théorème 3.3.2.

Démonstration. Puisque Φ est Markovienne, on sait que

$$T[x, \Phi] = 1 + \sum_{x' \in A} \Phi_x(x')T[x', \Phi].$$

Pour un $a \in A$ quelconque, cela peut aussi s'écrire

$$T[a, \Phi] = 1 + \sum_{x' \in A} \Phi_a(x')T[x', \Phi].$$

Considérons la fonction $V : A \rightarrow \mathbb{R}$ telle que $V(a) = T[a, \Phi]$ pour tout état a , et $\epsilon = 1$. V est par construction positive, et satisfait la propriété suivante :

Pour tout couple $a \rightarrow \mu$,

$$\begin{aligned} \Delta_\mu V(a) &= \sum_{x' \in A} \mu(x')V(x') - V(a) \\ &= \sum_{x' \in A} \mu(x')T[x', \Phi] - T[a, \Phi] \\ &= \sum_{x' \in A} \mu(x')T[x', \Phi] - 1 - \sum_{x' \in A} \Phi_a(x')T[x', \Phi] \\ &= -1 + (\sum_{x' \in A} \mu(x')T[x', \Phi] - \sum_{x' \in A} \Phi_a(x')T[x', \Phi]) \\ &\leq -1 \end{aligned}$$

□

Considérons une politique Markovienne ϕ_i . Soit S l'ensemble des états a tels que

$$T[a, \phi_i] < \max_{a \rightarrow \mu} (1 + \sum_{x'} \mu(x')T[x', \phi_i]).$$

Si S est vide, puisque $T[a, \phi_i] = 1 + \sum_{x'} \phi_{i_a}(x')T[x', \phi_i]$, cela veut dire que pour $\Phi = \phi_i$ on a

$$\sum_{x'} \mu(x')T[x', \Phi] \leq \sum_{x'} \Phi_a(x')T[x', \Phi]$$

pour tout $a \rightarrow \mu$, et donc on a les hypothèses du lemme 3.3.2.

Si S n'est pas vide, choisissons a quelconque dans S , et posons ϕ_{i+1} la stratégie telle que $\phi_{i+1_x} = \phi_{i_x}$ pour tout $x \neq a$, et

$$\phi_{i+1_a} = \operatorname{argmax}_{\{\mu | a \rightarrow \mu\}} (1 + \sum_{x'} \mu(x')T[x', \phi_i]).$$

Par construction, ϕ_{i+1} est Markovienne, si ϕ_i l'est.

Propriété 3.3.1 ((*)). Soit (*) la propriété suivante : $T[x, \phi_i] \leq T[x, \phi_{i+1}]$, pour tout x , avec au moins un état x avec l'inégalité stricte, pour tout i .

Supposons la propriété (*) vraie. En partant de ϕ_0 une politique markovienne quelconque, on obtient donc soit une stratégie ϕ_i qui permet de conclure, ou une suite infinie de stratégies $(\phi_i)_i$ avec $T[x, \phi_i] \leq T[x, \phi_{i+1}]$, pour tout x , avec au moins un état x avec l'inégalité stricte.

Si le nombre de stratégies markovienne est borné (par exemple sur un espace fini dans le cas finiment branchant) cela n'est pas possible, donc on conclue.

Remarque 3.3.1. Lorsque l'espace d'états est infini, le nombre des stratégies Markovienne peut devenir infini. Il suffit de considerer l'exemple 3.1.1 et la stratégie ϕ_0 qui à tout état n renvoie la distribution $\mu_{1,n}$ dans le cas où $p_1 < p_2$ pour voir qu'on peut construire une suite infinie de stratégies $(\phi)_{i \in \mathbb{N}}$ telle que $T[x, \phi_i] \leq T[x, \phi_{i+1}]$.

Maintenant, nous allons montrer que la propriété (*) est vraie.

Soit A' le sous-ensemble de A constitué des états non-terminaux.

Pour $X : A' \rightarrow \mathbb{R}$ une fonction, et ϕ une stratégie, notons L_ϕ pour la transformation qui envoie X sur une autre fonction $L_\mu X : A' \rightarrow \mathbb{R}$ définie par

$$L_\phi X(a) = 1 + \sum_{x' \in A'} \phi_a(x') X(x')$$

Si on pose M_ϕ la matrice des $\phi_a(x')$, cela s'écrit sous forme vectorielle

$$L_\phi X = 1 + M_\phi X,$$

où 1 est le vecteur constitué de 1.

L'équation 3.4 s'écrit alors

$$T[x, \phi] = L_\phi T[x, \phi],$$

où $T[x, \phi]$ dénote la fonction qui à $x \in A'$ associe $T[x, \phi]$, et ϕ est Markovienne.

Pour deux fonctions $X, X' : A' \rightarrow \mathbb{R}$, notons $X \leq X'$ lorsque $X(a) \leq X'(a)$ pour tout $a \in A'$.

On a les fait suivants :

Lemme 3.3.3. – Si $X \leq X'$, alors $L_\phi X \leq L_\phi X'$.

$$- (L_\phi)^n X = (M_\phi)^n X + \sum_{k=0}^{n-1} (M_\phi)^k 1$$

Démonstration. La matrice M_ϕ est à coefficients positifs, ce qui implique la première propriété. La seconde propriété se prouve grace à une simple récurrence. \square

Maintenant :

Lemme 3.3.4. $T[x, \phi] = \lim_{t \rightarrow \infty} \sum_{k=0}^t (M_\phi)^k 1$

Démonstration. $(M_\phi)^k$ est la matrice de transitions en k étapes sur A' . En la multipliant par le vecteur 1, on somme la probabilité qu'on soit encore en A' en k étapes composante par composante. Pour conclure, on applique le lemme 2.1.3 du télescope. \square

Corollaire 3.3.1. $T[x, \phi]$ s'obtient en partant de la fonction $0 : a \rightarrow 0$, et en prenant $\lim_{k \rightarrow \infty} (L_\phi)^k 0$.

Lemme 3.3.5. Pour tout X , on a en fait $\lim_{k \rightarrow \infty} (L_\phi)^k X = T[x, \phi]$.

Démonstration. Regardons l'expression de $(L_\phi)^k X$. Il suffit d'observer que $(M_\phi)^k X$ converge vers 0 pour tout X . En effet, $(M_\phi)^k$ est la matrice de transitions en k étapes sur A' : elle converge vers la matrice nulle, puisqu'avec probabilité 1, on quitte A' . \square

Lemme 3.3.6. La propriété (*) est vraie : $T[x, \phi_i] \leq T[x, \phi_{i+1}]$ pour tout x , avec inégalité stricte sur au moins une composante.

Démonstration. Puisque ϕ_i est Markovienne, on a $T[x, \phi_i] = L_{\phi_i} T[x, \phi_i]$.

On a construit ϕ_{i+1} de telle sorte qu'on est sur que $L_{\phi_i} T[x, \phi_i] \leq L_{\phi_{i+1}} T[x, \phi_i]$ (regarder composante par composante), avec inégalité stricte sur a .

On obtient donc $T[x, \phi_i] \leq L_{\phi_{i+1}} T[x, \phi_i]$ avec inégalité stricte sur a .

Par monotonie de $L_{\phi_{i+1}}$, on en déduit par récurrence sur k que $T[x, \phi_i] \leq (L_{\phi_{i+1}})^k T[x, \phi_i]$.

En prenant la limite quand k va vers $+\infty$, on obtient $T[x, \phi_i] \leq T[x, \phi_{i+1}]$, avec inégalité stricte sur a , c'est à dire (*). \square

Preuve du théorème 3.3.2 . Comme la propriété (*) est vraie, d'après le lemme 3.3.2 et par la discussion précédente, le théorème 3.3.2 est vrai. \square

On peut étendre le théorème 3.3.2 au cas où l'espace d'états A est infini mais où l'ensemble des états étant en relation avec plusieurs distributions est fini.

Théorème 3.3.3 (Complétude des système finiments branchants). *Si un PARS finiment branchant $\mathcal{A} = (A, \rightarrow)$ est +a.s terminant, et que l'ensemble*

$$\text{Mul}(\mathcal{A}) = \{a \in A \mid \exists \mu_1 \exists \mu_2 \mu_1 \neq \mu_2 \ a \rightarrow \mu_1 \wedge a \rightarrow \mu_2\}$$

est fini, alors il existe une fonction $V : A \rightarrow \mathbb{R}$ telle que $\inf_{i \in A} V(i) > -\infty$ et il existe $\epsilon > 0$ tel que $a \in A$, pour tout μ tel que $a \rightarrow \mu$, le drift évalué en a suivant μ défini par

$$\Delta_\mu V(a) = \sum_i \mu(i) V(i) - V(a)$$

vérifie

$$\Delta_\mu V(a) \leq -\epsilon.$$

Démonstration. Il suffit de voir que dans ce cas-ci, le nombre de stratégies Markoviennes est égal au produit pour chaque élément a de $\mathcal{Mul}(\mathcal{A})$ du nombre de distributions en relation avec a . Comme le PARS est finiment branchant, le nombre de stratégies Markoviennes est fini. On peut dans ces conditions appliquer la preuve du théorème précédent, voir la remarque 3.3.1. \square

Remarque 3.3.2. L'hypothèse finiment branchant est nécessaire est peut être vue comme une conséquence de la proposition 3.2.3. Pour voir ceci, il suffit de considérer le contre exemple utilisé dans la proposition 1.3.1 écrit sous la forme d'un PARS. Si il existait une fonction V inférieurement bornée, un $\epsilon > 0$ tels que décrit dans le théorème précédent, alors on peut calculer une nouvelle fonction V en ajoutant à l'ancienne fonction V une constante positive et en multipliant l'évaluation de l'ancienne fonction V par $\frac{1}{\epsilon}$. La nouvelle fonction V' décroît au moins de 1 en moyenne à chaque transition et est positive. Quel que soit $a \rightarrow \mu$, il existe un unique x vérifiant $\mu(x) = 1$ puisque μ est une distribution de Dirac. Ce x particulier satisfait la relation $V'(x) \leq V'(a) - 1$. Si on considère maintenant $k = V'(1, 1)$ et la stratégie faisant transiter de $(1, 1)$ vers $(0, k), (0, k - 1), \dots, (0, 0)$ alors la valuation des états par V' doit décroître d'au moins de 1 à chaque transition, ce qui est en fait impossible puisque la chaîne décrite ci dessus comporte $k + 1$ transitions et la valuation de V' vaut k au début de la chaîne. La valuation devrait donc être inférieure ou égale à -1 en fin de chaîne, chose impossible puisque V' est par hypothèse une fonction positive.

3.4 En résumé

Nous voilà maintenant en possession d'un formalisme qui va nous permettre d'exprimer les systèmes de réécriture probabiliste, et qui présente l'avantage d'être associé à des méthodes de preuves permettant de certifier si les systèmes décrits par ce formalisme terminent bien en temps moyen fini. Les propriétés de terminaison et les méthodes de preuves permettant de montrer qu'un PARS vérifie bien ces propriétés vont être appliqués de manière quasiment directe pour caractériser les systèmes de réécriture probabiliste.

4

Les systèmes de réécriture probabilistes

4.1 Introduction

Grâce au travail effectué dans le chapitre précédent, nous disposons maintenant de tous les outils nécessaires à l'introduction de notre formalisme, c'est à dire les systèmes de réécriture probabiliste. Rappelons que notre but est de concevoir un formalisme inspiré de la réécriture permettant de décrire le non déterminisme ainsi que des phénomènes probabilistes, tout en ayant des techniques de preuve de terminaison. Nous allons présenter notre formalisme en suivant la même démarche que celle utilisée dans [BN98] et dans le chapitre 1.7.1, c'est à dire que nous allons commencer par définir ce qu'est une règle de réécriture probabiliste, puis nous parlerons de ce que nous allons appeler la relation de réécriture probabiliste, et enfin les systèmes de réécriture probabiliste. Une fois cela fait, nous allons appliquer les méthodes de preuve de terminaison en temps moyen fini du chapitre précédent sur ces systèmes de réécriture, qui nous allons le voir sont des PARS.

4.2 Définition du modèle

A l'instar des systèmes de réécriture, un système de réécriture probabiliste n'est en fait qu'un ensemble fini de règles de réécriture probabiliste, où les règles de réécritures décrivent les transitions probabilistes.

Définition 4.2.1 (Règle de réécriture probabiliste). Une *règle de réécriture probabiliste* est un élément de $T(\Sigma, X) \times \text{Dist}(T(\Sigma, X))$.

Notation 7 (Représentation des règles de réécritures probabilistes). Soit $(l, \mu) \in (T(\Sigma, X), \text{Dist}(T(\Sigma, X)))$ une règle de réécriture. On représente cette règle de ré-

écriture de la façon suivante :

$$l \rightarrow \mu.$$

On représente également les règles de réécritures probabilistes de la façon suivante :

$$l \rightarrow \begin{cases} l_1 & : p_1 \\ \vdots & \\ l_n & : p_n \end{cases}.$$

On peut également représenter une règle de réécriture de la manière suivante $l \rightarrow \{r_1 : p_1 \mid \dots \mid r_n : p_n\}$.

Définition 4.2.2 (Systèmes de réécriture probabiliste). Un *système de réécriture probabiliste* est un ensemble *fini* de règles de réécriture probabiliste.

Notation 8 (PRS). On utilise la notation *PRS* pour *Probabilistic Rewrite System*, afin de désigner les systèmes de réécriture probabiliste.

Il nous faut maintenant définir l'équivalent de la relation de réécriture des systèmes de réécriture pour les systèmes de réécriture probabiliste. Pour cela remarquons que l'on peut associer à chaque système de réécriture probabiliste un système de réduction probabiliste $(T(\Sigma, X), \rightarrow_{\mathcal{R}})$ sur l'ensemble des termes $T(\Sigma, X)$ où $\rightarrow_{\mathcal{R}}$ représente ce que nous allons appeler la relation de réduction probabiliste. Rappelons qu'on note $Pos(t)$ l'ensemble des positions d'un terme $t \in T(\Sigma, X)$, $t|_{\rho}$ le sous terme de t à la position ρ avec $\rho \in Pos(t)$ et $t[s]_{\rho}$ le remplacement du sous terme de t à la position ρ par le terme s .

Définition 4.2.3 (Relation de réécriture probabiliste). Soit un système de réécriture probabiliste \mathcal{R} . On associe à \mathcal{R} le PARS $(T(\Sigma, X), \rightarrow_{\mathcal{R}})$ sur l'ensemble des termes $T(\Sigma, X)$ avec la relation $\rightarrow_{\mathcal{R}}$ définie par : $t \rightarrow_{\mathcal{R}} \mu$ si et seulement si il existe une règle de réécriture probabiliste $(g, M) \in \mathcal{R}$, une position $p \in Pos(t)$, une substitution $\sigma \in Sub$, telles que $t|_p = \sigma(g)$, et que pour tout t' ,

$$\mu(t') = \sum_{\{d \in T(\Sigma, X) \mid t' = t[\sigma(d)]_p\}} M(d).$$

Illustrons cette notion de relation réécriture probabiliste par un exemple graphique : Cette représentation d'une règle de réécriture probabiliste exprime le fait que le terme l du membre gauche est en relation avec la distribution de probabilité qui associe au terme r_1 la probabilité p_1 , au terme r_2 la probabilité p_2 et au terme r_3 la probabilité $1 - p_1 - p_2$. Maintenant, si dans un terme T_i il existe un sous terme qui est une instance du terme l , alors le terme T_i est en relation avec la distribution qui charge les autres termes T_{r_1} avec probabilité p_1 , T_{r_2} avec probabilité p_2 et T_{r_3} avec probabilité $1 - p_1 - p_2$. On précise que la position des sous termes correspondant aux

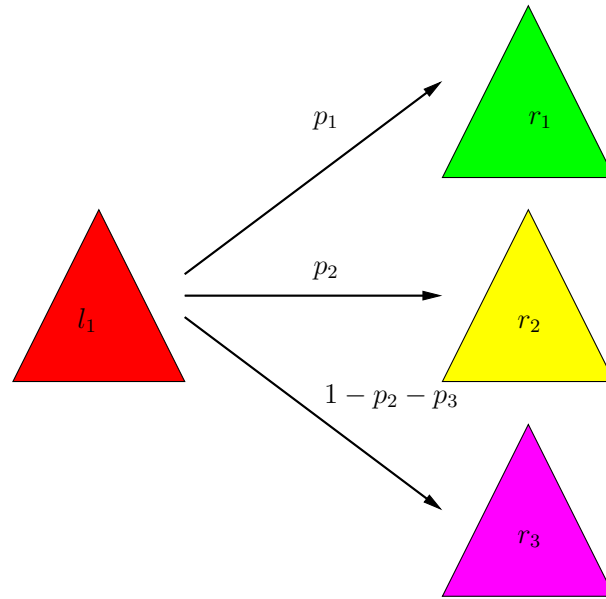


FIG. 4.1 – Une règle de réécriture probabiliste

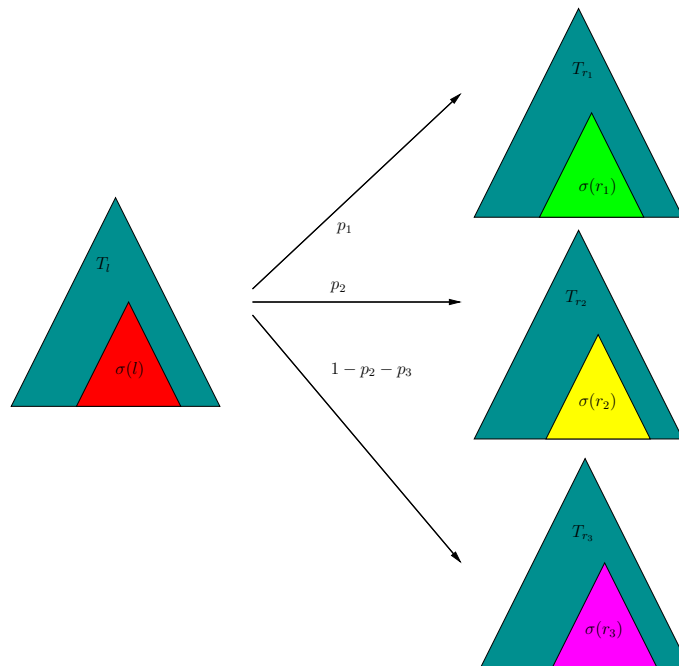


FIG. 4.2 – La relation de réécriture appliquée à un terme

instances de r_1, r_2 et r_3 sont les mêmes que celle de l dans T_l . Passons maintenant à l'étude d'un cas concret : on étudie le système de réécriture probabiliste formé par la

règle de réécriture probabiliste suivante

$$f(x, y) \mapsto \{g(a) : 1/2 \mid y : 1/2\}$$

Alors, en calculant la relation de réécriture, on obtient que le terme $f(b, c)$ se réécrit en $g(a)$ avec probabilité $1/2$, et en c avec probabilité $1/2$. De même on observe que $f(b, g(a))$ se réécrit en $g(a)$ avec probabilité 1 .

Le non déterminisme provient de la multiplicité des sous termes d'un terme qui peuvent être une instance d'un membre gauche d'une règle de réécriture probabiliste.

Exemple 4.2.1. Considerons le système de réécriture probabiliste $T(\Sigma, X), \rightarrow_{\mathcal{R}}$ avec :

$$\begin{aligned} \Sigma &= \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \\ \Sigma^0 &= \{a, b, c\} \\ \Sigma^1 &= \{F\} \\ \Sigma^2 &= \{G\} \\ \Sigma^3 &= \{H\} \\ \rightarrow_{\mathcal{R}} &= \{r_1\} \\ r_1 &: F(X) \rightarrow \begin{cases} G(X, a) & : \frac{1}{3} \\ b & : \frac{2}{3} \end{cases} \end{aligned}$$

On rappelle que Σ^i représente l'ensemble des symboles de fonction d'arité i et notons par $l(r_1)$ le membre gauche de la règle r_1 . Représentons l'arbre des sous termes du terme

$$t = G(G(F(G(c, a)), G(a, H(a, b, F(b))))), a).$$

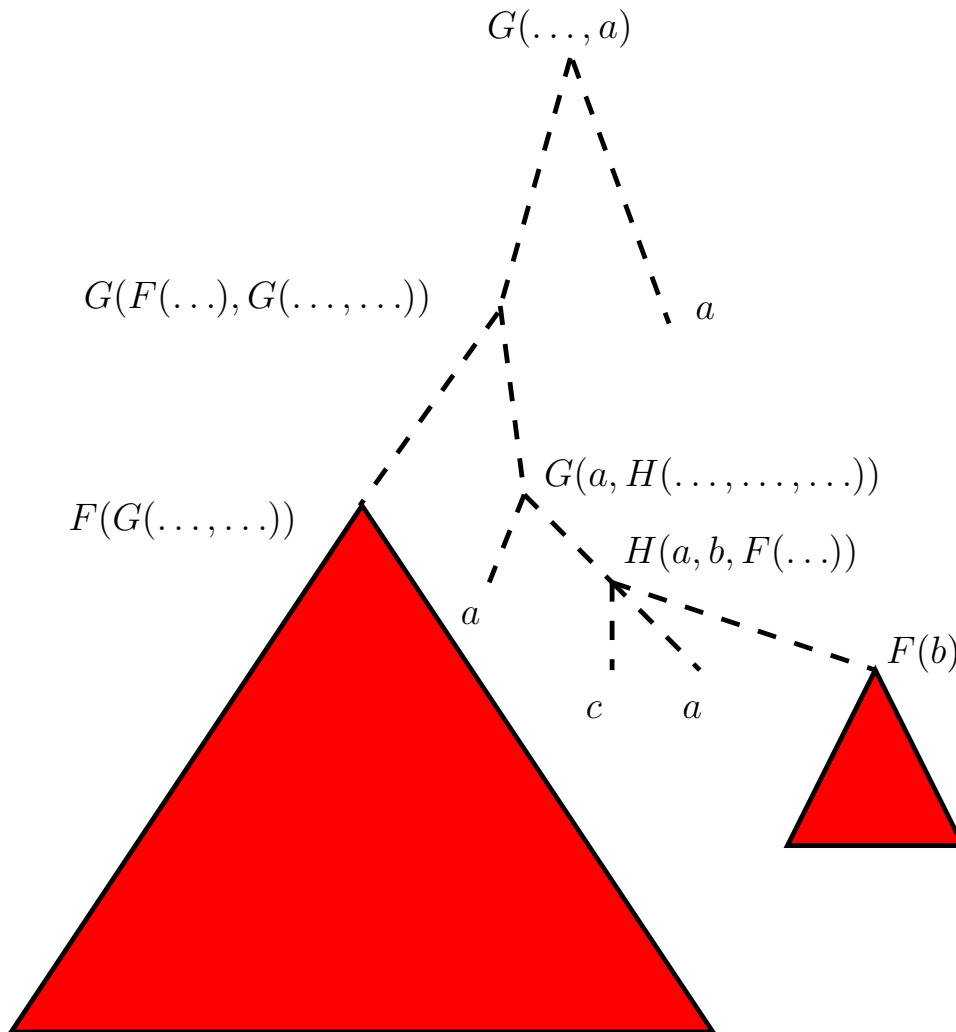
Dans la figure 4.3 on représente par un triangle rouge chaque instance du terme $F(X)$ dans l'arbre des sous termes de t . On trouve deux positions $p_1 = 11$ et $p_2 = 1223$ et deux substitutions vérifiant $\sigma_1(X) = G(c, a)$ et $\sigma_2(X) = b$ telles que $\sigma_1(t|_{p_1}) = \sigma_1(l(r_1)) = F(G(c, d))$ et $\sigma_2(t|_{p_2}) = \sigma_2(l(r_1)) = F(b)$

Comme la règle peut s'appliquer en p_2 , le terme t est en relation par $\rightarrow_{\mathcal{R}}$ avec les deux termes représentés à droite de la figure 4.4.

De même, lorsque plusieurs règles peuvent s'appliquer sur un terme, il faut choisir une de ces règles dès lors que l'on veut calculer une dérivation. Notre modèle ne spécifie pas la manière de choisir la règle à appliquer justement pour permettre d'exprimer le non déterminisme.

Proposition 4.2.1. *Les processus de décision Markoviens à espace d'états dénombrables expriment les systèmes de réécriture probabilistes.*

Démonstration. Soit \mathcal{R} un système de réécriture probabiliste et soit $(T(\Sigma, X), \rightarrow_{\mathcal{R}})$ le système abstrait de réduction probabiliste associé. Nous allons montrer qu'il existe

FIG. 4.3 – La règle r_1 peut s'appliquer en deux position.

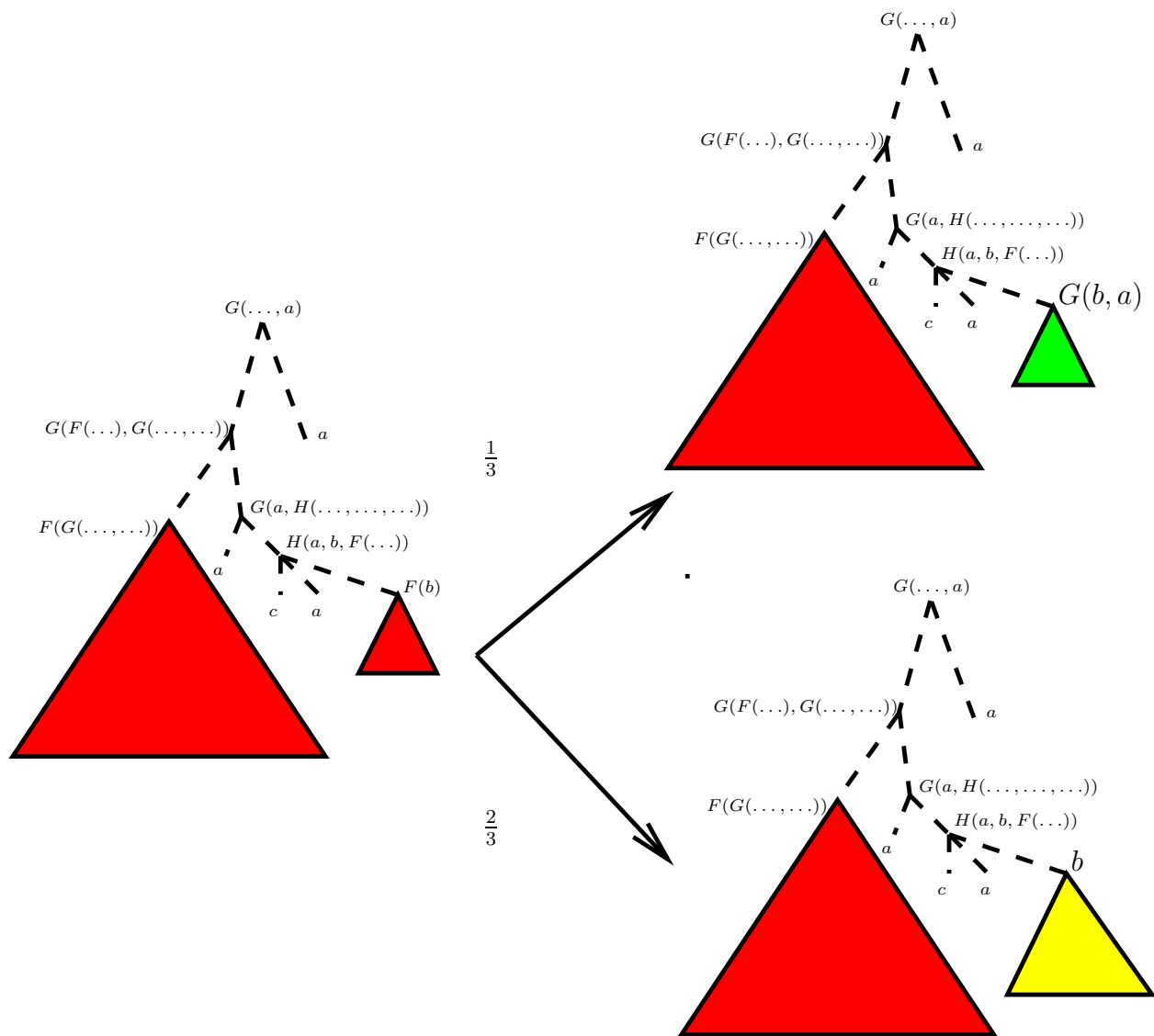


FIG. 4.4 – La règle r_1 appliquée à la position p_2 .

un processus de décision Markovien (S, A, p) représentant le PARS $(T(\Sigma, X), \rightarrow_{\mathcal{R}})$. L'ensemble des états S de ce processus de décision Markovien correspond à l'ensemble des termes $T(\Sigma, X)$ union $\{\perp\}$.

Il reste maintenant à déterminer A l'ensemble des actions, avec $A = \cup_{s \in S} A(s)$. On associe à l'état \perp une action, qu'on appelle T . $A(\perp) = T$. Pour tout état $s \in S$, $A(s) = \{\mu | s \rightarrow_{\mathcal{R}} \mu\}$. L'ensemble des actions disponible pour un état s , rappelons le est un terme, correspond en fait à l'ensemble des distributions en relation avec le terme s par la relation de réduction probabiliste $\rightarrow_{\mathcal{R}}$. Il reste maintenant à définir la fonction p . Soient $s, t \in T(\Sigma, X)$ et $s \rightarrow_{\mathcal{R}} \mu$. On a alors $\mu \in A(s)$ et :

$$\begin{aligned} p_{\perp, \perp}(T) &= 1 \\ p_{s, t}(\mu) &= \mu(t) \end{aligned}$$

On a bien représenté le PARS $(T(\Sigma, X), \rightarrow_{\mathcal{R}}$ associé à un système de réécriture quelconque \mathcal{R} par le processus de décision Markovien (S, A, p) . \square

Remarque 4.2.1. Cet exemple illustre bien le fait qu'avec peu de règles de réécriture on peut définir une relation de réécriture complexe sur un espace de termes infini.

Proposition 4.2.2. *Tous les processus de décision Markoviens à espace d'états fini et d'actions finis sont simulés par les PRS.*

Démonstration. Il suffit pour voir cela de numéroter chaque état et d'associer à chaque couple « état, action » une règle de réécriture probabiliste. On a bien ainsi défini un nombre fini de règles de réécriture probabilistes. \square

Cependant, comme par définition le nombre de règles de réécriture d'un système de réécriture probabiliste est fini, on a certaines limitations :

Proposition 4.2.3. *Il existe des instances de chaînes de Markov à espace d'états infinis mais dénombrable qui ne sont pas exprimable par des systèmes de réécriture probabilistes.*

Démonstration. Considérons la chaîne de Markov définie sur l'espace des entiers naturels telle que :

$$\begin{aligned} P(n, n+1) &= \frac{1}{n+1} \\ P(n, n-1) &= \frac{n}{n+1} \quad \text{si } n \geq 1 \\ P(0, 0) &= 1 \end{aligned}$$

Comme à chaque état est associé une distribution avec des probabilités différentes, on ne peut pas exprimer cette chaîne avec un nombre fini de règles de réécriture probabiliste. Comme les chaînes de Markov sont des cas particuliers de processus de décision Markoviens, cela montre le résultat. Ce résultat s'étend de façon trivial au cas des processus de décision Markoviens. \square

4.3 Terminaison des systèmes de réécriture probabilistes

Nous apportons dans cette section un résultat comparable à celui disant qu'un système de réécriture termine si et seulement si il existe un ordre de réduction monotone sur chaque règle de réécriture. La propriété nous intéressant ici, est bien entendu la terminaison presque sûre positive, pour les raisons évoquées dans la remarque 3.2.1.

Tout d'abord définissons la notion de terminaison presque sûre pour un système de réécriture probabiliste,

Définition 4.3.1 (Terminaison presque sûre). Un système de réécriture probabiliste \mathcal{R} est dit presque sûrement terminant si le PARS $(T(\Sigma, X), \rightarrow_{\mathcal{R}})$ est presque sûrement terminant.

Faisons de même pour la notion de terminaison presque sûre positive :

Définition 4.3.2 (Terminaison presque sûre positive). Un système de réécriture probabiliste \mathcal{R} termine presque sûrement et positivement si le PARS $(T(\Sigma, X), \rightarrow_{\mathcal{R}})$ est positivement presque sûrement terminant.

Nous pouvons maintenant énoncer nos critères de terminaison pour les systèmes de réécriture :

Théorème 4.3.1. *Un système de réécriture probabiliste \mathcal{R} est positivement et presque sûrement terminant s'il existe une fonction $V : T(\Sigma, X) \rightarrow \mathbb{R}$ inférieurement bornée, un $\epsilon > 0$, tels que*

1. "Le drift de chaque règle est plus petit que $-\epsilon$ ", où pour chaque règle $g \rightarrow M \in \mathcal{R}$, le drift

$$\Delta_M V(g) = \sum_d M(d)(V(d) - V(g))$$

satisfait

$$\Delta_M V(g) \leq -\epsilon.$$

2. "La majoration du drift par $-\epsilon$ est préservée par substitution", c'est à dire que pour chaque terme $s \in T(\Sigma, X)$, pour tout μ avec $s \rightarrow \mu$, pour toute substitution $\sigma \in \text{Sub}$, si

$$\Delta_{\mu} V(s) \leq -\epsilon$$

alors le drift

$$\Delta_{\sigma(\mu)} V(\sigma(s)) = \sum_{s'} \mu(s')(V(\sigma(s')) - V(\sigma(s)))$$

vérifie

$$\Delta_{\sigma(\mu)} V(\sigma(s)) \leq -\epsilon$$

3. “la majoration du drift par $-\epsilon$ est préservée par contexte”, c’est à dire que pour chaque terme $s_1, \dots, s_n, s \in T(\Sigma, X)$, pour toute distribution μ telle que $s \rightarrow \mu$, pour chaque symbole de fonction f , si

$$\Delta_\mu V(s) \leq -\epsilon,$$

alors le drift

$$\Delta_{f(s_1, \dots, \mu, \dots, s_n)} V(f(s_1, \dots, s, \dots, s_n)) = \sum_{s'} \mu(s') (V(f(s_1, \dots, s', \dots, s_n)) - V(f(s_1, \dots, s, \dots, s_n)))$$

satisfait

$$\Delta_{f(s_1, \dots, \mu, \dots, s_n)} V(f(s_1, \dots, s, \dots, s_n)) \leq -\epsilon.$$

Remarque 4.3.1. Les conditions 1, 2 et 3 généralisent au cas probabiliste la définition d’un ordre de réécriture.

Démonstration. Supposons 1, 2 et 3 vérifiées. On a alors $t \rightarrow \mu$ si et seulement si il existe une règle $(g, M) \in \mathcal{R}$, une position $p \in Pos(t)$, une substitution $\sigma \in Sub$, telle que $t|_p = \sigma(g)$, et pour tout t' , $\mu(t') = \sum_{t'=t[\sigma(d)]_p} M(d)$.

Comme toute dérivation $t \rightarrow \mu$ correspond à l’application d’une règle (g, M) , il suffit alors, d’après le théorème 3.3.1, de montrer que pour chaque règle (g, M) et tout terme t , où $t \rightarrow \mu$ par l’application de (g, M) , on a bien

$$\Delta_\mu V(t) \leq -\epsilon.$$

Et on montre cela par induction sur la longueur de p . Si p a pour longueur 0, alors $t = \sigma(g)$. Grâce à la condition 1, on a $\Delta_M V(g) \leq -\epsilon$.

Grâce à la condition 2, le fait que $g \rightarrow M$ vérifie

$$\Delta_M V(g) \leq -\epsilon,$$

implique que

$$\Delta_\mu V(t) = \Delta_{\sigma(M)} V(\sigma(g)) \leq -\epsilon,$$

ce qui au bout du compte permet d’établir l’égalité.

Si $p = p_1 p_2 \dots p_k$ est de longueur $k > 0$, alors t peut s’écrire $f(s_1, \dots, s, \dots, s_n)$ et $s \rightarrow \mu'$ par l’application de (g, M) .

En appliquant l’hypothèse d’induction $\Delta_{\mu'} V(s) \leq -\epsilon$. La condition 3 permet d’écrire que

$$\Delta_\mu V(t) = \Delta_{f(s_1, \dots, \mu', \dots, s_n)} V(f(s_1, \dots, s, \dots, s_n)) \leq -\epsilon,$$

où l’égalité s’établit de la même manière que dans le paragraphe précédent. \square

Pour montrer qu'un PRS termine presque sûrement et positivement, il suffira maintenant de montrer les conditions 1, 2 et 3. Mais, cela n'est pas une chose aisée. Nous avons cependant trouvé des conditions plus simples à vérifier, qui s'avèrent être suffisantes.

Propriété 4.3.1. On a la réciproque du théorème précédent si l'ensemble des termes est fini.

Démonstration. Si \mathcal{R} est positivement et presque sûrement terminant, et $T(\Sigma, X)$ est fini alors d'après le théorème 3.3.2, il existe une fonction $V : T(\Sigma, X) \rightarrow \mathbb{R}$, telle que $\inf_{i \in A} V(i) > -\infty$, et un $\epsilon > 0$, tels que pour tout état $a \in T(\Sigma, X)$, pour tout μ avec $a \rightarrow \mu$, $\Delta_\mu V(a) \leq -\epsilon$.

En particulier, pour une règle $g \rightarrow M$ pour $a = g$, pour $\mu = M$ on a,

$$\begin{aligned} \Delta_\mu V(a) &= \sum_{t'} \mu(t') V(t') - V(a) \\ &= \sum_d M(d) (V(d)) - V(a) \\ &= \Delta_M V(g) \\ &\leq -\epsilon. \end{aligned}$$

Dans le cas où on a $s \rightarrow \mu'$, avec $a = \sigma(s)$, on a $a \rightarrow \mu$, où $\mu(\sigma(s')) = \mu'(s')$, ce qui implique

$$\begin{aligned} \Delta_\mu V(a) &= \sum_{t'} \mu(t') V(t') - V(a) \\ &= \sum_{t'=\sigma(s')} \mu'(s') V(\sigma(s')) - V(a) \\ &= \Delta_{\sigma(\mu')} V(\sigma(s)) \\ &\leq -\epsilon. \end{aligned}$$

Et enfin, lorsque $s \rightarrow \mu'$, pour $a = f(s_1, \dots, s, \dots, s_n)$, on a $a \rightarrow \mu$ où $\mu(f(s_1, \dots, s', \dots, s_n)) = \mu'(s')$, donc

$$\begin{aligned} \Delta_\mu V(a) &= \sum_{t'} \mu(t') V(t') - V(a) \\ &= \sum_{s'} \mu'(s') V(f(s_1, \dots, s', \dots, s_n)) - V(f(s_1, \dots, s, \dots, s_n)) \\ &= \Delta_{f(s_1, \dots, \mu', \dots, s_n)} V(f(s_1, \dots, s, \dots, s_n)) \\ &\leq -\epsilon. \end{aligned}$$

□

4.4 Critères de terminaison presque sûre positive

Ici, nous allons énoncer des conditions suffisantes impliquant les conditions 1, 2 et 3 du théorème 4.3.1.

Définition 4.4.1 (Préservation contextuelle d'une fonction). Une fonction $V : T(\Sigma, X) \rightarrow \mathbb{R}$ est *préservée par contexte* si pour tout t, t', s_1, \dots, s_n et tout symbole de fonction f ,

$$V(f(s_1, \dots, t, \dots, s_n)) - V(f(s_1, \dots, t', \dots, s_n)) = V(t) - V(t').$$

Définition 4.4.2 (Décroissance par substitution pour une règle). Une fonction $V : T(\Sigma, X) \rightarrow \mathbb{R}$ est qualifiée de *décroissante par substitution* pour une règle de réécriture probabiliste (g, M) , si pour toute substitution $\sigma \in \text{Sub}$, si on écrit

$$\Delta_{\sigma(M)}V(\sigma(g)) = \sum_d M(d)(V(\sigma(d)) - V(\sigma(g)))$$

et

$$\Delta_M V(g) = \sum_d M(d)(V(d) - V(g))$$

comme auparavant, on a

$$\Delta_{\sigma(M)}V(\sigma(g)) \leq \Delta_M V(g).$$

Théorème 4.4.1. *Un système de réécriture probabiliste \mathcal{R} est positivement presque sûrement terminant si il existe une fonction $V : T(\Sigma, X) \rightarrow \mathbb{R}$ telle que $\inf_{i \in A} V(i) > -\infty$, un réel $\epsilon > 0$, tels que le drift de chaque règle est plus petit que $-\epsilon$, V préserve le contexte et V décroît par substitution pour chaque règle de \mathcal{R} .*

Démonstration. La condition 1 est donnée par hypothèse.

- Comme V préserve les contexte, pour tout f, s, s_1, \dots, s_n et μ , on a :

$$\Delta_{f(s_1, \dots, \mu, \dots, s_n)}V(f(s_1, \dots, s, \dots, s_n)) = \Delta_\mu V(s)$$

ce qui implique la condition 3.

- Etant donné que 1 et 3 sont vérifiées, il ne reste plus qu'à montrer que

$$\Delta_{\sigma(M)}V(\sigma(g)) = \sum_d M(d)(V(\sigma(d)) - V(\sigma(g))) \leq -\epsilon$$

pour chaque règle de réécriture (g, M) et pour toute substitution σ afin d'avoir la preuve du théorème 4.3.1 dans le sens indirect. Comme V est décroissant par substitution, on a bien :

$$\Delta_{\sigma(M)}V(\sigma(g)) \leq \Delta_M V(g) \leq -\epsilon$$

par la condition 1, ce qui achève la preuve. □

4.5 Application à quelques exemples simples

Nous présentons ici, un ensemble de systèmes de réécriture probabiliste, et nous prouvons qu'ils terminent en temps moyen fini en appliquant les méthodes de preuve introduites dans ce chapitre.

Exemple 4.5.1. Le système de réécriture probabiliste composé de la seule règle

$$a \rightarrow \{a : 1/2 | b : 1/2$$

est $+a.s$ terminant.

Démonstration. Pour cela il suffit de considérer la fonction V qui évalue les termes de la façons suivante : $V(a) = 10$, $V(b) = 2$, et de calculer le drift de la règle :

$$\Delta V(a) = 1/2 \times 10 + 1/2 \times 2 - 10 < 0.$$

□

Exemple 4.5.2. Le système de réécriture probabiliste

$$\begin{aligned} f(x) &\rightarrow \{x : p_1 | f(f(x)) : 1 - p_1 \\ f(x) &\rightarrow \{x : p_2 | f(f(x)) : 1 - p_2 \end{aligned}$$

est $+a.s$ terminant si $p_1 > 1/2$ et $p_2 > 1/2$.

Démonstration. Pour montrer cela, considérons la fonction V qui calcule la taille des termes. V préserve les contextes et décroît par substitution pour les deux règles. On calcule le drift des deux règles

$$-1 \times p_i + 1 \times (1 - p_i) = 1 - 2p_i \leq \min(1 - 2p_1, 1 - 2p_2) < 0$$

et on constate qu'il est majoré par une constante négative.

□

Exemple 4.5.3. Le système de réécriture probabiliste

$$\begin{aligned} f(x) &\rightarrow \{f(f(x)) : p_{11} | g(f(x)) : p_{12} | x : p_{13} \\ f(h(f(x), x)) &\rightarrow \{h(g(f(f(x))), f(x)) : p_{21} | g(f(x)) : p_{22} | f(g(f(f(x)))) : p_{23} \end{aligned}$$

est $+a.s$ terminant si $p_{11} + p_{13} < p_{13}$ et $2 * p_{21} + p_{23} < p_{22}$.

Démonstration. Pour montrer cela considérons la même fonction V . On calcule le drift de la première règle (g_1, M_1) qui vaut :

$$\Delta_{M_1} V(g_1) = p_{11} + p_{12} - p_{13}$$

De même pour (g_2, M_2) , on calcule

$$\Delta_{M_2} V(g_2) = 2p_{21} - 2p_{22}$$

On constate que les deux drifts sont négatifs. V est décroissante par substitution sur les deux règles : soit une substitution $\sigma \in Sub$, en notant $n = V(\sigma(x))$, on calcule que l'on a

$$\Delta_{\sigma(M_1)} V(\sigma(g_1)) = p_{11} + p_{12} - p_{13} = \Delta_{M_1} V(g_1)$$

et

$$\begin{aligned}\Delta_{\sigma(M_2)}V(\sigma(g_2)) &= (p_{2_1} - 1)n + 2p_{2_1} - p_{2,2} + p_{2,3} \\ &\leq 2p_{2_1} - 2p_{2,2} \\ &= \Delta_{M_2}V(g_2)\end{aligned}$$

comme $n \geq 1$ par ce que x est substitué par au moins un symbole, et

$$(p_{2_1} - 1) = -p_{2_2} - p_{2_3} < 0$$

ce qui achève la preuve. \square

4.6 Preuve de terminaison grâce à une méthode d'interprétation polynomiale.

Le fait que le formalisme de la réécriture probabiliste soit une généralisation des systèmes de réécriture va permettre d'adapter des méthodes de preuve de terminaison utilisées sur les systèmes de réécriture. Nous allons ici prouver qu'un système de réécriture probabiliste termine en temps moyen fini, en utilisant une méthode de preuve inspirée des interprétations polynomiales (voir l'exemple 1.6.1, [BN98]). Nous avons adapté cette méthode pour appliquer le théorème 4.4.1 et prouver la terminaison presque sûre positive du système.

Exemple 4.6.1.

$$\begin{aligned}X \odot (Y \oplus Z) &\rightarrow \begin{cases} (X \odot Y) \oplus (X \odot Z) & : p_1 \\ X \odot (Y \oplus Z) & : 1 - p_1 \end{cases} \\ ((X \odot Y) \oplus (X \odot Z)) \oplus X &\rightarrow \begin{cases} (X \odot (Y \oplus Z)) \oplus X & : p_2 \\ X \odot ((Y \oplus Z) \oplus X) & : 1 - p_2 \end{cases}\end{aligned}$$

Le système de réécriture probabiliste ci dessus, termine en temps moyen fini.

Démonstration. Considérons l'interprétation des symboles de fonction $\{\oplus, \odot\}$ donné par les deux polynomes

$$\begin{aligned}P_{\oplus} &:= 2X + Y + 1 \\ P_{\odot} &:= XY\end{aligned}$$

et à l'instar de la preuve de la terminaison de l'exemple 1.6.1 on associe P_{\oplus} à \oplus et P_{\odot} à \odot . Sur $\mathcal{A} = \mathbb{N}$, l'interprétation polynomiale satisfait

$$\begin{aligned}\oplus^{\mathcal{A}}(m, n) &:= 2m + n + 1 \\ \odot^{\mathcal{A}}(m, n) &:= m \times n\end{aligned}$$

On calcule l'interprétation polynomiale d'un terme comme dans le cas classique, c'est-à-dire que pour un terme t contenant n variables, on va construire un polynome P_t à n variables X_1, \dots, X_n . Dans la suite de la preuve de terminaison de cet exemple,

on utilisera la notation $[t]$ pour représenter le polynôme P_t et rendre la lecture des preuves plus aisée.

$[X \oplus Y] = 2[X] + [Y] + 1$ et $[X \odot Y] = [X] * [Y]$, où $[P] \in \mathbb{N}[X_1, \dots, X_n]$ représente l'interprétation polynomiale du terme $P \in \Sigma^n$, c'est-à-dire les symboles de fonction d'arité n .

Fixons la valeur de $n_0 \geq 2$ à une valeur à déterminer. L'ensemble des entiers supérieurs à n_0 sont préservés par les polynômes $[P]$, $P \in \Sigma$. Soit V la fonction qui évalue les termes $P \in \Sigma$ de la manière suivante $[P](n_0, \dots, n_0)$. On notera par la suite $\{P\}$ pour $V(P)$.

On a

$$\begin{aligned} [X \odot (Y \oplus Z)] &= 2[X][Y] + [X][Z] + [X] \quad \text{et,} \\ [(X \odot Y) \oplus (X \odot Z)] &= 2[X][Y] + [X][Z] + 1 \end{aligned}$$

et le drift de la première règle est donné par :

$$\begin{aligned} \Delta_{R_1} V(X \odot (Y \oplus Z)) &= p_1 \times \{(X \odot Y) \oplus (X \odot Z)\} \\ &\quad + (1 - p_1) \{X \odot (Y \oplus Z)\} - \{X \odot (Y \oplus Z)\} \\ &= p_1 \times (1 - \{X\}) \end{aligned}$$

cette évaluation est bien négative, et de plus, la décroissance moyenne est préservée par toute substitution des variables. En d'autres termes cette évaluation est décroissante par substitution. Si on regarde la seconde règle, on a

$$\begin{aligned} [((X \odot Y) \oplus (X \odot Z)) \oplus X] &= 4[X][Y] + 2[X][Z] + [X] + 3 \\ [X \odot ((Y \oplus Z) \oplus X)] &= 2[X][Y] + 2[X][Z] + [X] + 2 \end{aligned}$$

donc,

$$\begin{aligned} \Delta_{R_2} V(((X \odot Y) \oplus (X \odot Z)) \oplus X) &= p_2 \times \{(X \odot (Y \oplus Z)) \oplus X\} \\ &\quad + (1 - p_2) \times \{X \odot ((Y \oplus Z) \oplus X)\} \\ &\quad - \{((X \odot Y) \oplus (X \odot Z)) \oplus X\} \\ \Delta_{R_2} V(((X \odot Y) \oplus (X \odot Z)) \oplus X) &= 2(p_2 - 1)\{X\}\{Y\} + 2p_2\{X\} - p_2 - 1 \end{aligned}$$

Le drift de cette règle n'est pas nécessairement négatif, en particulier si on prend $p_2 = 1$, ce drift est positif. Cependant, si on considère que $p_2 < 1$, en prenant $n_0 \geq p_2/(1-p_2)$, le drift de cette règle devient négatif et pour un tel n_0 , l'interprétation polynomiale conserve la décroissance par substitution.

Maintenant, en prêtant attention aux symboles d'interprétation $\{\oplus, \odot\}$, on constate qu'ils sont linéaires en fonction de chacun de leur arguments, linéaires avec des coefficients entiers positifs. Cette propriété va nous permettre de montrer que cette interprétation vérifie bien toutes les conditions nécessaire pour appliquer le théorème 4.4.1.

Montrons que la majoration du drift pas $-\epsilon$ est contextuellement préservée. Pour faire cela, il va falloir vérifier que pour tout symbole de fonction de $f \in \Sigma$ et pour tout $s \rightarrow \mu$ tel que $\Delta_{s \rightarrow \mu}\{s\} \leq -\epsilon$ on a bien :

$$\Delta_{f(s_1, \dots, \mu, \dots, s_n)}\{f(s_1, \dots, s, \dots, s_n)\} \leq -\epsilon$$

Dans ce cas d'étude, les fonctions considérées sont $f \in \{\oplus, \odot\}$. Nous allons montrer que ces deux fonctions satisfont bien les conditions énoncées ci dessus. Supposons que $s \rightarrow \mu$ vérifie $\Delta_{s \rightarrow \mu}\{s\} \leq -\epsilon$. De même considérons s_1 un terme de $T(\Sigma, X)$.

$$\begin{aligned} \Delta_{\mu \oplus s_1}\{s \oplus s_1\} &= \sum_{s'} \mu(s')(2 \times \{s'\} + \{s_1\} + 1) - 2 \times \{s\} - \{s_1\} - 1 \\ &= 2 \times (\sum_{s'} \mu(s')\{s'\} - \{s\}) \\ &= 2 \times \Delta_{\mu}\{s\} \leq -2 \times \epsilon \end{aligned}$$

On calcule que $\Delta_{s_1, \mu}\{s_1, s\} \leq -\epsilon$ ce qui implique que la majoration du drift par $-\epsilon$ est préservée par contexte pour le symbole de fonction \oplus .

Vérifions que cela est également le cas pour \odot :

$$\begin{aligned} \Delta_{\{\mu \odot s_1\}}\{s \odot s_1\} &= \sum_{s'} \mu(s')(\{s'\}\{s_1\}) - \{s\}\{s_1\} \\ &= \{s_1\} \times (\sum_{s'} \mu(s')\{s'\} - \{s\}) \\ &= \{s_1\} \times \Delta_{\mu}\{s\} \\ &\leq -\epsilon \end{aligned}$$

car pour tout terme s_1 de $T(\Sigma, X)$ on a $\{s_1\} > 1$. En remarquant que $\{X \odot Y\} = \{Y \odot X\}$ on peut conclure en disant que la majoration du drift par $-\epsilon$ est préservée par contexte pour le symbole de fonction \odot . On en conclut que pour $p_2 < 1$, on peut construire une fonction V qui à un polynôme $P \in \mathbb{N}[X_1, \dots, X_n]$, associe $V(P) := \{P\}$ avec V vérifiant les hypothèses du théorème 4.4.1. \square

4.7 Résumé du chapitre

Nous avons dans ce chapitre défini un formalisme permettant de décrire avec peu de règles des PARS complexes. Ce formalisme exprime les systèmes de réécriture classiques et naturellement modélise des systèmes de transitions où interviennent des transitions probabilistes. Nous avons expliqué la sémantique d'un tel formalisme, puis nous avons adapté les théorèmes du chapitre précédent pour pouvoir fournir un ensemble de conditions entraînant la terminaison en temps moyen fini de tels systèmes de réécriture probabiliste. Néanmoins, nous devons insister sur le fait que la notion de terminaison presque sûre positive d'un système de réécriture probabiliste est une propriété très forte est foncièrement difficile à montrer. Dans le cas général, nous n'avons pas nécessairement besoin que les systèmes de réécritures terminent en temps moyen fini quel que soit l'ordre d'application des règles. En effet, s'intéresser

à une notion de terminaison plus faible, comparable à la terminaison sous stratégie dans le cadre de la réécriture classique va permettre de modéliser des systèmes plus intéressants et complexes. Nous allons nous employer à cela dans le prochain chapitre.

5

La terminaison presque sûre positive sous stratégies

5.1 Introduction

Dans le chapitre précédent, on a étudié la notion de terminaison presque sûre positive, qui garantit que la longueur moyenne des dérivations est bornée, indépendamment de la manière qu'on a de choisir les règles de réécriture qu'on applique.

Cependant, nous allons montrer qu'il est possible d'exprimer l'équivalent de stratégies de réécriture pour les systèmes de réécriture probabilistes, ce qui est intéressant dès lors que l'on n'est pas intéressé par la terminaison de toute les dérivations possibles d'un système de réécriture probabilistes, mais d'un sous ensemble. On établit un parallèle avec le fait que la terminaison sous stratégies dans le cadre de la réécriture classique apporte un raffinement dans l'étude des problèmes de terminaison [FGK03, FGK04, BKKR01]. De même, la preuve de la terminaison sous stratégie permet de travailler sur des systèmes de réécriture plus complexes, mais où le sous ensemble des dérivations « intéressantes » peut s'exprimer grâce à un langage de stratégies.

Nous allons dans ce chapitre définir la notion de terminaison presque sûre positive sous stratégie, puis nous présenterons certaines propriétés concernant la fréquence d'application de classe de règles pour enfin pouvoir présenter des critères suffisants de terminaison en temps moyen fini sous stratégies.

5.2 Stratégies et terminaison sous stratégie

Tout d'abord rappelons que nous avons déjà défini la notion de stratégie pour les PARS, que nous avons appelé politique dans [BG05]. Une stratégie pour un PARS est une fonction qui associe à une histoire non terminale la prochaine règle de réécriture probabiliste à appliquer. En d'autres termes, cette notion de stratégie est un moyen

de résoudre les choix non-déterministes, c'est-à-dire que la stratégie choisit une et une seule solution parmi toutes celles possibles.

En ce qui concerne les systèmes de réécriture probabiliste le non déterminisme a plusieurs sources : en effet, il est possible que plusieurs règles puissent s'appliquer, et pour chaque règle que cette règle puisse s'appliquer à différentes positions et pour une même position plusieurs substitutions peuvent permettre d'appliquer la règle. En d'autres termes, dans le cas d'un système de réécriture probabiliste, une stratégie détermine quelle règle à appliquer et à quelle position, cela en fonction de l'ensemble des termes parcourus dans l'histoire courante.

Définition 5.2.1 (Terminaison presque sûre positive sous stratégie). Étant donnée une classe de stratégies Φ , un PARS $\mathcal{A} = (A, \rightarrow)$ va être qualifié de *positivement et presque sûrement terminant sous Φ* , si pour tout état $a \in A$, le nombre moyen de réductions, noté $T[a, \phi]$ menant à un état terminal depuis l'état a suivant toute stratégie ϕ de Φ est fini.

Exemple 5.2.1. Soit le système de réécriture probabiliste composé de deux règles,

$$\begin{aligned} r_1 &:= a \rightarrow \{a : 1 \\ r_2 &:= a \rightarrow \{b : 1 \end{aligned}$$

Ce système de réécriture code le processus de décision Markovien présenté dans la figure 5.1, où l'état \perp représente l'état du système une fois qu'un état terminal a été atteint. Ici, l'état b est terminal, puisque sa seule transition de sortie est l'état \perp . Le fait d'avoir un état \perp sur lequel on boucle pour représenter le fait que le système a terminé, permet d'étudier les systèmes de réécriture probabiliste d'une façon similaire à celle qu'on a d'étudier les processus de décision Markoviens.

Ce système n'est clairement pas presque sûrement terminant, puisqu'il existe une dérivation infinie $a \rightarrow a \rightarrow \dots a \rightarrow \dots$.

Cependant, ce système termine presque sûrement et positivement sous toutes stratégies Markoviennes aléatoire ϕ telle que pour toute histoire réalisable $h = a_0 \dots a_n$ on ait :

$$\begin{aligned} P(\phi(h) = r_1) &= p_1 \\ P(\phi(h) = r_2) &= 1 - p_1 \end{aligned}$$

avec $p_1 < 1$. Pour montrer cela, il suffit de voir que le système de réécriture probabiliste $\{r_1, r_2\}$ se comporte comme le PRS ci-dessous quand le choix des règles est conditionné par une telle stratégie ϕ ,

$$a \rightarrow \begin{cases} a & : p_1 \\ b & : 1 - p_1 \end{cases}$$

La terminaison presque sûre positive de ce dernier peut être établie grâce aux théorèmes énoncés lors des chapitres précédents avec la fonction $V(a) = 1, V(b) = 0$.

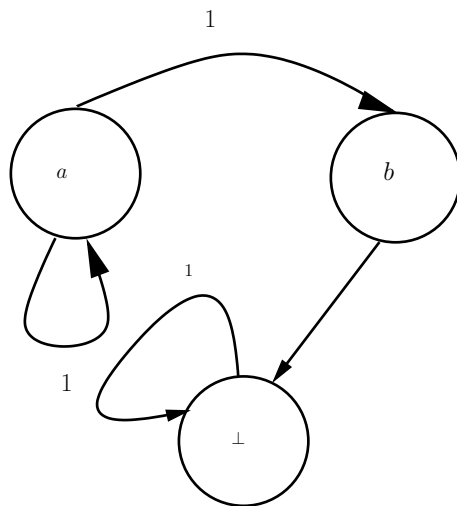


FIG. 5.1 – Simulation du système de réécriture probabiliste par un processus de décision Markovien

En d'autres termes, sous cette stratégie le PRS se comporte alors de la même manière que la chaîne de Markov représentée dans la figure 5.2

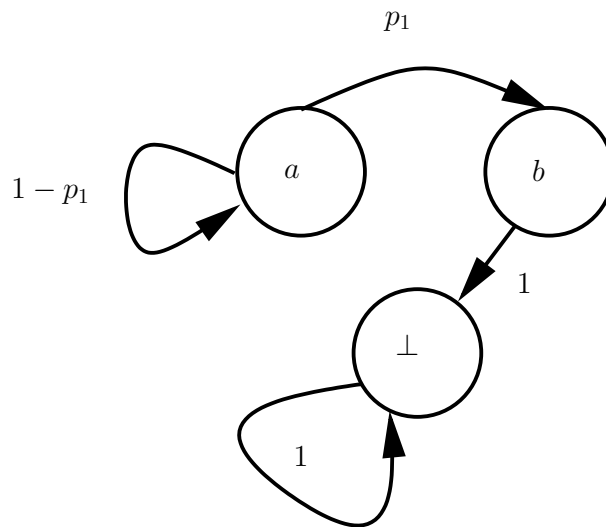


FIG. 5.2 – Comportement du PARS sous une stratégie aléatoire

Exemple 5.2.2. Considérons un autre exemple de système de réécriture probabiliste :

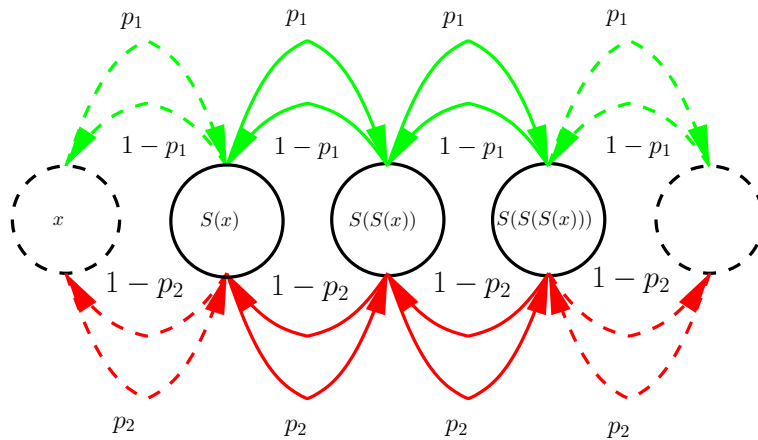


FIG. 5.3 – Relation de réécriture codée dans 5.2.2

$$r_3 := s(x) \rightarrow \begin{cases} x & : p_1 \\ s(s(x)) & : 1 - p_1 \end{cases}$$

$$r_4 := s(x) \rightarrow \begin{cases} x & : p_2 \\ s(s(x)) & : 1 - p_2 \end{cases}$$

Le PRS formé de la règle r_3 , respectivement la règle r_4 , est facilement prouvé étant $+a.s$ terminant si et seulement si $p_1 > 1/2$, respectivement $p_2 > 1/2$. Maintenant considérons le PRS $\{r_3, r_4\}$ avec $p_1 > 1/2$ et $p_2 < 1/2$. Là le PRS n'est pas $+a.s$ terminant : Il suffit de prendre la stratégie qui renvoie pour toute histoire non terminale la distribution qui met en relation $s(x)$ avec la distribution qui charge x avec probabilité $1 - p_1$ et $s(s(x))$ avec probabilité p_1 . Cela montre que le PRS ne termine pas $+a.s$ sous cette stratégie, voir la remarque 3.2.1. Par contre, si de la même manière on considère la stratégie qui retourne pour toute histoire non terminale la règle r_4 , alors toujours en se référant à la remarque 3.2.1, le PRS termine $+a.s$ sous cette stratégie.

Pour donner une intuition de ce que peut être une stratégie sous laquelle un PRS termine $+a.s$, on peut considérer que l'ensemble des stratégies sous lesquelles les PRS terminent $+a.s$ sont celles qui permettent de choisir les règles qui rapprochent des termes terminaux plus souvent que celles qui réécrivent les termes vers des termes "plus distants des termes terminaux". En ce qui concerne l'exemple 5.2.2, considérons une stratégie Markovienne aléatoire ψ qui applique la règle r_3 avec probabilité p et la règle r_4 avec probabilité $1 - p$. Sous cette stratégie, le système de réécriture probabiliste considéré se comporte comme le système de réécriture probabiliste ci dessous :

$$s(x) \rightarrow \begin{cases} x & : p_1 * p + p_2 * (1 - p) \\ s(s(x)) & : (1 - p_1) * p + (1 - p_2) * (1 - p) \end{cases}$$

Ce système de réécriture probabiliste est $+a.s$ terminant si et seulement si $p_1 * p + p_2 * (1 - p) > 1/2$. On en déduit que le PRS de l'exemple 5.2.2 termine positivement et presque sûrement sous la stratégie ψ si l'inégalité $p < (1 - 2p_2)/(2(p_2 - p_1))$ est vraie. Dans cet exemple, quand l'application des règles du PRS étudié est contrôlée par la stratégie ψ , celui-ci se comporte comme la chaîne de Markov représentée dans la figure 5.4.

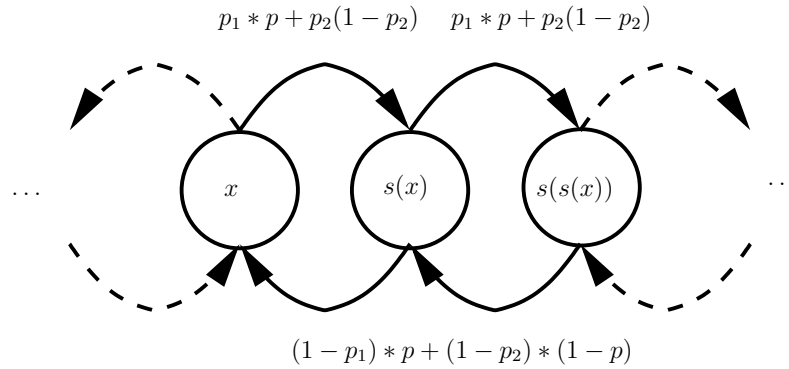


FIG. 5.4 – Comportement du PRS sous une stratégie Markovienne aléatoire

Exemple 5.2.3. Soit le système de réécriture probabiliste composé de quatre règles de réécriture probabilistes, où $p_1 > 0$, $p_2 > 0$.

$$A \rightarrow \begin{cases} B & : p_1 \\ A & : 1 - p_1 \end{cases}$$

$$B \rightarrow \begin{cases} A & : p_2 \\ B & : 1 - p_2 \end{cases}$$

$$A \rightarrow \{C : 1\}$$

$$B \rightarrow \{C : 1\}$$

Le système de réécriture probabiliste se comporte de la même façon que le processus de décision Markovien présenté dans la figure 5.5.

Dans le cas où un PRS est défini sur les seuls termes A , B et C et que dans chacun de ses états une stratégie choisit toujours une règle parmi les deux premières et jamais une des deux dernières, il est clair que l'on obtient sous une telle stratégie des dérivations infinies où apparaissent infiniment souvent les termes A et B . Naturellement, l'existence d'une telle stratégie montre que ce système de réécriture probabiliste n'est pas $+a.s$ terminant. Les termes atteints sont toujours ceux qui sont contenus dans le cadre en pointillés. Cependant, dès lors qu'une famille de stratégies

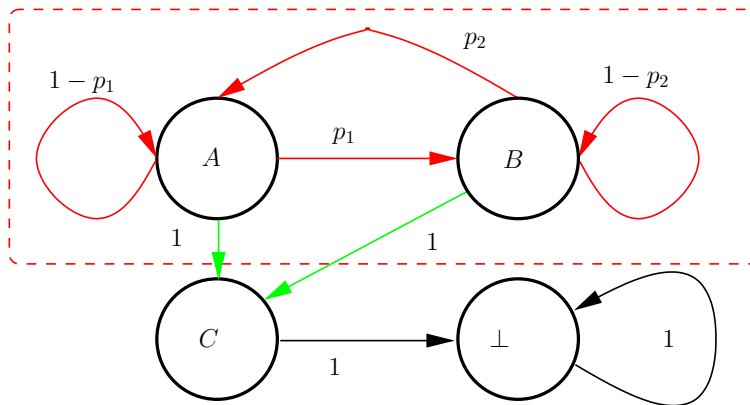


FIG. 5.5 – Comportement du PRS décrit dans l'exemple 5.2.3.

ne contient pas la stratégie précédente, le système est au moins presque sûrement terminant sous n'importe quelle stratégie de la famille. Cet exemple illustre qu'il faut parfois restreindre l'étude de la terminaison à l'ensemble des stratégies qu'on appelle « juste » [HSP83, HS85], c'est à dire celles qui n'appliquent pas de manière récurrente les règles de réécritures qui font en sorte que le processus de réécriture se comporte comme une composante irréductible d'une chaîne de Markov. Le concept de stratégie juste correspond à casser les choix qui enferment le processus de réécriture dans des "cycles" ou composantes connexes en appliquant toute les règles possibles dès lors que l'on passe de manière répétée sur le même terme.

5.3 Terminaison presque sûre sous stratégie.

Nous allons présenter dans cette section des critères permettant de conclure à la terminaison en temps moyen fini des PRS sous stratégie.

Théorème 5.3.1. *Soit Φ une classe de stratégies.*

Un PARS $\mathcal{A} = (A, \rightarrow)$ est +a.s terminant sous la classe de stratégies Φ si il existe une fonction $V : A \rightarrow \mathbb{R}$, avec $\inf_{i \in A} V(i) > -\infty$ et un $\epsilon > 0$, tels que pour toute histoire non terminale et réalisable $h = a_0 a_1 \cdots a_n$, pour tout $\phi \in \Phi$, le drift en h suivant la stratégie ϕ défini par

$$\Delta_\phi V(h) = \sum_{t \in A} \phi(h)(t) V(t) - V(a_n)$$

vérifie l'inégalité

$$\Delta_\phi V(h) \leq -\epsilon.$$

Démonstration. Nous allons appliquer le théorème 2.3.6 pour montrer que sous les hypothèses du théorème, le PARS atteint un état terminal au bout d'un nombre de transition fini, quelque soit la stratégie choisissant le successeur.

On étend V à $A \cup \{\perp\}$ avec $V(\perp) = 0$, et sans perte de généralités, on suppose que $\inf_{a \in A} [V(a)] = C > \epsilon$. En effet, si C est inférieur à ϵ , il suffit de considérer la fonction $V'(t) := V(t) + K$ avec $K > -C + \epsilon$ pour se ramener au cas du début. Pour tout $a \in A$ et pour tout $a \rightarrow \mu$ le drift de V' évalué en a suivant μ est égal au drift de V évalué en a et suivant μ , en effet :

$$\begin{aligned} \Delta_\mu V'(a) &= \sum_{t \in A} \mu(t) [V(t) + K] - [V(a) + K] \\ \Delta_\mu V'(a) &= \sum_{t \in A} \mu(t) [V(t)] - V(a) + K - K \\ \Delta_\mu V'(a) &= \sum_{t \in A} \mu(t) [V(t)] - V(a) = \Delta_\mu V(a). \end{aligned}$$

Soit $a_0 \in A$ un élément de A . Posons $S_0 = V(a_0)$ et soit ϕ une stratégie de la famille Φ . On pose $S_n = V(\pi_n)$ avec π_n le n -ième successeur de a_0 de la dérivation $(\pi)_{n \in \mathbb{N}}$ partant de a_0 et suivant ϕ . On note τ le premier temps d'entrée de $(S_n)_{n \in \mathbb{N}}$ dans l'intervalle $[0, \epsilon]$ qui correspond également au plus petit entier $n + 1$ tel que $\pi_n = s$ avec s un état terminal. Si $\pi_n = s$ est terminal alors le drift de V en π_n est inférieur ou égal à $-C$. La variable aléatoire τ satisfait la propriété suivante $\forall n > \tau$ $\pi_n = \perp$ et $V(\pi_n) = 0$.

Comme pour toute stratégie $\phi \in \Phi$, pour tout $a \in A$ non terminal on a $\Delta_{\phi(a)} V(a) \leq -\epsilon$, alors si $\pi_n = a_n$ est non terminal on a

$$E[V(\pi_{n+1}) | \pi_n = a_n, \dots, \pi_0 = a_0] \leq V(a_n) - \epsilon,$$

c'est à dire

$$E[S_{n+1} | \pi_n = a_n, \dots, \pi_0 = a_0] \leq S_n - \epsilon \times 1_{\{\tau > n\}}.$$

En rappelant qu'on note $\tilde{S}_n = S_{n \wedge \tau}$, et en considérant la filtration $\mathcal{F}_n = \sigma(\pi_0, \dots, \pi_n)$ on a l'inégalité suivante

$$E[\tilde{S}_{n+1} | \mathcal{F}_n] \leq \tilde{S}_n - \epsilon \times 1_{\{\tau > n\}},$$

car si $\tau > n$ on a $S_n = \tilde{S}_n$ et si $\tau \leq n$ on a $\tilde{S}_{n+1} = \tilde{S}_n = S_\tau = 0$.

Donc d'après le théorème 2.3.6

$$E[\tau] \leq \frac{S_0}{\epsilon}$$

ce qui montre bien que sous les hypothèses du théorème, les dérivation d'un PARS sous les stratégies de Φ ont bien une longueur moyenne bornée. □

5.3.1 Généralisation

Nous allons maintenant montrer qu'il existe une forme plus générale de ce résultat, permettant de considérer des fonctions dont le drift n'est pas nécessairement négatif pour toutes les règles. S'il est garanti que l'évaluation moyenne des termes diminue

$$E[V(X_n)|X_{n-1}, \dots, X_0]$$

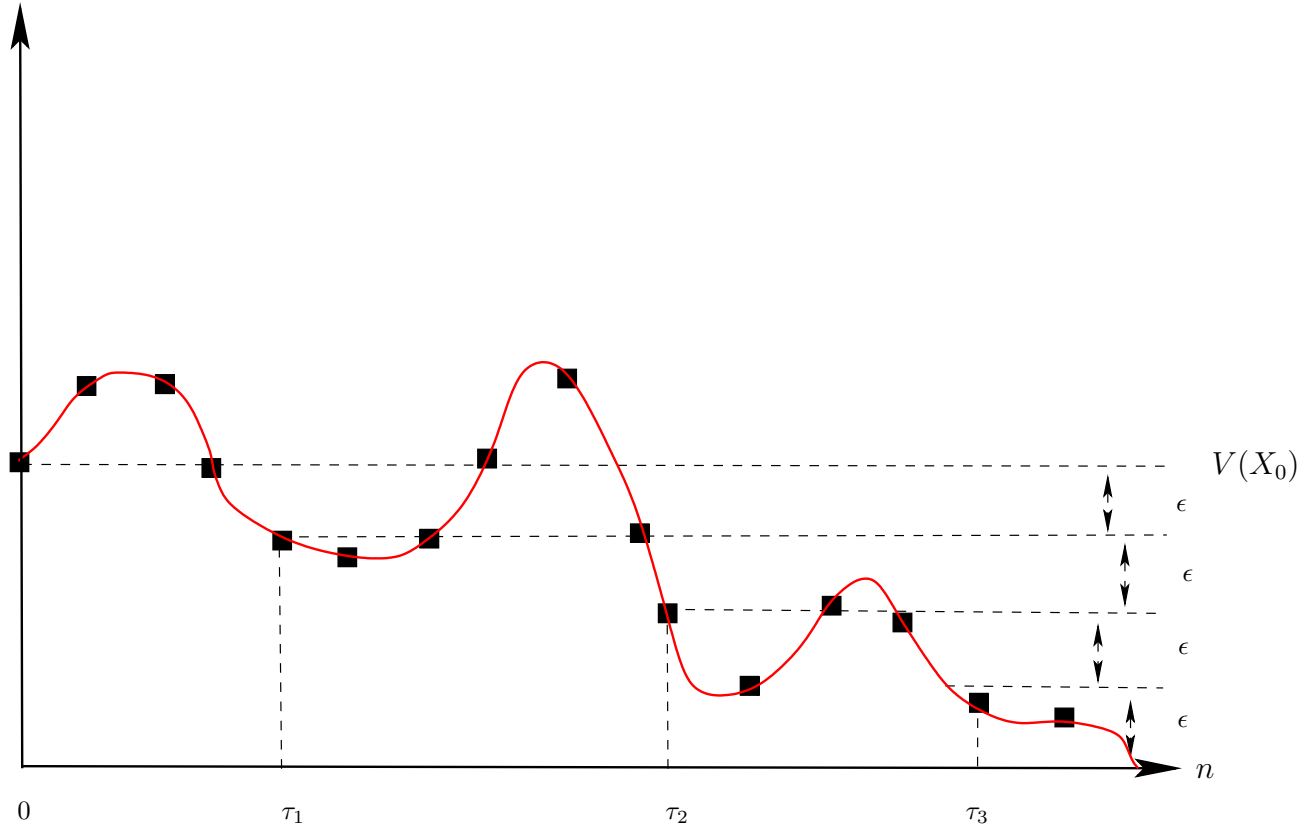


FIG. 5.6 – Trajectoire suivie par l'évaluation de termes successifs.

après un certain nombre de pas de réécriture, alors il est possible de tolérer que la moyenne de cette évaluation puisse augmenter lors des premières applications des règles de réécriture. Cela permet de travailler avec des fonctions d'évaluation pour lesquelles il existe des règles dont le drift est positif. Cependant, pour que la moyenne de l'évaluation des dérivations à un indice fini finisse par devenir inférieure à l'évaluation du terme initial, il faut apporter certaines garanties à propos de la fréquence d'application de certaines classes de règles par rapport à d'autres. Intuitivement, on espère que les règles dont le *drift* est négatif vont s'appliquer "plus souvent" que celles dont le *drift* est positif, moyennant bien sûr une pondération de la fréquence d'application de ces règles en fonction des drifts. On pourra ainsi garantir que l'évaluation des termes d'une dérivation atteindra bien la borne inférieure de la fonction de valuation. Pour illustrer notre réflexion, supposons que l'évaluation moyenne d'une dérivation en fonction des tirages passés pour une stratégie fixée – qui est bien comme nous allons le voir une variable aléatoire – suive la trajectoire représentée par les "carrés" sur la courbe :

Sur la figure 5.6 les τ_i représentent les plus petit indices pour lesquels la variable

aléatoire $E[V(X_n)|X_{n-1}, \dots, X_0]$ est inférieure à $V(X_0) - i \times \epsilon$. Nous allons également expliquer pourquoi les $(\tau_i)_{i \in \mathbb{N}}$ sont des variables aléatoires et quelles propriétés ces variables vont devoir vérifier pour garantir la terminaison en temps moyen fini. Intuitivement, si on parvient à montrer que la différence moyenne entre deux indices successifs des τ_i est finie, c'est à dire $E[\tau_{i+1} - \tau_i] < +\infty$, alors la valuation de $V(X_n)$ atteindra sa borne inférieure en un temps moyen fini. Pour parler de ce résultat, nous devons préalablement énoncer un ensemble de définitions et de propriétés.

Pour un PARS $\mathcal{A} = (A, \rightarrow)$, notons $Dist(\mathcal{A})$ pour l'ensemble des distributions de probabilité μ telle que $a \rightarrow \mu$ pour un certain $a \in A$. On considère que cet ensemble de distributions est partitionné en un nombre fini de sous ensembles, de la manière suivante $Dist(\mathcal{A}) = D_1 \cup D_2 \cup \dots \cup D_k$. En considérant que le PARS \mathcal{A} est généré par un PRS, selon le mécanisme induit par la relation de réécriture probabiliste décrite dans la définition 4.2.3, l'intuition est que chaque D_i contient les distributions obtenues par l'application d'une règle de réécriture $R_i : D_i$ correspond à toutes les distributions résultantes de l'application de la règle de réécriture R_i à une position p quelconque et sous une substitution σ quelconque.

Définition 5.3.1 (Prochaine sélection d'une règle). Soit un D_i fixé. Soit une stratégie ϕ et $h = a_0 a_1 \dots a_n$ une histoire réalisable et non terminale.

La prochaine sélection d'une règle de D_i sachant h est la variable aléatoire $\tau_{D_i, \phi/h}$, dénotant le plus petit index supérieur à n , tel que D_i soit sélectionné à cet index, ou bien qu'un état terminal soit atteint : Formellement, on définit

$$\tau_{D_i, \phi/h} = \begin{cases} \infty & \text{si } \forall m > n \ \pi_m \neq \perp \text{ et } \phi(\pi_0 \dots, \pi_m) \notin D_i \\ n & \text{si } \pi_n = \perp \\ m & \text{si } \begin{cases} \forall n < k < m \ \pi_k \neq \perp \text{ et } \phi(\pi_0 \dots \pi_k) \notin D_i \\ \text{et} \\ \pi_m = \perp \text{ ou } \phi(\pi_0 \dots \pi_m) \in D_i \end{cases} \end{cases}$$

Propriété 5.3.1. Chaque variable aléatoire $\tau_{D_i, \phi/h}$ est un temps d'arrêt par rapport à la filtration induite par la dérivation π . En effet il s'agit d'une variable aléatoire à valeur dans $\mathbb{N} \cup \{\infty\}$, telle que pour tout entier positif m , l'événement $\{\tau = m\}$ s'exprime en fonction de $\pi_0, \pi_1, \dots, \pi_m$.

On généralise la précédente notion de prochaine sélection d'une règle de la manière suivante :

Définition 5.3.2 (La n -ième sélection d'une règle). Soit un D_i fixé. Soit une stratégie ϕ et $h = a_0 \dots a_n$ une histoire réalisable sous la stratégie ϕ et non terminale. Soit $(\pi_i)_{i \in \mathbb{N}}$ une dérivation dont les premières réalisations correspondent à l'histoire h . La n -ième sélection de D_i , noté $I_{D_i, \phi}[\pi, n]$, est définie de la façon suivante :

$$\begin{aligned} I_{D_i, \phi}[\pi, 0] &= 0 \\ I_{D_i, \phi}[\pi, n+1] &= \tau_{D_i, \phi/\pi_0 \dots \pi_{I_{D_i, \phi}[\pi, n]}} \end{aligned}$$

On note I_n à la place de $I_{D_i, \phi}[\pi, n]$ quand D_i, ϕ, π sont clairs.

Définition 5.3.3 (Temps Moyen de Selection Borné). Une classe de stratégies Φ à un temps moyen de sélection borné $\alpha \in \mathbb{R}$ pour D_i si pour toute histoire, l'espérance du nombre de règles à appliquer pour atteindre un état terminal ou pour sélectionner une règle de D_i est inférieur à α : Formellement, pour tout histoire réalisable $h = a_0 \cdots a_n$, pour toute stratégie $\phi \in \Phi$, $\tau_{D_i, \phi/h}$ a une espérance finie avec

$$E[\tau_{D_i, \phi/h}] \leq n + \alpha.$$

On rappelle qu'une variable aléatoire à valeur dans $\mathbb{N} \cup \{\infty\}$ avec une espérance finie est nécessairement finie presque sûrement, c'est à dire que si les conditions de la définition précédente sont satisfaites alors à partir d'une histoire h , on finit toujours par atteindre un état terminal ou sélectionner une règle de D_i avec probabilité 1.

Définition 5.3.4 (Espérance de V au Temps τ). Soit $V : A \rightarrow \mathbb{R}$ une fonction. Soit $\tau \in \mathbb{N} \cup \{\infty\}$ un temps d'arrêt presque sûrement fini : $P(\tau < \infty) = 1$. Soit une dérivation $(\pi_i)_{i \in \mathbb{N}}$.

Notons par $E_\tau V$ l'espérance de V au temps τ :

$$E_\tau V = E[V(\pi_\tau)]$$

lorsque cette valeur existe.

Théorème 5.3.2 (Terminaison Presque sûre sous Stratégie). Soit une classe de stratégies Φ . Un PARS $\mathcal{A} = (A, \rightarrow)$ est presque sûrement terminant sous la famille de stratégies Φ s'il existe une fonction $V : A \rightarrow \mathbb{R}$, avec $\inf_{i \in A} V(i) > -\infty$, un $\epsilon > 0$, et un D_i tels que pour toute stratégie $\phi \in \Phi$, pour toute histoire réalisable non terminale $h = a_0 \dots a_n$,

1. le temps d'arrêt $\tau_{D_i, \phi/h}$ est presque sûrement fini, i.e. $P(\tau_{D_i, \phi/h} < \infty) = 1$,
- 2.

$$E_{\tau_{D_i, \phi/h}} V \leq V(a_n) - \epsilon.$$

Démonstration. En reprenant la démonstration du théorème 3.3.1, on peut supposer sans pertes de généralités que $V(a) \geq C$, avec $C = 2\epsilon$, et que V est étendu sur $A \cup \{\perp\}$ avec $V(\perp) = 0$. Nous allons montrer grâce au théorème 2.3.6 que la suite de l'évaluation des termes successifs atteint zéro avec probabilité 1.

Soient $\phi \in \Phi$ une stratégie, h une histoire réalisable et non terminale.

Pour $(\pi_n)_{n \in \mathbb{N}}$ une dérivation, on note S_n la suite définie par $S_n = V(\pi_n)$. On définit les variables aléatoires suivantes :

$$\tau = \begin{cases} \infty & \text{si } \forall n \in \mathbb{N} \ \pi_n \neq \perp \\ \inf\{n \in \mathbb{N} \mid \pi_n = \perp\} & \text{sinon} \end{cases}$$

$$Y_n = S_{I_{D_i, \phi}[\pi, n]}$$

$$\tilde{Y}_n = S_{I_{D_i, \phi}[\pi, n] \wedge \tau}$$

Par définition et par la propriété 1 :

- Lorsque $\pi_{I_{D_i, \phi}[\pi, n]} = \perp$ on a $\pi_{I_{D_i, \phi}[\pi, n+1]} = \perp$,
- Lorsque $\pi_{I_{D_i, \phi}[\pi, n]} \neq \perp$ on a $I_{D_i, \phi}[\pi, n+1] < \infty$ presque sûrement.

Donc presque sûrement, pour tout n $I_n < \infty$, ce qui est suffisant pour dire que les suites Y_n et \tilde{Y}_n sont bien définies presque sûrement.

On note ici $\mathcal{F}_n = \sigma(\pi_0, \dots, \pi_n)$. Pour tout $n \in \mathbb{N}$ tel que $\pi_{I_n} \neq \perp$, on a sachant les tirages des variables aléatoires π_0, \dots, π_{I_n} – c'est-à-dire \mathcal{F}_{I_n} – l'inégalité suivante :

$$E_{I_{n+1}} V \leq V(\pi_{I_n}) - \epsilon \quad (5.1)$$

En effet, il suffit de considérer l'histoire $h' = \pi_0, \dots, \pi_{I_{D_i, \phi}[\pi, n]}$ et d'appliquer l'hypothèse 2 sur $E_{\tau_{D_i, \phi/h'}}$.

Cela s'écrit encore :

$$E[\tilde{Y}_{n+1} | \mathcal{F}_{I_n}] \leq \tilde{Y}_n - \epsilon 1_{\{\tau > I_n\}} \quad (5.2)$$

et comme la famille de tribus $(\mathcal{F}_{I_n})_{n \in \mathbb{N}}$ est bien croissante pour l'inclusion, on peut appliquer le théorème 2.3.6 sur la suite de variable aléatoire $(\tilde{Y}_n)_{n \in \mathbb{N}}$ et donc conclure que si $\tau' = \inf\{n \in \mathbb{N} \mid \pi_{I_n} = \perp\}$, alors

$$E[\tau'] \leq \frac{Y_0}{\epsilon} \quad (5.3)$$

ce qui implique :

$$P(\tau' < \infty) = 1. \quad (5.4)$$

D'où

$$P(\tau < \infty) = 1 \quad (5.5)$$

ce qui achève la preuve. \square

Remarque 5.3.1. Les hypothèses du théorème précédent permettent de dire qu'un PARS termine presque sûrement sous une famille de stratégies. Néanmoins, elles ne suffisent pas pour impliquer la terminaison en temps moyen fini, c'est-à-dire la terminaison *+a.s.* Pour voir cela, reprenons le moment dans la preuve où l'on construit une sous-suite d'index I_n qui implique la terminaison presque sûre. Il n'y a rien qui garantit que pour tout n la différence $I_{n+1} - I_n$ soit bornée en moyenne et donc que l'index du terme terminal soit fini en moyenne. Pour illustrer cela, nous allons coder une marche aléatoire sur \mathbb{N}^2 qui termine presque sûrement mais pas positivement presque sûrement.

Exemple 5.3.1. Soit le PARS $(T(\Sigma, X), \rightarrow_{\mathcal{R}})$ composé des trois règles probabilistes suivantes :

$$l_1 : (a + 1, b + 1) \rightarrow \begin{cases} (a + 1, b + 2) & : \frac{1}{2} \\ (a + 1, b) & : \frac{1}{2} \end{cases}$$

$$l_2 : (a + 1, 0) \rightarrow \begin{cases} (a, 10) & : \frac{2}{3} \\ (a + 2, 10) & : \frac{1}{3} \end{cases}$$

$$l_3 : (a + 1, b + 1) \rightarrow \begin{cases} (a + 1, b + 2) & : \frac{2}{3} \\ (a + 1, b) & : \frac{1}{3} \end{cases}$$

Il existe une stratégie et une fonction V sous lesquelles ce système termine presque sûrement mais le système ne termine pas $+a.s$.

Démonstration. Soit $V : \mathbb{N}^2 \rightarrow \mathbb{N}$ la fonction qui à $V(a, b)$ associe a . On remarque ici que l'ensemble des termes terminaux est l'ensemble des termes $(0, b)$ avec $b \in \mathbb{N}$. Soit ϕ la stratégie qui choisit toujours d'appliquer la règle l_1 lorsque l_1 et l_3 peuvent s'appliquer. On suppose données une histoire h de longueur n , une dérivation $(\pi_n)_{n \in \mathbb{N}}$ dont les n premières réalisations correspondent à l'histoire h . Soit $\tau_{D_2, \phi/h}$ le temps d'arrêt correspondant à l'indice du prochain déclenchement de la règle l_2 . Il est facile de constater que :

$$E_{\tau_{D_2, \phi/h}} V \leq V(a_n) - \frac{1}{3}$$

ce qui satisfait l'hypothèse 2 du théorème 5.3.2. L'hypothèse 1 du théorème 5.3.2 est vérifiée : en effet, à partir de $(a + 1, 0)$, la règle l_2 envoie sur un couple de la forme $(-, 10)$, à partir duquel seul la règle l_1 va être appliquée jusqu'à revenir sur $(-, 0)$. La règle l_1 correspondant à une marche aléatoire symétrique, avec probabilité 1 cela se produira. Néanmoins,

$$E[\tau_{D_2, \phi/h}] = \infty$$

si le dernier élément de h est un couple de la forme $(a + 1, b)$, ce qui montre bien que l'on a une sous-partie de la dérivation qui a une longueur dont la moyenne est infinie. \square

On trouve dans [HS85] des conditions plus faibles entraînant aussi la terminaison presque sûre, en particulier on peut appliquer les théorèmes cités et présentés dans ce manuscrit avec $\epsilon = 0$. Cependant, dès que l'on cherche à prouver la terminaison presque sûre positive, on doit renforcer nos hypothèses.

Théorème 5.3.3 (Terminaison Presque Sûre Positive Sous Stratégie). *Soit une classe de stratégies Φ . Un PARS $\mathcal{A} = (A, \rightarrow)$ est $+a.s$ terminant sous les stratégies de Φ s'il existe une fonction $V : A \rightarrow \mathbb{R}$, avec $\inf_{i \in A} V(i) > -\infty$, un $\epsilon > 0$, et un D_i tel*

que Φ a un temps moyen de sélection fini pour D_i et que pour toute stratégie $\phi \in \Phi$, pour toute histoire non terminale et réalisable $h = a_0 \dots a_n$ on ait :

$$E_{\tau_{D_i, \phi/h}} V \leq V(a_n) - \epsilon.$$

Démonstration. Le fait que Φ ait un temps moyen de sélection borné pour D_i implique la condition 1 du théorème précédent, ce qui implique la terminaison presque sûre. Dans la preuve du théorème précédent l'inégalité 5.3 permet d'affirmer que la variable aléatoire τ' donnant le plus petit entier n tel que $\pi_{I_n} = \perp$ a une espérance finie vérifiant l'inégalité $E[\tau'] \leq V(a_n)/\epsilon$. La variable aléatoire τ satisfait nécessairement $\tau = I_{\tau'}$ et $I_{\tau'} = \sum_{i=0}^{\tau'} I_{i+1} - I_i$ on a :

$$E[\tau] = E \left[\sum_{i=0}^{\tau'} I_{i+1} - I_i \right] \quad (5.6)$$

Comme par hypothèse on a pour tout $0 \leq i < \tau'$ l'inégalité $E[I_{n+1} - I_n] < \alpha$, on obtient à partir de l'équation 5.6 l'inégalité 5.7 :

$$E[\tau] \leq E \left[\sum_{i=0}^{\tau'} \alpha \right] \quad (5.7)$$

$$E[\tau] \leq \alpha \times E[\tau'] \quad (5.8)$$

Comme $E[\tau'] \leq \frac{V(a_n)}{\epsilon}$, cela montre bien que le PARS est *a.s* terminant puisque $E[\tau] \leq \alpha \times \frac{V(a_n)}{\epsilon}$.

□

5.4 Applications

Exemple 5.4.1. Soit PARS $(T(\Sigma, X), \rightarrow_{\mathcal{R}})$ composé des trois règles suivantes R_1, R_2, R_3 :

$$\begin{aligned} R_1 : (X, Y + 1) &\rightarrow \begin{cases} ((X + 1)\%3, Y) & : \frac{2}{3} \\ ((X - 1)\%3, Y) & : \frac{1}{3} \end{cases} \\ R_2 : (X, Y + 1) &\rightarrow \begin{cases} (X + 2, Y + 1) & : \frac{1}{2} \\ (X - 2, Y + 1) & : \frac{1}{2} \end{cases} \\ R_3 : (0, Y + 1) &\rightarrow \{ (0, Y) : 1 \} \end{aligned}$$

Les termes terminaux pour ce système de réécriture sont les termes de la forme $(X, 0)$. Ce système de réécriture probabiliste n'est pas presque sûrement terminant. Pour le voir, il suffit de considérer les stratégies qui appliquent systématiquement les deux premières règles et jamais la troisième.

Proposition 5.4.1. *Il existe une classe de stratégies sous lesquelles le PRS précédent termine positivement et presque sûrement.*

Démonstration. Pour montrer cette propriété, nous allons considérer la fonction $V : T(\Sigma, X) \rightarrow \mathbb{R}^+$ telle que :

$$V((X, Y)) = Y.$$

Le drift de chaque règle vaut :

$$\begin{aligned} \Delta_{R_1} V &= 0 \\ \Delta_{R_2} V &= 0 \\ \Delta_{R_3} V &= -1. \end{aligned}$$

Soit ϕ une stratégie, $h = a_0 \dots a_n$ une histoire réalisable sous ϕ et non terminale. On remarque que tant que ϕ sélectionne soit la règle R_1 ou la règle R_2 , l'évaluation du terme courant par V reste constant. Cependant, quand on applique la règle R_3 , on fait diminuer de 1 l'évaluation du terme courant par la fonction V .

Soit D_3 l'ensemble des distributions de probabilités sur les termes associés à l'application des règles de réécriture R_3 . Quand ϕ applique R_3 , le drift de l'évaluation vaut -1 . C'est à dire que si on note τ_{D_3} le prochain temps de sélection de D_3 , on a :

$$E_{\tau_{D_3}, \phi/h} \leq V(a_n) - 1.$$

Le théorème 5.3.3 permet de d'affirmer que si la règle R_3 a un temps de sélection moyen borné pour la stratégie ϕ alors le système de réécriture termine positivement et presque sûrement sous la stratégie ϕ . \square

Troisième partie
Exemple d'application

6

Produit synchronisé d'automates temporisés

Après un bref rappel à propos des automates temporisés, nous allons définir un modèle de produit synchronisé par passage de messages permettant de simuler les phénomènes de stations cachées qui interviennent dans les réseaux WIFI. Nous utiliserons ce modèle dans le chapitre 7 pour montrer que le protocole termine en temps moyen fini.

6.1 Les Automates temporisés à la Alur et Dill

Ce modèle d'automates introduit dans [ACD93] a permis d'apporter un cadre formel à l'étude des « processus à temps réel ». Il est suffisamment général et concis pour permettre d'exprimer facilement des systèmes complexes tels que des calculs de plages horaires pour la navigation aérienne et le trafic ferroviaire. Ce modèle a de même permis l'essor du « *model checking* » de protocoles ou de programmes temps réel. Ils ont été plus tard étendus pour permettre d'exprimer des systèmes à temps réel et probabilistes. Pour notre part nous les présentons dans ce document car nous allons étendre ce formalisme de façon à pouvoir décrire simplement de quelle manière nous simulerons un réseau de stations WIFI.

6.1.1 Séquence temporelle

Une des principales notions qu'il est important de clarifier avant d'entreprendre l'étude des automates temporisés est le sens que l'on donne à la notion de « temps ».

Définition 6.1.1 (Séquence temporelle). Une séquence temporelle est une suite de réels $(\tau_i)_{i \in \mathbb{N}} \in \mathbb{R}^+$ satisfaisant les deux priorités suivantes :

$$\begin{aligned} \forall i \in \mathbb{N} \quad \tau_i < \tau_{i+1} & \quad (\text{Monotonie}) \\ \forall t \in \mathbb{R} \quad \exists i \text{ tq } \tau_i > t & \quad (\text{Progression}) \end{aligned}$$

Cette définition garantit que le temps avance toujours dans le même sens et ne converge pas vers une limite finie.

Définition 6.1.2 (Mot temporisé). Un mot temporisé sur un alphabet Σ est une paire (σ, τ) avec $\sigma = \sigma_1\sigma_2\dots$ un mot infini sur Σ et τ une séquence temporisée.

Définition 6.1.3 (Langages temporisés). Un langage temporisé sur un alphabet Σ est un ensemble de mots temporisés sur Σ .

6.1.2 Contraintes et opérations sur les horloges

Les horloges peuvent être remises à zéro par les transitions. Ces transitions sont conditionnées par des contraintes particulières. Alur et Dill ont introduit la notion de table de transitions temporisée pour définir la sémantique des automates temporisés. Ces tables associent à certaines places un ensemble d'horloges, des contraintes temporelles dépendant de ces horloges et des actions permettant de choisir une transition parmi celles possibles. À chaque transition, un ensemble d'opérations permet de modifier la valeur des horloges, typiquement de réinitialiser la valeur des horloges à zéro.

Ainsi si un mot est lu en entrée par un automate temporisé, seules les transitions correspondant à ce mot et pour lesquelles les contraintes d'horloges sont satisfaites peuvent s'appliquer. Si à un instant déterminé aucune transition ne peut se déclencher alors le système demeure dans la place courante jusqu'à ce que le temps ait suffisamment avancé pour que les contraintes d'horloges soient satisfaites pour une transition.

Définition 6.1.4 (Horloges). Une horloge est une variable de \mathbb{R}^+ . On appelle le temps T une horloge de référence. Les horloges progressent à la même vitesse que le temps et de façon synchrone avec le temps.

Définition 6.1.5 (Date). La date est la valeur contenue par l'horloge de référence T au moment où on l'observe.

Remarque 6.1.1. Soit t_1 une horloge. Si à la date T l'horloge t_1 satisfait $t_1 = d$ et si aucune remise à zéro n'est effectuée sur t_1 alors à la date $T = d + d'$ on a $t_1 = d + d'$.

Chaque horloge peut en fait être vue comme une copie de l'horloge T représentant le temps. Leurs valeurs augmentent de façon synchrone. Chaque horloge est initialisée à zéro en début d'exécution.

Définition 6.1.6 (Contraintes d'horloges). Pour un ensemble X d'horloges, l'ensemble $\Phi(X)$ de contraintes d'horloges δ est défini par induction de la façon suivante :

$$\delta := x \leq c \mid c \leq x \mid \neg\delta \mid \delta_1 \wedge \delta_2$$

avec c une constante rationnelle et x une horloge de X .

Définition 6.1.7 (Interprétation d'horloge). Une interprétation d'horloge ν d'un ensemble d'horloges X assigne à chaque horloge une valeur réelle positive.

En d'autres termes, une interprétation d'horloge est une fonction de X vers \mathbb{R}^+ . On dit qu'une interprétation d'horloge μ de X satisfait une contrainte d'horloge δ si et seulement si δ s'évalue à vrai avec les valeurs données par l'interprétation ν de X .

Exemple 6.1.1. Pour tout réel $t \in \mathbb{R}$, $\mu + t$ représente l'interprétation d'horloge qui à chaque horloge x associe la valeur $\mu(x) + t$. De même pour tout $Y \subseteq X$, $[Y \rightarrow t]\mu$ représente l'interprétation d'horloge qui à toute horloge de Y associe la valeur t et qui à toute horloge $x \in X \setminus Y$ associe $\mu(x)$.

Définition 6.1.8 (Table de transition temporisée). Une table de transition temporisée \mathcal{A} est un 5-uplet $\langle \Sigma, S, S_0, C, E \rangle$ avec :

- Σ un alphabet fini,
- S un ensemble fini de places,
- $S_0 \subseteq S$ est l'ensemble des places de départ,
- C un ensemble fini d'horloges,
- $E \subseteq S \times S \times \Sigma \times 2^C \times \Phi(C)$ définit l'ensemble des transitions. Un arc $\langle s, s', a, \lambda, \delta \rangle$ représente une transition de la place s vers s' quand le symbole d'entrée vaut a . L'ensemble $\lambda \subseteq C$ représente l'ensemble des horloges à réinitialiser et δ est une contrainte d'horloge sur C .

Étant donné un mot temporisé (σ, τ) , la table de transition \mathcal{A} s'initialise dans une de ses places de départ, au temps 0 et toutes les horloges sont mises à zéro. Au fur et à mesure que le temps avance, la valeur de chaque horloge augmente de façon similaire. Au temps τ_i , \mathcal{A} passe de la place s à la place s' en utilisant une transition de la forme $\langle s, s', \sigma_i, \lambda, \delta \rangle$ où les valeurs des horloges satisfont les contraintes δ . Les horloges λ sont alors réinitialisées à zéro.

La table de transition décrit pour chaque exécution un ensemble de choix possibles pour le futur. En effet le modèle permet de définir des comportements non déterministes s'il existe plusieurs éléments dans l'ensemble des transitions $\langle s, s', \sigma_i, \lambda, \delta \rangle$ qui peuvent s'appliquer à un moment déterminé.

Le comportement d'un automate temporisé suivant une table de transition temporisée est mémorisé par ce que l'on appelle des exécutions, où les exécutions sont l'enregistrement des places atteintes associées à la date à laquelle l'automate y est entré.

Définition 6.1.9 (Exécution). Une exécution r , notée $(\bar{s}, \bar{\nu})$, d'une table de transition $\langle \Sigma, S, S_0, C, E \rangle$ pour un mot temporisé (σ, τ) est une séquence infinie de la forme :

$$r : \langle s_0, \nu_0 \rangle \xrightarrow[\tau_1]{\sigma_1} \langle s_1, \nu_1 \rangle \xrightarrow[\tau_2]{\sigma_2} \langle s_2, \nu_2 \rangle \xrightarrow[\tau_3]{\sigma_3} \dots$$

avec $s_i \in S$ et $\nu_i \in [C \rightarrow \mathbb{R}]$, pour tout $i \geq 0$, satisfaisant les deux conditions :

- Initialisation : $s_0 \in S_0$ et $\nu_0(x) = 0$ pour tout $x \in C$.
- Consécution : Pour tout $i \geq 1$, il existe un arc de E de la forme $\langle s_{i-1}, s_i, \sigma_i, \lambda_i, \delta_i \rangle$ tel que $(\nu_{i-1} + \tau_i - \tau_{i-1})$ satisfait les contraintes d'horloge δ_i et ν_i vaut $[\lambda_i \rightarrow 0](\nu_{i-1} + \tau_i - \tau_{i-1})$.

Nous avons à ce stade défini les principales notions nécessaires pour décrire le comportement des automates temporisés. Pour rendre aisée la description de tels automates, il convient de fixer une description graphique représentant les places de ces automates, des transitions codées dans une table de transition temporisée et les opérations sur les interprétations d'horloges et les contraintes d'horloge.

Représentation graphique des automates temporisés

On représente chaque place par un cercle au centre duquel figure le numéro ou le nom de la place représentée. Les transitions décrites par la table de transition temporisée sont représentées par des arcs étiquetés. Les étiquettes de ces arcs sont de trois sortes :

- Un caractère de Σ , par exemple **a**. Une telle étiquette signifie que la transition en question peut se déclencher seulement si l'automate lit (**a**, τ_i) en entrée.
- Une étiquette de la forme $(\delta) ?$ avec $\delta \in \Phi(X)$ une contrainte d'horloge. Cette étiquette signifie que l'automate temporisé dans son état actuel $\langle s_i, \nu_i \rangle$ doit avoir une interprétation d'horloge ν_i qui vérifie $\delta(\nu_i(X))$.
- Les étiquettes de la forme $\mathbf{x} := 0$ avec $\mathbf{x} \in X$ signifient que l'automate dans l'état $\langle s_i, \nu_i \rangle$ transitera vers $\langle s_{i+1}, \nu_{i+1} \rangle$ avec $\nu_{i+1} = [\mathbf{x} \rightarrow 0]\nu_i$.

Illustrons maintenant par quelques exemples les automates temporisés ainsi que les langages qu'ils acceptent et une exécution correspondant à un mot temporisé :

Exemple 6.1.2. L'automate représenté dans la figure 6.1.2 est composé des places S_0, S_1, S_2, S_3 .

La table temporelle de transitions correspondant à cet automate comporte deux horloges x et y . On définit S_0 comme étant la place initiale.

L'automate boucle sur les places S_0, S_1, S_2 et S_3 . L'horloge \mathbf{x} est réinitialisée à zéro à chaque fois que l'automate lit le caractère **a** en entrée et passe de la place S_0 à la place S_1 . À chaque transition de la place S_2 vers la place S_3 , l'automate lit le caractère **c** et vérifie que l'horloge \mathbf{x} est bien inférieure à 1. Cette condition garantit que cette transition s'effectuera au plus tard une unité de temps après que le caractère

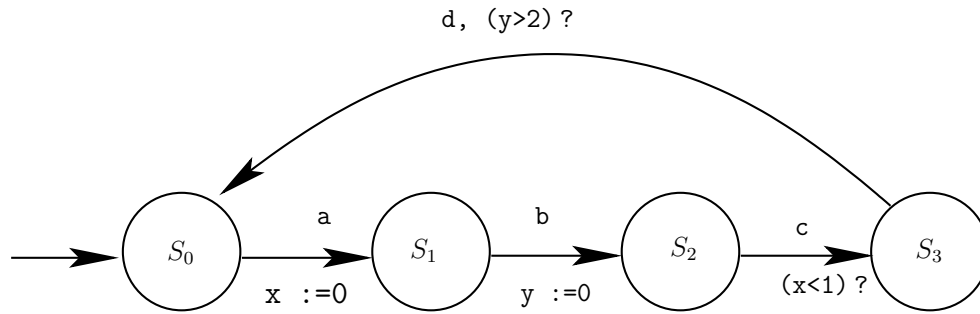


FIG. 6.1 – Un automate temporisé simple

a a été lu. Le même mécanisme est utilisé pour garantir que les transitions entre les places S_3 et S_0 s'effectuent seulement après au moins deux unités de temps après les transitions entre les places S_1 et S_2 .

Cet automate accepte le langage temporisé suivant :

$$L = \{((abcd)^\omega, \tau) \mid \forall j ((\tau_{4j+3} < \tau_{4j+1} + 1) \wedge (\tau_{4j+4} > \tau_{4j+2} + 2))\}.$$

6.2 Composition parallèle d'automates temporisés pour simuler le comportement d'un ensemble de stations WIFI.

Afin de simuler un réseau de stations WIFI [con] où chaque station se comporte comme un automate indépendant et cherche à transmettre des messages à des moments déterminés par l'état d'un canal radio partagé par tous les automates, nous avons étendu le modèle d'automate temporisé introduit par Alur et Dill [ACD93]. Cette solution a été préférée à d'autres modèles tels que les automates temporisés, les p-automates [BF99] ainsi que les automates temporisés probabilistes [Seg95, KGSS02] même si ces derniers sont souvent utilisés dans la communauté model-checking pour modéliser des protocoles réseau, tels que le CSMA-CA et CSMA-CD [KNP02, DFH⁺05].

Nous avons préféré introduire un nouveau formalisme plutôt que de réutiliser celui des automates temporisés probabilistes pour les raisons suivantes :

- On avait besoin d'exprimer la synchronisation par passage de messages d'automates temporisés sur un système où chaque automate ne peut envoyer des message qu'à un sous ensemble des autres automates qui lui est propre. On a introduit un modèle de ressource de synchronisation qui permet de représenter les caractéristiques d'accessibilité d'un réseau radio et qui permet de synchroniser les actions de plusieurs automates temporisés.

- L'ensemble des transitions se décrit très bien grâce au modèle classique d'automates temporisés, puisque les seuls phénomènes probabilistes intervenant concernent le tirage de la période d'observation du canal radio lors de l'exécution de l'algorithme du `backoff` que nous décrirons dans le chapitre 7. Il suffisait juste d'autoriser l'appel d'une fonction représentant le tirage d'une variable aléatoire dans l'expression de contraintes d'horloge.
- On voulait avoir une représentation graphique du comportement des automates indépendante de la topologie du réseau pour rendre la lecture des schémas plus aisée. De même l'application des preuves formelles présentées au cours des chapitres précédents peut se faire plus simplement à partir d'une abstraction de la topologie du réseau.

Une fois cela fait, nous pourrons définir de façon précise la sémantique de la synchronisation des différents automates communicant par ce même canal.

Résumons les fonctionnalités que nous attendons d'un tel modèle : à partir d'un ensemble fini d'automates temporisés $\mathcal{A}_1, \dots, \mathcal{A}_n$ on veut définir la sémantique du produit parallèle $\mathcal{A}_1 || \dots || \mathcal{A}_n$. La synchronisation des divers automates se fera par passage de messages sur un « support » où chaque automate ne pourra envoyer de messages qu'à un sous ensemble déterminé des autres automates. Cette donnée d'accessibilité propre au support sera modélisée sous la forme d'une matrice d'accessibilité.

Modèle de la ressource de synchronisation

On désire définir une ressource pour synchroniser plusieurs automates temporisés $\mathcal{A}_0, \dots, \mathcal{A}_{n-1}$ d'une façon qui permette de simuler le mécanisme de synchronisation utilisé dans les réseaux WIFI. Ces automates liront les caractères d'entrée sur la ressource de synchronisation grâce à une fonction de lecture \mathbf{r} et disposeront d'une fonction d'écriture \mathbf{w} sur cette même ressource.

Pour cela on définit la ressource de synchronisation S de la manière suivante :

Définition 6.2.1 (Ressource de synchronisation). Une ressource de synchronisation à n entrées acceptant un alphabet Σ est un couple $S = (\Sigma, T)$ où $T \in \mathcal{M}_n(\{0, 1\})$ est une matrice carrée de booléens de taille $n \times n$.

Définition 6.2.2 (Accessibilité d'une entrée à partir d'une autre). L'entrée j de la ressource de synchronisation $S = (\Sigma, T)$ est accessible depuis l'entrée i si et seulement si $T[i, j] = 1$.

Exemple 6.2.1. La figure 6.2 représente par un graphe la topologie d'un réseau de communication comportant 7 automates temporisés. Un sommet de ce graphe représente un automate et deux sommets i et j sont connectés si la topologie du réseau permet à l'automate i d'envoyer un message à l'automate j .

La matrice d'accessibilité d'un tel réseau est

$$T = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 \end{pmatrix}.$$

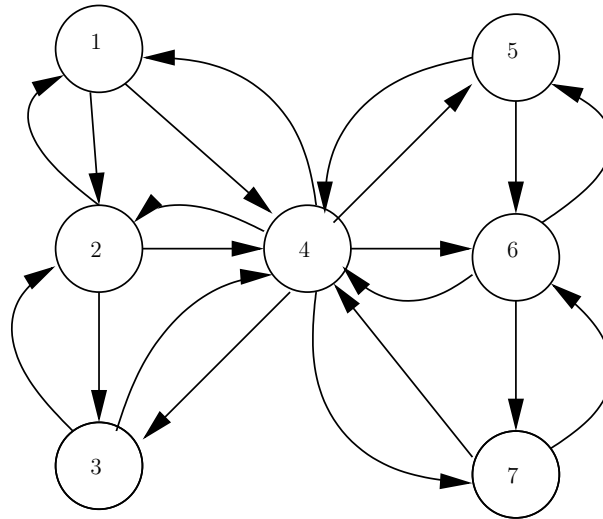


FIG. 6.2 – Topologie d'un réseau de communication

On aura besoin dans notre modèle d'introduire le caractère ϵ correspondant au mot vide.

Afin de définir le passage de messages sur cette ressource, on définit une opération d'écriture ainsi qu'une fonction de lecture.

Définition 6.2.3 (Écriture sur une ressource de synchronisation). On définit l'opération d'écriture w sur une ressource de synchronisation $S = (\Sigma, T)$ à n entrées comme l'opération qui prend les deux paramètres d'entrées suivants : le numéro de l'entrée sur la ressource de synchronisation et le caractère à transmettre.

Exemple 6.2.2. Pour écrire le caractère $a \in \Sigma$ sur l'entrée k de la ressource de synchronisation $S = (\Sigma, T)$, on utilise l'appel de fonction suivant : $w(k, a)$.

Définition 6.2.4 (Lecture sur une ressource de synchronisation). La fonction de lecture r sur une ressource de synchronisation $S = (\Sigma, T)$ à n entrées prend un paramètre d'entrée qui est le numéro de l'entrée à partir de laquelle on lit l'état de

la ressource de synchronisation. La fonction retourne l'ensemble des caractères écrits sur la ressource au moment de la lecture.

$r(k) = \cup_{\{i|T[i,k]=1\}}\{\mathbf{t} \in \Sigma|\mathbf{w}(i, \mathbf{t})\}$ et $r(k) = \emptyset$ si aucun automate n'écrit sur la ressource au moment de la lecture.

Exemple 6.2.3. $\mathbf{r}(\mathbf{k})=\{\mathbf{a}\}$ si et seulement si il existe une unique entrée l vérifiant $T[l, k] = 1$ sur laquelle l'opération $\mathbf{w}(l, \mathbf{a})$ est effectuée simultanément au moment de la lecture.

Produit synchronisé d'automates temporisés

On peut maintenant définir le produit synchronisé d'automates temporisés communiquant grâce à une ressource. Pour cela au lieu de lire un mot temporisé, les différents automates lisent les entrées de la ressource de synchronisation et déclenchent les règles de transition d'une table de transition temporisée synchronisable. Lors de ces transitions les automates peuvent effectuer des opérations d'écriture sur la ressource de synchronisation. Ici le temps est discret et notre modèle permet de modéliser des phénomènes où plusieurs automates du produit changent simultanément d'état.

Chaque automate sera associé à une entrée de la ressource de synchronisation qui lui sera propre et la table d'accessibilité de la ressource de synchronisation permet de définir quels automates sont accessibles à partir d'un automate.

Tout d'abord, l'état initial du produit synchronisé $\mathcal{A}_1 || \dots || \mathcal{A}_n$ est représentée par $\langle s_{1,0}, \nu_{1,0} \rangle || \dots || \langle s_{n,0}, \nu_{n,0} \rangle$ où $\langle s_{k,0}, \nu_{k,0} \rangle$ dénote la place à laquelle se situe le k -ème automate du produit ainsi que l'interprétation d'horloge qui lui est associée. On représente par $\langle s_{1,i}, \nu_{1,i} \rangle || \dots || \langle s_{n,i}, \nu_{n,i} \rangle$ l'état du produit synchronisé après i transitions. Les transitions sont définies par la table de transition temporisée synchronisable que l'on définit ci-dessous :

Définition 6.2.5 (Table de transition temporisée synchronisable). Une table de transition temporisée synchronisable de taille n d'un produit synchronisé $\mathcal{A}_1 || \dots || \mathcal{A}_n$ est un ensemble de n 4-uplets $\langle S_k, S_{k,0}, C_k, E_k \rangle$ représentant chacun un automate \mathcal{A}_k avec :

- S_k est l'ensemble fini des places de A_k ,
- $S_{k,0} \subseteq S_k$ est l'ensemble des places de départ de l'automate k ,
- C_k est l'ensemble fini des horloges du k -ème automate,
- $E_k \subseteq S_k \times S_k \times \mathcal{P}(\Sigma) \times 2^{C_k} \times \Phi(C_k) \times \Sigma$ définit l'ensemble des transitions, et où $\mathcal{P}(\Sigma)$ représente l'ensemble des parties de Σ . Chaque automate A_k lit et écrit sur l'entrée k de la ressource de synchronisation. Un arc $\langle s, s', a, \lambda, \delta, b \rangle$ représente une transition de la place s vers s' quand les symboles lu en entrée valent $a \in \mathcal{P}(\Sigma)$, c'est à dire lorsque $\mathbf{r}(\mathbf{k})=a$. L'ensemble $\lambda \subset C_k$ représente l'ensemble des horloges à réinitialiser et δ est une contrainte d'horloge sur C_k . L'automate écrit b sur la ressource de synchronisation lors de la transition, c'est à dire effectuée l'opération $\mathbf{w}(\mathbf{k}, \mathbf{b})$.

On va décrire la sémantique du produit d'automates temporisés. Pour faire cela, on a besoin de définir la sémantique des fonctions de lecture et d'écriture sur les ressources de synchronisation. 6.3.

$$\begin{array}{c}
\frac{\exists! k \mid T[k, i] = 1 \wedge \mathbf{w}(\mathbf{k}, \mathbf{a}) \wedge (\forall \mathbf{a} \in \Sigma \ \forall j \neq k T[j, k] = 0 \ \vee \mathbf{w}(\mathbf{j}, \mathbf{a}))}{\mathbf{r}(\mathbf{i}) = \{\mathbf{a}\}} \\
\frac{\forall \mathbf{a} \in \Sigma \ \forall k \mid T[k, i] = 0 \ \vee \neg \mathbf{w}(\mathbf{k}, \mathbf{a})}{\mathbf{r}(\mathbf{i}) = \emptyset} \\
\frac{\exists k_1, \dots, k_l \mid T[k_j, i] = 1 \ \forall j \in \{1, \dots, l\} \wedge \mathbf{w}(k_j, a_j) \wedge (\forall j \notin \{k_1 \dots k_l\} \forall \mathbf{a} \in \Sigma \neg \mathbf{w}(\mathbf{j}, \mathbf{a}))}{\mathbf{r}(\mathbf{i}) = \cup_{i \in \{k_1, \dots, k_l\}} \{a_{k_i}\}}
\end{array}$$

FIG. 6.3 – Lecture et écriture sur la ressource de synchronisation

Remarque 6.2.1. On remarque que la première règle décrite dans la figure 6.3 est un cas particulier de la troisième.

Notation 9 (Emploi abusif d' ϵ). *Dans la suite de ce document, on écrira $\mathbf{r}(\mathbf{k}) = \epsilon$ au lieu de $\mathbf{r}(\mathbf{k}) = \emptyset$. On emploiera également la terminologie « mot vide » pour désigner ϵ et l'ensemble vide que ce mot représente dans ce cas.*

On précise que les transitions sont instantanées, comme dans le cas des automates temporisés classiques. De même, lorsque l'on fait référence au temps, on fait référence à une horloge T qui représente le temps de référence. Toutes les autres horloges sont synchrones à T . On numérote les transitions à partir de zéro et on note la date de la i -ème transition par τ_i .

Initialisation du produit synchronisé

Le produit synchronisé commence par une phase d'initialisation où toutes les horloges sont mises à zéro et les différents automates sont positionnés sur une de leurs places initiales. Ainsi le temps global est initialisé à zéro $T := 0$ et quel que soit \mathcal{A}_k parmi les automates du produit $\mathcal{A}_1 \parallel \dots \parallel \mathcal{A}_n$ toutes les horloges $t \in C_k$ de l'automate \mathcal{A}_k sont initialisées à zéro, c'est à dire $t = 0$.

Exécution

L'exécution consiste à laisser le « temps » avancer suffisamment pour arriver à une date où une transition peut se produire et faire avancer le système. On construit ainsi la séquence temporisée² $(\tau)_{i \geq 0}$. Une transition peut se produire s'il existe un des

²Partielle si le produit synchronisé arrive dans un état de deadlock.

automates du produit qui peut lire un caractère sur son entrée de la ressource de synchronisation qui correspond à une transition décrite par la table de synchronisation temporisée à une date où ses contraintes d'horloges sont satisfaites.

Pour faire cela, on définit un opérateur booléen permettant de savoir si un automate du produit est actif à une date donnée et par rapport à un élément de la table de transitions synchronisée.

Définition 6.2.6 (Automate actif). Le fait que le k -ième automate du produit synchronisé s_k soit actif pour la transition $t = (s_k, s', a, \lambda, \delta, b)$ au temps T est défini par :

$$\frac{\delta(\nu_{k,i} + T - \tau_i) \wedge \mathbf{r}(\mathbf{k}) = \mathbf{a}}{\text{Actif}^k(t, \nu_{k,i}, \tau_i)}$$

Pour pouvoir simuler le protocole WIFI, nous devons imposer à notre produit synchronisé de faire transiter les automates dès qu'ils sont actifs, dans la mesure où leur gardes sur leur horloges sont vérifiées. On définit ainsi une fonction de transition :

Définition 6.2.7 (Fonction de transition). La fonction de transmission du k -ième automate à la date τ_{i+1} du produit synchronisé est défini par la fonction suivante :

$$\begin{aligned} & \text{Trans}_{\tau_{i+1}}^k(S_{k,i}, \nu_{k,i}, S_{k,i+1}, \nu_{k,i+1}) \\ &= \begin{cases} \exists t \in E_k \ t = (S_{k,i}, l, a, \lambda, \delta, b) \ \text{Actif}^k(t, \nu_{k,i}, \tau_{i+1}) \\ \wedge \forall \tau' < \tau_{i+1} \ \forall t = (S_{k,i}, l', a', \lambda', \delta', b') \ \neg \text{Actif}^k(t, \nu_{k,i}, \rho_{k,i}, \tau') \\ \wedge S_{k,i+1} = l \wedge \nu_{k,i+1} = [\lambda \rightarrow 0](\nu_{k,i} + \tau_{i+1} - \tau_i) \wedge \mathbf{w}(\mathbf{k}, \mathbf{b}) \end{cases} \end{aligned}$$

Maintenant que nous avons défini la notion d'état actif et la fonction de transition pour un automate du produit, on peut poser la sémantique du produit synchronisé d'automates temporisés via une ressource de synchronisation :

$$\begin{array}{c} \exists \tau_{i+1} > \tau_i \ \exists k \quad \text{Trans}_{\tau_{i+1}}^k(S_{k,i}, \nu_{k,i}, S_{k,i+1}, \nu_{k,i+1}) \\ \forall k' \left\{ \begin{array}{l} \text{Trans}_{\tau_{i+1}}^{k'}(S_{k',i}, \nu_{k',i}, S_{k',i+1}, \nu_{k',i+1}) \\ \vee \left\{ \begin{array}{l} S_{k',i+1} = S_{k',i} \wedge \nu_{k',i+1} = \nu_{k',i} + \tau_{i+1} - \tau_i \\ \wedge \forall \tau_i < \tau' < \tau_{i+1} \ \forall t = (S_{k',i}, l', a', \lambda', \delta', b') \ \neg \text{Actif}^{k'}(t, \nu_{k',i}, \rho_{k',i}, \tau') \end{array} \right. \end{array} \right. \\ \hline \langle S_{1,i}, \nu_{1,i} \rangle \parallel \dots \parallel \langle S_{n,i}, \nu_{n,i} \rangle \xrightarrow{\tau_{i+1}} \langle S_{1,i+1}, \nu_{1,i+1} \rangle \parallel \dots \parallel \langle S_{n,i+1}, \nu_{n,i+1} \rangle \end{array}$$

On représente une exécution r du produit synchronisé par :

$$r : \langle s_{1,0}, \nu_{1,0} \rangle \parallel \dots \parallel \langle s_{n,0}, \nu_{n,0} \rangle \xrightarrow{\tau_1} \langle s_{1,1}, \nu_{1,1} \rangle \parallel \dots \parallel \langle s_{n,1}, \nu_{n,1} \rangle \xrightarrow{\tau_2} \langle s_{1,2}, \nu_{1,2} \rangle \parallel \dots \parallel \langle s_{n,2}, \nu_{n,2} \rangle \xrightarrow{\tau_3} \dots$$

Remarque 6.2.2. Deux automates du produit d'automates peuvent changer d'état à la même date. De plus ils peuvent très bien lire des caractères différents à partir de leur entrée sur la ressource de synchronisation. Le système représenté par la figure 6.2 représente bien ce cas. En effet on imagine bien que les automates 1 et 7 peuvent écrire à une même date et qu'alors les automates 2 et 6 reçoivent respectivement le message émis par 1 et le message émis par 7.

6.2.1 Comportements non déterministes.

On peut décrire un produit d'automates temporisés via une ressource de synchronisation temporisée qui admette des comportements non déterministes. Comme dans le cas des automates temporisés décrits en introduction de ce chapitre, plusieurs exécutions peuvent satisfaire les contraintes du produit d'automates décrit. En effet, à une date déterminée un automate du produit peut changer d'état en ayant le choix entre différentes transitions.

Ce phénomène peut avoir pour source les choses suivantes :

- L'absence de contrainte sur le caractère lu en entrée de la ressource de synchronisation. des contraintes de lecture sur la ressource de synchronisation compatibles et il existe des plages de temps où les contraintes d'horloges sont vérifiées.

Représentation graphique des automates et des actions de lecture/écriture sur une ressource de synchronisation

Si on représente chaque automate du produit par un graphe où les places sont les sommets du graphe et les arcs modélisent les transitions possibles entre ces places, alors les conditions de déclenchement de transitions de la table de synchronisation temporisée se représentent graphiquement comme les étiquettes des arcs reliant les places de la table temporisée synchronisable. Ces étiquettes sont de la forme $(\delta\tau), a?, \tau :=0, b!$ où :

- (δ) indique que la contrainte entre parenthèses doit être satisfaite pour que la règle représentée puisse se déclencher.
- $a?$ indique que l'on doit lire a sur le support de synchronisation pour que la transition se déclenche.
- $\tau :=0$ représente le fait que l'horloge τ est réinitialisée à zéro lorsque la transition est déclenchée.
- $b!$ marque le fait que l'automate écrit le caractère b sur le support de synchronisation.
- Le symbole « - » représente l'absence de condition d'horloge, ou de lecture de caractère, l'absence d'opération de ré-initialisation d'horloge ou bien d'écriture.

Exemple 6.2.4 (Synchronisation de deux automates). Considérons deux automates $\mathcal{A}_1, \mathcal{A}_2$ partageant la même ressource de synchronisation $S = (\Sigma, T)$. L'automate \mathcal{A}_1 lit et écrit sur l'entrée 1 de S et \mathcal{A}_2 lit et écrit sur l'entrée 2. Les deux automates sont mutuellement accessibles. On rappelle que le caractère ϵ est la valeur envoyée par la fonction de lecture $\mathbf{r}(\mathbf{k})$ quand il n'y a pas d'opération d'écriture effectuée sur les entrées de la ressource i telles que $T[i, k] = 1$.

On suppose que le support de transmission accepte l'alphabet $\Sigma = \{\mathbf{a}\}$.

Au démarrage chaque horloge est initialisée à zéro. L'automate \mathcal{A}_2 attend de lire le message \mathbf{a} sur la ressource de synchronisation. L'automate \mathcal{A}_1 transite immédiatement

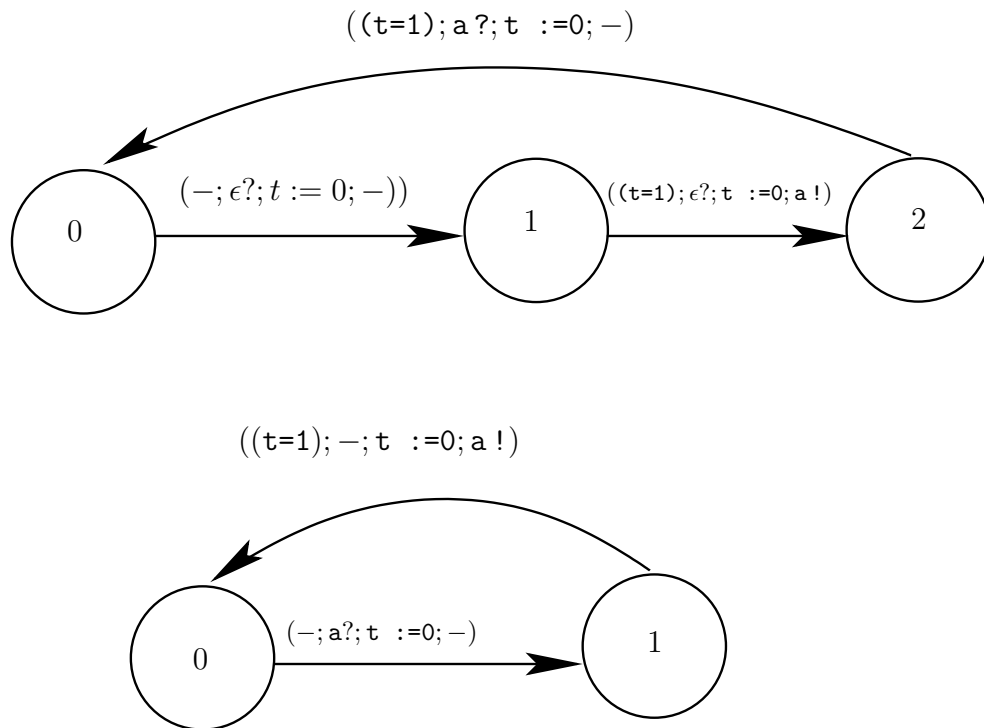


FIG. 6.4 – Synchronisation entre deux automates

de la place 0 vers la place 1 puis attend que la valeur de l'horloge t vaille 1 pour ensuite transiter de la place 1 vers la place 2. Cette transition est bien possible car la ressource de synchronisation ne contient pas de message. En effet pour qu'elle contienne un message il faudrait que l'automate \mathcal{A}_2 soit en phase d'écriture, c'est à dire en train de transiter de sa place 1 vers la place 2. Durant sa transition entre les places 1 et 2, l'automate \mathcal{A}_1 écrit le caractère a sur la ressource S , ce qui déclenche la transition de \mathcal{A}_2 entre ses places 0 et 1, puis réinitialise son horloge t à 0.

L'automate \mathcal{A}_1 doit lire le message a exactement une unité de temps après avoir transité de 1 vers 2 pour pouvoir transiter de 2 vers 0. Cela est en effet possible car l'automate \mathcal{A}_2 écrit le message a exactement une unité de temps après que sa transition entre ses places 0 et 1 a été déclenchée par l'écriture de a par \mathcal{A}_1 au moment de sa transition entre ses places 1 et 2.

Exemple 6.2.5 (Un petit exercice). On veut modéliser l'état d'une administration devant faire circuler et viser un document selon la voie hiérarchique. Le document part du bureau 0 et circule jusqu'au bureau n qui est le bureau du responsable le plus haut placé. Puis celui-ci redescend en suivant scrupuleusement le chemin inverse. On modélise le bureau 0 par un automate temporisé à deux places, une première place modélise que le document est dans le bureau et l'autre modélise le fait que le document n'est plus dans le bureau 0.

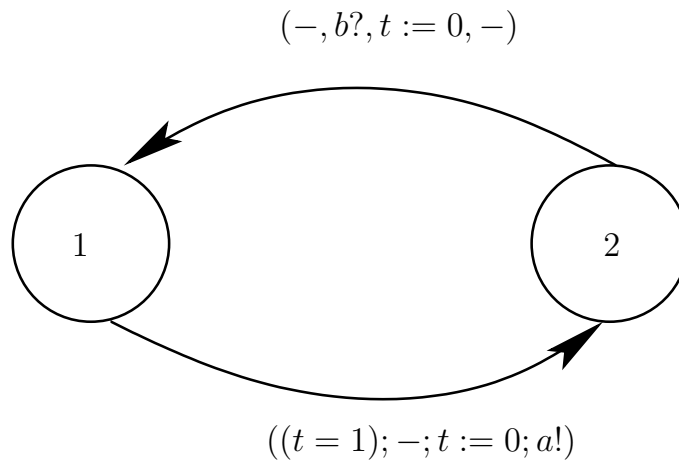


FIG. 6.5 – 0 : Le bureau du thésard qui va encore rendre ses documents en retard

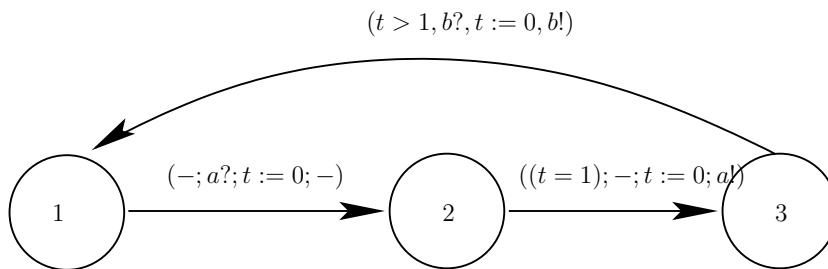


FIG. 6.6 – Les bureaux intermédiaires.

Tous les bureaux intermédiaires sont modélisés par un automate temporisé à trois places. La première place modélise la situation où le document n'est pas encore remonté jusqu'au bureau. Le second modélise le cas où le document a bien été reçu et est en cours de lecture et de vérification. La transition vers la troisième place demande une unité de temps pour modéliser le fait que la vérification du document dure une unité de temps. L'état numéro trois modélise le fait que le document a été traité et transmis au bureau suivant et qu'on attend que le document revienne. On veut également modéliser le fait que le bureau ne sera pas disponible avant une unité de temps après que le document a été transmis car la personne en charge du bureau prend systématiquement un café après avoir transmis ce genre de dossier.

Enfin, en haut de l'échelle siège le grand patron qui reçoit le dossier, lit les rapports des n bureaux précédents et vise le dossier. Cette tâche prend 2 unités de temps. On distingue deux places pour modéliser le comportement du grand patron dans le cas du traitement d'un tel dossier : il n'a pas de dossier ou bien il en traite un qu'il renvoie par la voie hiérarchique après s'en être occupé.

On appelle $\mathcal{A}_1, \dots, \mathcal{A}_n$ les automates décrits précédemment. On modélise la transmission du dossier d'un bureau à l'autre par l'envoi du message $a!$ et la réception du

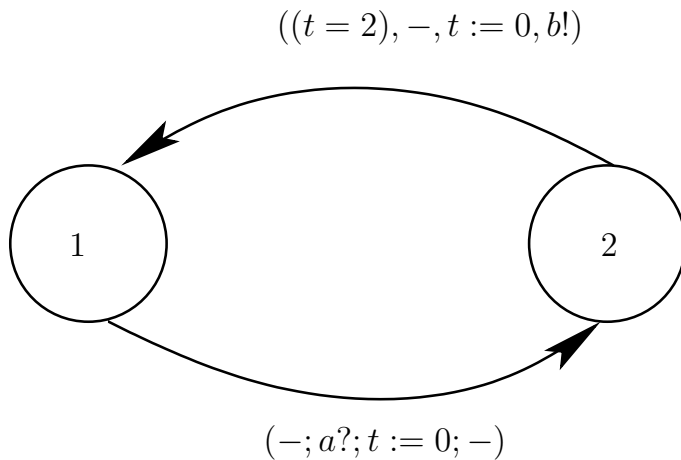


FIG. 6.7 – Le grand chef.

message $a?$ lorsque le dossier circule dans le sens ascendant. Dans le sens descendant les messages transmis sont respectivement $b!$ et $b?$. Comme décrit dans la définition du produit synchronisé d'automates temporisés que nous avons introduite dans ce même chapitre, chaque automate \mathcal{A}_k lit et écrit ses messages sur l'entrée k d'une ressource de synchronisation qui accepte l'alphabet $\Sigma = \{a, b\}$. Pour modéliser le fait qu'un dossier ne peut passer que d'un bureau vers les bureaux voisins immédiats, on fixe la table d'accessibilité de la ressource de synchronisation à la valeur suivante :

$S = (\Sigma, T)$ avec

$$T = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 1 & 0 & 1 & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & \cdots \\ \vdots & & \ddots & & 1 \\ \vdots & & & 1 & 0 \end{pmatrix}.$$

Blocage mutuel d'un produit d'automates temporisés

Il est possible de spécifier des produits synchronisés d'automates temporisés pour lesquels il existe des configurations qui peuvent provoquer des blocages mutuels communément appelés « deadlock ».

Définition 6.2.8 (Deadlock). Un produit synchronisé d'automates temporisés est dans un état de deadlock à une date $T = t$ si pour tout automate \mathcal{A}_k dans son état courant s , pour toute date $t' \geq t$, aucune des transition de la forme $\langle s, s', a, \lambda, \delta, b \rangle$ définies par la table de synchronisation temporisée ne peut s'appliquer.

Exemple 6.2.6. Le produit des deux automates temporisés \mathcal{A}_1 et \mathcal{A}_2 représentés par la figure 6.8 partageant une ressource de synchronisation $S = (\Sigma, T)$ dont le

support accepte l'alphabet $\Sigma = \{a\}$ et tels qu'ils soient mutuellement accessible est dès l'initialisation dans un état de deadlock. En effet, les deux automates attendent

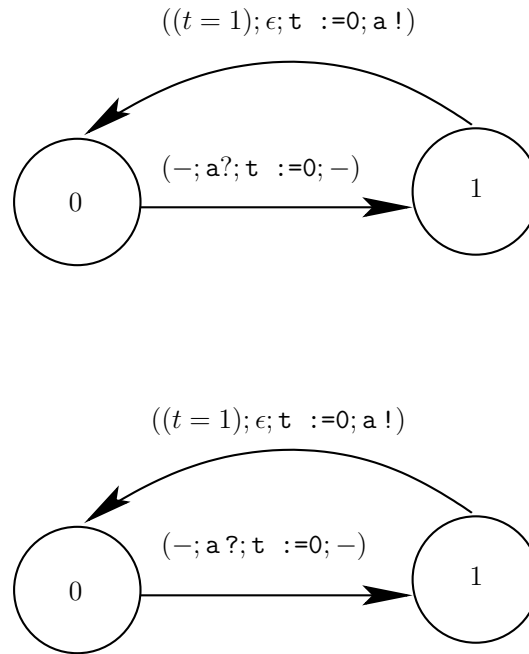


FIG. 6.8 – Un exemple de deadlock.

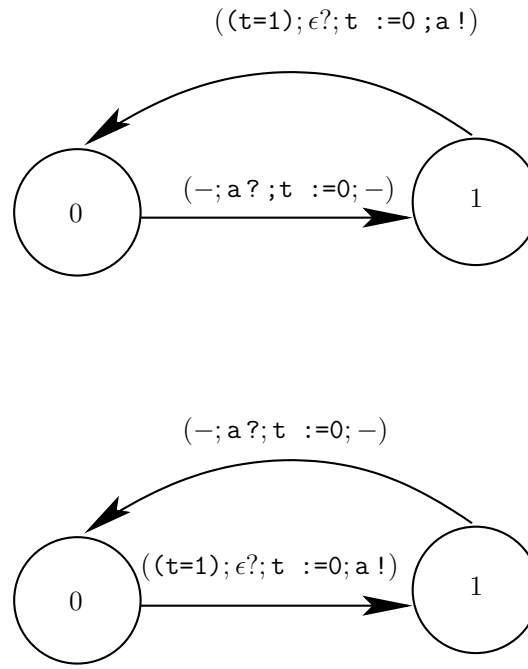
tous les deux de lire le caractère a sur la ressource de synchronisation et tous deux doivent l'avoir lu une fois pour pouvoir écrire ce même caractère sur la ressource.

Néanmoins, si on permute les étiquettes du premier arc du second automate avec celles de son second arc, il n'y a plus de blocage mutuel.

Remarque 6.2.3. Si à partir d'un certain rang tous les automates du système bouclent sur un même place, il ne s'agit pas d'un deadlock.

6.3 Produit synchronisé d'automates temporisés avec variables via une ressource de synchronisation

Étendons maintenant le modèle précédent au cas des automates temporisés avec variables. On veut simplement associer à chaque automate un ensemble fini de variables réelles que l'on veut pouvoir modifier lors des transitions entre deux places. Nous allons décrire une façon simple de faire cela et étendre la représentation graphique du comportement des automates temporisés pour spécifier comment on modifie la valeur des variables lors des transitions.



Exemple 6.2.7.

FIG. 6.9 – Synchronisation entre deux automates.

6.3.1 Automates temporisés avec variables

Nous avons défini la synchronisation des automates temporisés via une ressource de synchronisation en présentant une ressource de synchronisation temporisée et une table de transition temporisée synchronisable. Nous allons pour arriver à nos fins étendre la notion de table de transition temporisé synchronisable pour associer à des automates différentes variables, spécifier comment ces variables sont modifiées en fonction des transitions et comment les transitions peuvent être conditionnées par la valeur contenu par les variables.

Pour faire cela, nous devons associer à un automate \mathcal{A} un ensemble fini de variables x_1, \dots, x_n de type réel, définir une interprétation de variables inspirée des interprétation d'horloges et étendre les contraintes de garde sur les horloges aux valeurs des variables interprétées par une certaine interprétation.

Définition 6.3.1 (Interpretation de variables.). Une interprétation de variable ρ est une fonction qui prend en entrées n variables x_1, \dots, x_n et retourne une valeur dans \mathbb{R}^n .

Naturellement on compte pouvoir modifier la valeur de l'interprétation d'un sous ensemble de variables au cours des transitions et pour faire cela on définit l'opération suivante :

Définition 6.3.2 (Modification d'une interprétation de variables). Soit ρ une interprétation de variables pour l'ensemble de variables $x_1 \dots x_n$ telle que $\rho(x_1, \dots, x_n) =$

(a_1, \dots, a_n) . Soit $\{i_1, \dots, i_k\} \subseteq \{1, \dots, n\}$ et a'_1, \dots, a'_k un vecteur de constantes de \mathbb{R}^n , On définit par

$$[x_{i_1} := a'_{i_1}, \dots, x_{i_k} := a'_{i_k}] \rho = (b_1, \dots, b_n)$$

l'interprétation de variables telle que :

$$b_i = \begin{cases} a'_i & \text{si } i \in \{i_1, \dots, i_k\} \\ a_i & \text{sinon} \end{cases}$$

En employant la même notation, pour une opération \oplus et une constante c , on définit la nouvelle évaluation d'une des variables en fonction de l'évaluation courante des autres variables :

$$[\dots, x_i := x_j \oplus c, \dots] \rho = (b_1, \dots, b_n)$$

qui signifie que l'on modifie l'interprétation de la i -ème variable en fonction de l'interprétation par ρ de la j -ème variable :

$$b_i = \begin{cases} a_j \oplus c & \text{si } i \in \{i_1, \dots, i_k\} \\ a_i & \text{sinon} \end{cases}$$

Définition 6.3.3 (Contraintes de variables). L'ensemble $\Phi_v(X_v)$ des contraintes de variables δ_v pour un ensemble X_v de variables se définit de la manière suivante

$$\delta_v := x \leq c \mid c \leq x \mid \neg \delta_v \mid \delta_{v_1} \wedge \delta_{v_2}$$

Maintenant nous disposons de tout les outils nécessaires pour définir une table de transition temporisée avec variables :

Définition 6.3.4 (Table de transition temporisée avec variables). Une table de transition temporisée avec variables \mathcal{A} est un 7-uplet $\langle \Sigma, S, S_0, C, X_v, X_{v_0}, E \rangle$ avec :

- Σ un alphabet fini,
- S un ensemble fini de places,
- $S_0 \subseteq S$ est l'ensemble des places de départ,
- C un ensemble fini d'horloges,
- X_v un ensemble fini de variables typées,
- X_{v_0} est l'ensemble des interprétations de variables initiales possibles,
- $E \subseteq S \times S \times \mathcal{P}(\Sigma) \times 2^C \times \Phi(C) \times \Phi(X_v) \times \Sigma$ calcule l'ensemble des transitions. Un arc $\langle s, s', a, \lambda, \delta, \lambda_v, \delta_v, b \rangle$ représente une transition de la place s vers s' quand le symbole d'entrée vaut a . L'ensemble $\lambda \subset C$ représente l'ensemble des horloges à réinitialiser, δ est une contrainte d'horloge sur X_v , λ_v est une modification d'interprétation de variables et δ_v représente les contraintes que l'interprétation courante des variables doit satisfaire pour déclencher cette transitions. Le caractère b est écrit lorsque que la transition se produit.

On peut maintenant introduire ce qu'est une exécution d'un automate temporisé avec variables :

Définition 6.3.5 (Exécution d'un automate temporisé avec variables). Une exécution r notée $(\bar{r}, \bar{\nu}, \bar{\rho})$ d'une table de transition temporisée avec variables

$$\langle \Sigma, S, S_0, C, X_v, X_{v_0}, E \rangle$$

pour un mot temporisé (σ, τ) est une séquence infinie de la forme :

$$r : \langle s_0, \nu_0, \rho_0 \rangle \xrightarrow[\tau_1]{\sigma_1} \langle s_1, \nu_1, \rho_1 \rangle \xrightarrow[\tau_2]{\sigma_2} \langle s_2, \nu_2, \rho_2 \rangle \xrightarrow[\tau_3]{\sigma_3} \dots$$

avec $s_i \in S$, $\nu_i \in [C \rightarrow \mathbb{R}]$ et qui pour tout $i \geq 0$ satisfait les conditions :

- Initialisation : $s_0 \in S_0$, $\nu_0(x) = 0 \ \forall x \in C$, $\rho_0 \in X_{v_0}$,
- Consécution : Pour tout $i \geq 1$, il existe un arc de E de la forme

$$\langle s_{i-1}, s_i, \sigma_i, \lambda_i, \delta_i, \lambda_{v_i}, \delta_{v_i} \rangle$$

tel que $(\nu_{i-1} + \tau_i - \tau_{i-1})$ satisfait les contraintes d'horloge δ_i , ν_i vaut $[\lambda_i \rightarrow 0](\nu_{i-1} + \tau_i - \tau_{i-1})$ et $\rho_{i-1}(X_v)$ satisfait les contraintes de variables δ_{v_i} et ρ_i vaut $[\lambda_{v_i}] \rho_{i-1}$.

6.3.2 Produit synchronisé via une ressource de synchronisation

Les divers automates s'échangent des messages via une ressource de synchronisation, comme défini dans la section 6.2. Pour définir la sémantique de ce produit par rapport au modèle de produit synchronisé d'automate temporisés sans variable, il suffit d'ajouter la modification de l'interprétation de variables ainsi que l'expression des conditions de variables à la table de transition temporisée synchronisable.

Définition 6.3.6 (Table de transition temporisée synchronisable). Une table de transition temporisée synchronisable de taille n d'un produit synchronisé $\mathcal{A}_1 || \dots || \mathcal{A}_n$ est un ensemble de n 6-uplets $\langle S_k, S_{k,0}, X_{v_{k,0}}, C_k, X_{V_k}, E_k \rangle$ représentant chacun un automate temporisé avec variables \mathcal{A}_k avec :

- S_k est l'ensemble fini des places de A_k ,
- $S_{0,k} \subseteq S_k$ est l'ensemble des places de départ de l'automate k ,
- $X_{v_{k,0}}$ est l'ensemble des interprétations de variables initiale,
- C_k est l'ensemble fini des horloges du k -ème automate,
- X_{V_k} est un ensemble fini de variables,
- $E_k \subseteq S_k \times S_k \times \mathcal{P}(\Sigma) \times 2^{C_k} \times \Phi(C_k) \times \text{Mod}(X_{v_k}) \times \Phi(X_{v_k}) \times \Sigma$ définit l'ensemble des transitions. Chaque automate A_k lit et écrit sur l'entrée k de la ressource de synchronisation. Un arc $\langle s, s', a, \lambda, \delta, \lambda_v, \delta_v, b \rangle$ représente une transition

de la place s vers s' quand le symbole lu en entrée vaut a , c'est à dire lorsque $\mathbf{r}(\mathbf{k})=\mathbf{a}$. L'ensemble $\lambda \subset C_k$ représente l'ensemble des horloges à réinitialiser et δ est une contrainte d'horloge sur C_k . On représente par $\lambda_v \subset Mod(X_{v_k})$ une modification d'interprétation de variable et par $\delta_k \subset \Phi(X_{v_k})$ une contrainte sur les variables X_{v_k} . Cette transition peut être appliquée quand l'automate considéré se trouve dans la place s , qu'il lit le caractère a sur son entrée de la ressource de synchronisation et que les contraintes d'horloges et de variables sont satisfaites. L'automate écrit b sur la ressource de synchronisation lors de la transition, c'est à dire effectue l'opération $\mathbf{w}(\mathbf{k}, \mathbf{b})$.

À partir d'une telle table de synchronisation, on va décrire la sémantique du produit synchronisé d'automates temporisés avec variables via une ressource de synchronisation. Comme dans la section précédente, il va falloir dans un premier temps identifier sous quelles conditions un automate est actif.

Définition 6.3.7 (Automate actif). Le fait que le k -ième automate du produit synchronisé s_k soit actif pour la transition $t = (s_k, s', a, \lambda, \delta, b)$ est défini par :

$$\frac{\delta(\nu_{k,i} + T - \tau_i) \wedge \delta_v(\rho_{k,i}(X_v)) \wedge \mathbf{r}(\mathbf{k}) = \mathbf{a}}{Actif^k(t, \nu_{k,i}, \rho_{k,i}, \tau_i)}$$

Pour savoir si un automate est actif, il faut ajouter en plus des conditions vérifiées dans le cas des automates temporisés sans variables un test sur les contraintes de variables δ_v .

Définition 6.3.8 (Fonction de transition). La fonction de transmission du k -ième automate à la date τ_{i+1} du produit synchronisé est défini par la fonction suivante :

$$\begin{aligned} & Trans_{\tau_{i+1}}^k(S_{k,i}, \nu_{k,i}, \rho_{k,i}, S_{k,i+1}, \nu_{k,i+1}, \rho_{k,i+1}) \\ &= \begin{cases} \exists t \in E_k \ t = (S_{k,i}, l, a, \lambda, \delta, \lambda_v, \delta_v, b) \ Actif^k(t, \nu_{k,i}, \rho_{k,i}, \tau_{i+1}) \\ \wedge \forall \tau' < \tau_{i+1} \ \forall t = (S_{k,i}, l', a', \lambda', \delta', \lambda'_v, \delta'_v, b') \ \neg Actif^k(t, \nu_{k,i}, \rho_{k,i}, \tau') \\ \wedge \left\{ \begin{array}{l} S_{k,i+1} = l \wedge \nu_{k,i+1} = [\lambda \rightarrow 0](\nu_{k,i} + \tau_{i+1} - \tau_i) \\ \wedge \mathbf{w}(\mathbf{k}, \mathbf{b}) \wedge \rho_{k,i+1} = [\lambda_v]\rho_{k,i} \end{array} \right. \end{cases} \end{aligned}$$

Maintenant que nous avons défini la notion d'état actif et la fonction de transition pour un automate du produit, on peut poser la sémantique du produit synchronisé d'automates temporisés via une ressource de synchronisation :

$$\begin{aligned} & \exists \tau_{i+1} > \tau_i \ \exists k \quad Trans_{\tau_{i+1}}^k(S_{k,i}, \nu_{k,i}, S_{k,i+1}, \nu_{k,i+1}) \\ & \forall k' \left\{ \begin{array}{l} Trans_{\tau_{i+1}}^{k'}(S_{k',i}, \nu_{k',i}, S_{k',i+1}, \nu_{k',i+1}) \\ \vee \left\{ \begin{array}{l} S_{k',i+1} = S_{k',i} \wedge \nu_{k',i+1} = \nu_{k',i} + \tau_{i+1} - \tau_i \\ \wedge \forall \tau_i < \tau' < \tau_{i+1} \ \forall t \in E_k \ t = (S_{k',i}, l, a, \lambda, \delta, \lambda_v, \delta_v, b) \\ \neg Actif^{k'}(t, \nu_{k',i}, \rho_{k',i}, \tau') \end{array} \right. \end{array} \right. \\ & \hline & \langle S_{1,i}, \nu_{1,i}, \rho_{1,i} \rangle \parallel \dots \parallel \langle S_{n,i}, \nu_{n,i}, \rho_{n,i} \rangle \xrightarrow{\tau_{i+1}} \langle S_{1,i+1}, \nu_{1,i+1}, \rho_{1,i+1} \rangle \parallel \dots \parallel \langle S_{n,i+1}, \nu_{n,i+1}, \rho_{n,i+1} \rangle \end{aligned}$$

6.3.3 Représentation graphique

On étend juste la représentation graphique présentée dans le cas du produit d'automates temporisés sans variables en ajoutant aux étiquettes des arcs du graphe représentant les transitions des automates, les contraintes de variables et les modification d'interprétation d'horloges. On ne représente pas les valeurs des variables des automates dans la représentation graphique, seules y figurent les places représentées par des sommets étiquetés et les transitions entre les places représentés sous la forme d'arc étiquetés. Une étiquette d'un arc est de la forme :

$$((\delta), a?, \lambda, \delta, [\delta_v; \lambda_v], b !)$$

Exemple 6.3.1. On considère un automate avec trois places et disposant de deux variables entières $x : \text{int}, y : \text{int}$. Pour transiter de la place 1 à la place 2 l'automate doit lire le caractère a sur son entrée de la ressource de synchronisation quand l'interprétation de son horloge t vaut 1 et que l'interprétation des variables x et y satisfait les contraintes $x=1$ et $y \leq 3$. La nouvelle interprétation des variables satisfait $x :=5$ et $y :=y+1$ car x vaut 1 au moment de la transition.

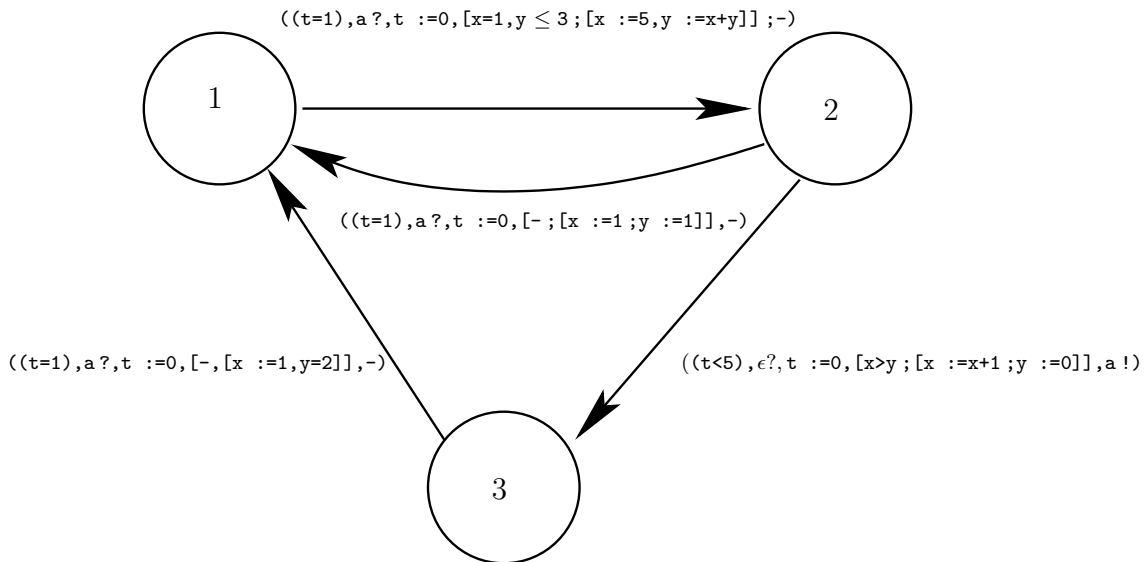


FIG. 6.10 – Un automate temporisé synchronisable avec variables.

7

Terminaison en temps moyen fini du protocole 802.11b CSMA/CA

Dans ce chapitre nous allons mettre en application les méthodes de preuve introduites dans cette thèse pour montrer qu'un protocole réseau probabiliste, couramment utilisé, parvient à effectuer son travail au bout d'un temps moyen fini. En effet, nous allons montrer qu'à partir d'une configuration initiale bien déterminée, l'algorithme du CSMA-CA 802.11b [con] termine son travail au bout d'un temps dont la moyenne est finie. On trouve de nombreuses variantes de ce protocole ainsi que de nombreuses documentations, mais nous avons utilisé les spécifications du protocole au niveau physique présentées par l'organisme de normalisation IEEE dans [IEE03].

Nous allons organiser l'exposé de ce résultat de la façon suivante : Après avoir introduit le principe de fonctionnement du protocole réseau CSMA-CA 802.11b, nous allons expliquer de quelle manière nous allons pouvoir simuler un réseau de stations communicant grâce à ce protocole, dans le cas où celui ci utilise une borne centrale pour synchroniser et retransmettre les messages. Une fois que nous aurons décrit le principe de l'algorithme de simulation, nous présenterons de quelle manière on peut le coder grâce à un système de règles de réécriture probabiliste. Enfin nous calculerons une fonction d'évaluation des termes codant le simulateur, qui certifie que celui-ci permet d'atteindre une configuration terminale au bout d'un temps moyen fini à partir d'un ensemble d'états initiaux bien déterminés.

L'objet principal de ce chapitre est donner un exemple de l'expressivité des systèmes de réécriture probabilistes sous stratégies pour exprimer certains formalismes de haut niveau tels que les automates temporisés probabilistes et les processus de décision Markoviens. Cette étude de cas voue à valider l'expressivité des systèmes de réécriture probabilistes ainsi qu'à montrer que les méthodes de preuves décrites au cours des chapitres précédents peuvent servir à vérifier si des systèmes complexes satisfont des propriétés de terminaison en temps moyen fini.

En ce qui concerne le résultat de l'étude lui même, nous allons le rapprocher avec

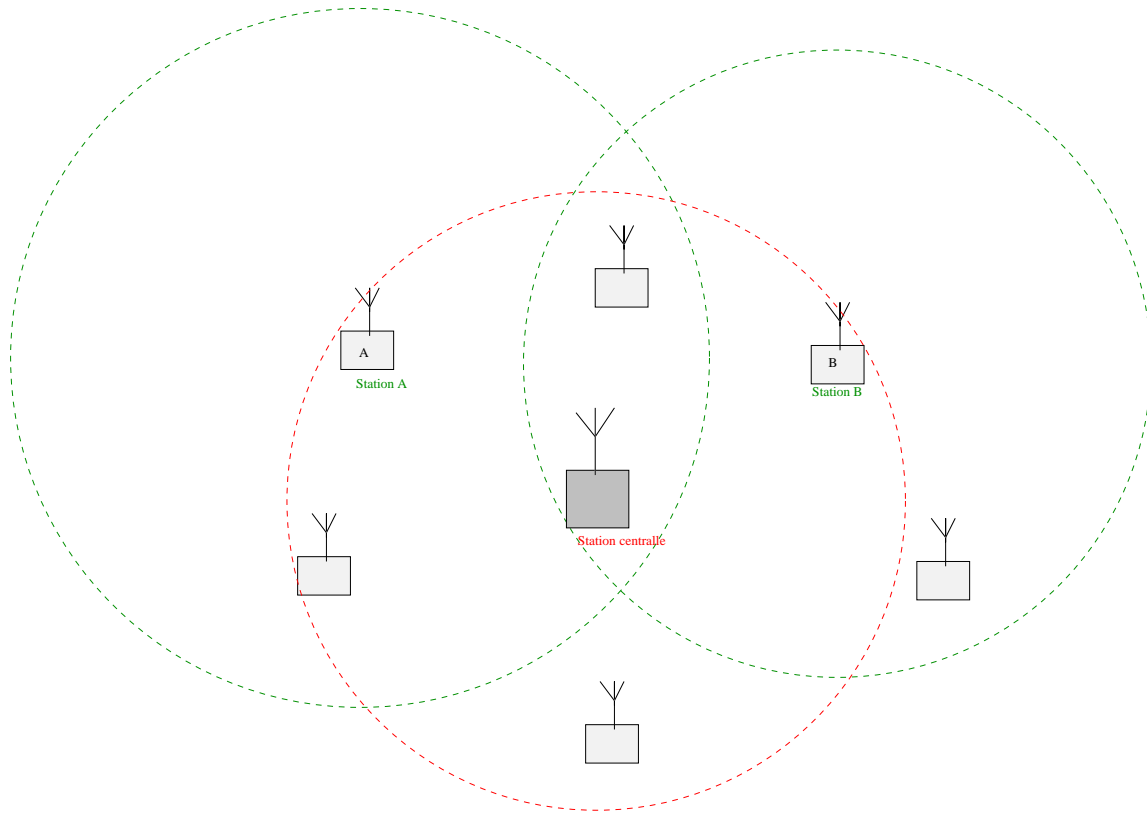


FIG. 7.1 – Représentation d'un réseau de stations avec borne centrale

les résultats venant de la communauté « model-checking » en général et de la communauté « *PRISM* » en particulier, puis nous aurons une discussion sur les avantages et les limitations de chacune des deux approches.

7.1 Description du protocole CSMA/CA

Le protocole CSMA/CA est le protocole réseau utilisé dans les réseaux informatiques sans fils communément appelés « réseaux WIFI ». Ce protocole permet de réaliser plusieurs tâches, la première étant de partager une unique bande passante radio afin permettre à chaque participant d'accéder à un droit d'émission. La seconde est de détecter et de corriger les erreurs de transmission et éventuellement de réclamer la retransmission des données qui sont trop corrompues pour pouvoir être corrigées. De même le protocole permet de gérer les collisions.

On dit qu'une collision se produit lorsque deux émetteurs écrivent simultanément un message sur la même ressource critique, c'est-à-dire le canal radio dans le cas du WIFI.

Comme le fait de communiquer par radio empêche toute station en phase d'émis-

sion de vérifier qu'une autre station émet en même temps, le protocole fournit un mécanisme de détection des erreurs. De plus, pour éviter toute collision récurrente, un mécanisme de désynchronisation des émissions permet de réduire la probabilité qu'au moins deux stations réclament le droit de réserver la bande passante ou émettent des données en même temps. Ce mécanisme utilise un algorithme probabiliste connu sous le nom d'algorithme de **backoff** qui permet de tirer aléatoirement des temps d'attente de plus en plus grand au fur et à mesure que le nombre de collisions enregistrées depuis la dernière transmission réussie augmente.

Quand il est utilisé avec une borne centrale, le protocole du CSMA/CA s'exécute du point de vue d'une station suivant les phases principales suivantes :

1. Attente que le canal radio soit libre de toute émission.
2. Attente d'un temps constant « DIFS » plus un temps aléatoire **backoff**(**cback**) dépendant du nombre de collisions enregistrées depuis le dernier envoi avec succès **cback**. Durant cette phase, si aucun signal n'a été détecté alors on passe à l'étape suivante, sinon on retourne à l'étape précédente après avoir attendu un temps. NAV décrit par le protocole si le signal lu est un CTS . Si le signal détecté n'est ne correspond pas à un CTS alors on reviens à la première phase de l'algorithme sans attendre.
3. Émission d'une requête d'émission « RTS ».
4. Attente de la réception de la confirmation de la requête, « CTS », envoyée par la station centrale. Si le temps d'attente dépasse une certaine constante, on retourne à la première étape.
5. Envoi d'une trame de données.
6. Attente de l'acquiescement de réception « ACK », si la trame « ACK » témoigne que la communication s'est bien passée alors la transmission est validée, sinon on compte une collision supplémentaire et on retourne au début du processus.
7. Fin de transmission.

La figure 7.3 représente le diagramme temporel de l'envoi avec succès d'un paquet de données avec le protocole CSMA/CA.

Remarque 7.1.1. Le temps d'attente NAV permet à la station de retourner dans la première phase du protocole en même temps que la station à qui la trame CTS était adressée. Si ce mécanisme n'existait pas, deux stations cachées entreraient trop souvent en collision dans le cas où le temps d'envoi des messages est sensiblement plus long que le temps DIFS spécifié dans le protocole, additionnée aux valeurs tirées par l'algorithme du **backoff**.

Néanmoins, il est possible que deux stations éloignées ne puissent pas s'entendre, et que l'une émette une requête une trame RTS alors que autre station l'a déjà fait.

Dans ce cas, il est possible que cette trame puisse parasiter d'autres signaux tels qu'une trame CTS émise par la borne centrale.

Chaque station comptabilise les collisions qu'elle a détecté depuis la dernière trame émise ou depuis le début de l'exécution de l'algorithme si il n'y a pas encore eu de trame émise. On comptabilise une collision supplémentaire lorsqu'en phase 2 on détecte une trame « RTS » émise par une autre station et lorsque la réception de la trame « CTS » n'intervient pas dans la phase 4. On comptabilise également une collision quand la trame d'acquittement permet de détecter une erreur de transmission ou que celle-ci n'a pas été reçue.

On dit qu'une station est cachée par rapport à une seconde lorsque cette première ne peut pas capter les signaux de la seconde.

Exemple 7.1.1. Dans la figure 7.1 la portée des signaux des stations A, B et centrale sont représentées par des cercles en pointillés. Les stations A et B sont cachées l'une par rapport à l'autre.

Nous résumons dans le tableau de la figure 7.2 l'ensemble des acronymes et abréviations propres aux protocoles WIFI 802.11b.

Nous devons maintenant expliquer à quel niveau interviennent les phénomènes probabilistes et à quelles fins ils sont destinés.

L'algorithme de « *l'exponential backoff* »

Toute station qui a un message à transmettre doit dans un premier temps attendre que le canal radio devienne libre de tout message. Puis, pour éviter que toutes les stations en phase d'attente ne commencent à émettre simultanément, chaque station va calculer de manière aléatoire et indépendante un temps d'attente qu'elle ajoutera à un temps d'attente constant DIFS spécifié par le protocole.

Ce temps d'attente aléatoire est calculé grâce à l'algorithme de l'« *exponential backoff* », décrit ci-dessous :

Algorithme: backoff (nb_collisions)

Variables d'entrée:

entier: nb_collision

Renvoie:

entier.

```
backoff(int nb_collision){
    return(random(1,pow(2,max(nb_collision,10))));
}
```

DIFS	Temps constant fixé par le protocole.
RTS(k)	Trame « Request To Send », émise par une station pour demander l'exclusivité de la bande passante à la station centrale. Elle contient l'identifiant de la station qui l'émet, ici représenté par l'entier k .
CTS(k)	Trame « Clear To Send », émise par la station centrale, contient l'identifiant de la station destinataire, ici représenté par l'entier k .
MSG	Trame contenant le message à transmettre.
ACK	Trame « Acknowledgment », trame émise par la station centrale, contient un code de contrôle d'erreurs. Permet de certifier que les données de MSG ont bien été reçues et correctement transmises.
backoff	Paramètre de l'algorithme du backoff exponentiel
SIGNAL	Un des messages suivant RTS, CTS, MSG, ACK
TCTS	Temps d'émission d'une trame CTS
TRTS	Temps d'émission d'une trame RTS
TACK	Temps d'émission d'une trame ACK
MSG_TIME	Temps d'émission d'un message
NAV	Temps d'attente calculé à partir des informations contenues dans les trames CTS

FIG. 7.2 – Table des symboles du CSMA/CA

fonction: random (min,max)

Variables d'entrée:

entier: min, entier: max

description:

random renvoie un entier compris entre min et max suivant une loi uniforme.

fonction: pow (a,b)

Variables d'entrée: entier:a,b

Renvoie:

entier.

description:

pow renvoie la valeur de a élevée à la puissance b.

L'algorithme de l'« *l'exponential backoff* » retourne simplement une valeur tirée uniformément sur l'ensemble des entiers $\{1, \dots, 2^{\max(\text{nb_collision}, 10)}\}$. On remarque qu'à chaque fois qu'une station comptabilise une collision supplémentaire, elle aura une valeur tirée par son algorithme de **backoff** qui sera deux fois plus élevée en moyenne, et cela tant que son compteur de collision est inférieur à dix.

Remarque 7.1.2. Ce sont ces tirages de temps d'attente qui d'une part permettent de briser la symétrie de l'algorithme et qui d'autre part vont permettre au protocole de réduire le nombre de collisions. Toutes les autres opérations du protocole se déroulent de manière déterministe et le comportement de chaque station, y compris celui de la borne centrale, va pouvoir se modéliser avec le modèle de produit synchronisé d'automates temporisés avec variables décrit dans la section 6.3. Néanmoins, on peut calculer la probabilité de certains phénomènes en connaissant la valeur des tirages des variables aléatoires **backoff** de chaque station, valeur qui dépend du nombre de collisions apparues depuis la dernière transmission avec succès.

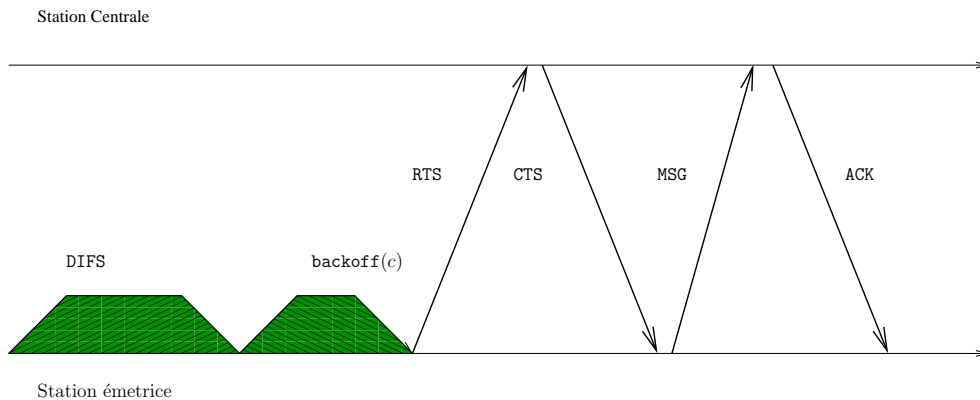


FIG. 7.3 – Phases d'un envoi d'un paquet de données par le protocole CSMA-CA en fonction du temps.

7.2 Modélisation du système sous la forme d'un produit d'automates temporisés

Nous présentons de quelle manière nous modélisons un réseau de stations WIFI. On suppose que réseau est constitué de $n + 1$ stations $\{\mathcal{A}_1, \dots, \mathcal{A}_{n+1}\}$ dont la station

centrale. Nous utilisons le modèle de produit synchronisé d'automates temporisés défini dans la section 6.2. Le canal radio est représenté sous la forme d'une ressource de synchronisation $S = (\Sigma, T)$ où le support de transmission accepte les mots suivants $\{\text{RTS}(1), \dots, \text{RTS}(n), \text{CTS}(1), \dots, \text{CTS}(n), \text{MSG}, \text{ACK}\}$. Chaque station \mathcal{A}_i lit et écrit sur la ressource de synchronisation à partir de l'entrée numéro i . La table d'accessibilité T permet de coder quelles sont les stations qui peuvent recevoir un message émis par une autre station. Toutes les stations reçoivent les messages émis à partir de la station centrale et la station centrale est accessible à partir de toutes les stations.

Comportement de la station centrale

Le comportement de la station centrale en fonction de l'état du canal radio est modélisé par la figure 7.4. On distingue quatre états pour la station centrale,

- Etat 0 dans lequel la station attend de recevoir une trame **RTS**.
- Etat 1 : une trame **RTS** a été reçue.
- Etat 2 : La station a émis une trame **CTS**, attente d'une trame **MSG**.
- Etat 3 : La station a reçu un message de données **MSG**, on transite vers 0 en émettant une trame **ACK**.

On transite de l'état 0 vers l'état 1 lorsque la station détecte sur le canal radio le passage d'une trame **RTS**, c'est à dire au moment où on lit le caractère **RTS** sur l'entrée de la ressource de synchronisation.

La station transite de l'état 1 vers l'état 2 sans attendre pour modéliser l'envoi d'une trame **CTS**. La station centrale écrit le caractère **CTS** sur son entrée de la ressource de synchronisation.

On transite de 2 vers 3 si la station détecte sur la ressource de synchronisation le mot **MSG** et que le temps d'attente de ce message a été inférieur à **Timeout**. Cependant on transite de 2 vers 0 si rien n'a été reçu au bout de **Timeout** unités de temps ou qu'un mot différent de **MSG** a été lu durant la phase d'attente.

La transition de 3 vers 0 modélise la confirmation de la bonne réception du paquet et de son routage en écrivant le mot **ACK** sur la ressource de synchronisation.

Modélisation d'une station émettrice

L'état d'une station est modélisé par un ensemble de valeurs : Un identifiant, la phase de l'algorithme d'émission dans laquelle se trouve la station, le nombre de messages à émettre, le nombre de collisions recensées depuis la dernière émission d'un paquet de données et une horloge correspondant au temps à attendre avant de déclencher la prochaine transition.

Le déroulement de l'algorithme du CSMA-CA est représenté par 6 places différentes :

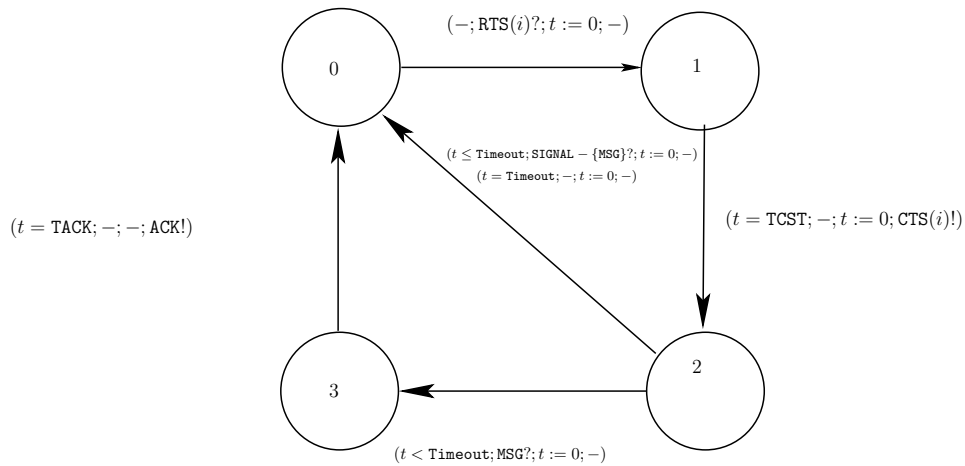


FIG. 7.4 – Comportement de la station centrale.

- La place 0 correspond à l’attente de libération du canal radio, c’est à dire à attendre que la fonction de lecture appliquée sur l’entrée de la ressource de synchronisation associée à la station considérée retourne le le mot vide ϵ . La valeur lue sur l’entrée du canal doit valoir ϵ durant DIFS unités de temps pour pouvoir transiter vers la place 1.
- La place 1 marque le début de l’attente du temps aléatoire $\text{backoff}(c)$.
- La place 2 correspond au début de la phase d’émission d’une trame RTS.
- La place 3 correspond à l’attente d’une trame CTS envoyée par la station centrale en réponse à la trame RTS.
- La place 4 marque la phase d’émission d’une trame de données.
- La place 5 correspond à la phase d’attente d’une trame d’acquiescement « ACK ».
- La place 6 non représenté ici correspond en fait à recommencer le même processus à partir de l’état 0 mais avec un nombre de messages à émettre diminué de 1 s’il restait au moins un message et avec un nombre de collisions réinitialisé à 0. S’il ne restait plus qu’un seul message à émettre alors on considère que l’on arrive sur un état particulier appelé \perp codant un émetteur inactif.

On va énumérer successivement les différents états de l’automate modélisant un émetteur et décrire sous quelles conditions on passe d’un état vers un autre et quelles sont les actions qui se produisent lors de ces transitions.

Les conditions permettant de transiter entre ces états sont représentées dans la figure 7.5.

Chaque automate possède sa propre horloge, notée \mathbf{t} . Un émetteur \mathbf{t} reste dans la place 0 tant que la fonction $\mathbf{r}(\mathbf{k})$ renvoie un mot différent du mot vide ϵ . Dès que $\mathbf{r}(\mathbf{k})$ lit le mot vide ϵ durant une période de temps égale à DIFS l’automate \mathbf{k} transite vers la place 1.

En entrant dans la place 1, l’automate calcule au bout de combien de temps

devra se déclencher la transition vers la place 2 en utilisant l'algorithme du **backoff**. Si durant cette période d'attente l'automate lit un mot autre que ϵ sur la ressource de synchronisation alors il transite immédiatement vers l'état 0 si le message détecté n'est pas $\text{CTS}(k)$ et au bout de **NAV** unités de temps si la trame détectée est une trame $\text{CTS}(i)$ adressée à la station i avec $i \neq k$. Dans ces deux cas de figure on comptabilise une collision supplémentaire. Dans le cas contraire l'automate transite vers l'état 2 et écrit le caractère $\text{RTS}(k)$ sur la ressource de synchronisation au moment où la transition a lieu.

La place 2 marque le début de la phase d'attente d'une trame **CTS**. Dans le cas où l'automate k est dans la place 2, il attend la trame que $r(k)=\text{CTS}(k)$. Cette phase d'attente dure au plus **Timeout** unités de temps. Si avant cette échéance l'automate lit sur la ressource de synchronisation un autre mot que $\text{CTS}(k)$ alors l'automate transite vers la place 0. Il transite également vers la place 0 si la trame **CTS** comportant son identifiant n'est pas reçue au bout de **Timeout** unités de temps. Dans ces deux cas, une collision supplémentaire est comptabilisée. Si le message **CTS** est bien reçu avant **Timeout** unités de temps alors l'automate transite vers la place 3.

La place 3 marque le début de l'émission d'une trame de données **MSG** dont le temps d'émission est fixé par le protocole et est représenté par la constante **MSG_TIME**.

La place 4 marque la phase d'attente de la réception d'une trame d'acquiescement **ACK** qui dure au maximum **Timeout** unités de temps. Si cette échéance est atteinte sans que l'automate ait reçu la trame d'acquiescement alors l'automate transite vers l'état 0. Si l'automate lit bien la trame **ACK** alors il transite vers l'état 5.

La place 5 marque la phase de vérification des données transmises avec l'algorithme de checksum. Si le contrôle des données certifie que la transmission s'est déroulée correctement et que le nombre de message restant à envoyer était supérieur à 1 alors on transite vers l'état 0 en réinitialisant le compteur de collisions à 0 et en décrémentant le nombre de messages restant à émettre. Si jamais une erreur a été détectée alors on transite vers l'état 0 tout en incrémentant le nombre de collisions.

7.3 Codage du produit synchronisé d'automates temporisés sous la forme d'un système de réécriture probabiliste sous stratégie.

Nous allons dans cette partie décrire comment le produit des automates correspondant aux stations du réseau est simulé et comment on peut le coder grâce à un système de réécriture probabiliste. Cette étape permettra d'appliquer les méthodes de preuve de terminaison presque sûre positive sous stratégies décrites dans le chapitre 5.

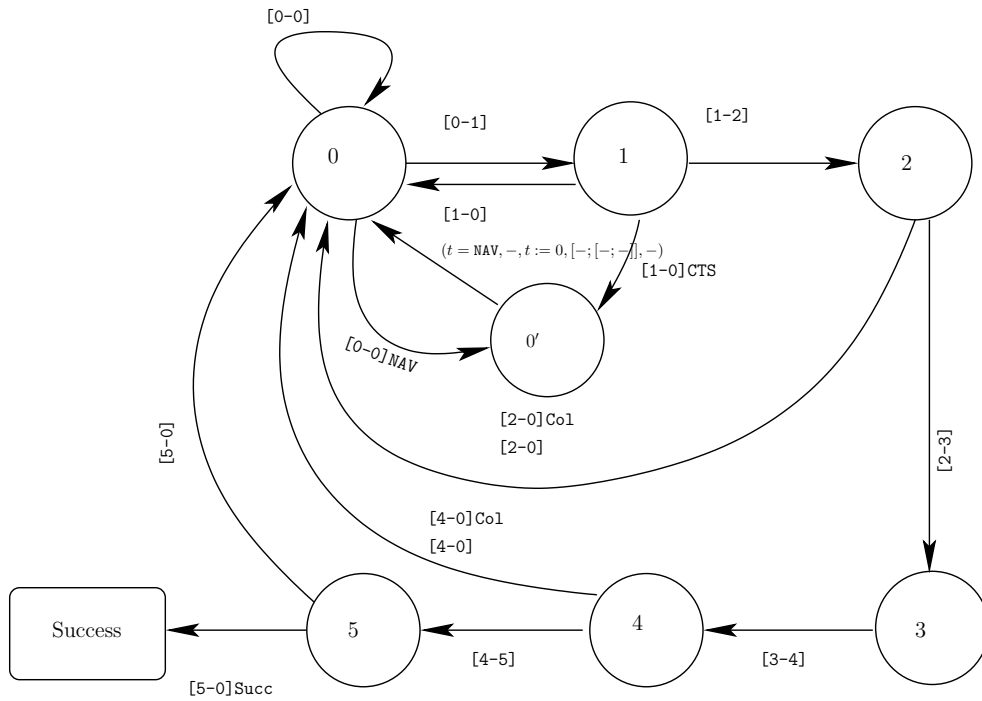


FIG. 7.5 – Vue synthétique des transitions entre les étapes principales de l’algo CSMA-CA.

[0-0]	$((t < \text{DIFS}), \text{SIGNAL} - \text{CTS}?, t := 0, [-; [-]], -)$
[0-0']NAV	$((t \leq \text{DIFS}), \text{CTS}?, t := 0, -, [-; [-; -]], -)$
[0'-0]	$((t = \text{NAV}), -, t := 0, -, [-; [-; -]], -)$
[0-1]	$((t = \text{DIFS}); \epsilon?; t := 0; [\text{nbmess} \geq 0; [-]]; -)$
[1-0]	$((t < \text{backoff}), \text{SIGNAL}?, t := 0; [-; [\text{nbcol} := \text{nbcol} + 1]]; -)$
[1-0]CTS	$((t < \text{backoff}), \text{CTS}?, t := 0, [-; [-, -]], -)$
[1-2]	$((t = \text{backoff} + \text{TRTS}); -, t := 0; [-; [-]]; \text{RTS}(i)!$
[2-0]	$((t = \text{Timeout}); -, t := 0; [-; [\text{nbcol} := \text{nbcol} + 1]]; -)$
[2-0]Co1	$((t < \text{Timeout}); \text{SIGNAL} - \text{CTS}(i)?; t := 0; [\text{nbcol} := \text{nbcol} + 1]]; -)$
[2-3]	$((t < \text{Timeout}); \text{CTS}(i)?; t := 0; [-; [-]]; -)$
[3-4]	$((t = \text{MSG_TIME}); -, t := 0; [-; [-]]; \text{MSG}!$
[4-0]	$((t = \text{Timeout}); -, t := 0; [-; [\text{nbcol} := \text{nbcol} + 1]]; -)$
[4-0]Co1	$((t < \text{Timeout}); \text{SIGNAL} - \text{ACK}(i)?; t := 0; [-; [\text{nbcol} := \text{nbcol} + 1]]; -)$
[4-5]	$((t < \text{Timeout}); \text{ACK}(i)?; t := 0; [-; [-]]; -)$
[5-0]	$(-, \text{Checksum_bad}?; t := 0; [-; [\text{nbcol} := \text{nbcol} + 1]]; -)$
[5-0succ]	$(-, \text{Checksum_ok}?; t := 0; [-; [\text{nbmess} := \text{nbmess} - 1; \text{nbcol} := 0]]; -)$

FIG. 7.6 – Opérations effectuées lors des transitions de la figure 7.5.

7.3.1 Algorithme de simulation du produit d'automates.

Simuler le comportement du réseau nécessite d'être capable d'effectuer les opérations suivantes :

- Simuler le temps et avoir une horloge globale.
- Simuler les horloges propres à chaque automate.
- Simuler le canal radio par une ressource de synchronisation et calculer la valeur lue sur chaque entrée de la ressource de synchronisation.
- Ordonnancer les actions de chaque automate en fonction des contraintes d'horloge et de la valeur lue sur chaque entrée de la ressource de synchronisation.
- Simuler le comportement de chaque automate comme décrit dans la section précédente en prenant soin de mettre à jour le nombre de trames de données à émettre ainsi que le nombre de collisions enregistrées depuis la dernière émission réussie.

Nous allons maintenant expliquer comment nous allons pouvoir coder avec un système de réécriture simple un simulateur se comportant comme un réseau de stations WIFI communicant grâce au protocole 802.11b. Pour cela nous devons commencer par décrire l'algèbre de termes $T(\Sigma, X)$ que nous allons utiliser pour représenter les objets du système. Par la suite, nous ferons évoluer ce système en respectant la sémantique du produit synchronisé d'automates temporisés avec variables, défini dans la section 6.3. Nous ferons cela en réécrivant les sous-termes correspondants aux parties du système qui changent lors des transitions du système global. On prend à titre d'exemple les états courants des automates du produit devant transiter, les interprétations d'horloges ainsi que la variable du temps global.

Représentation des objets modélisant le système.

On va donc construire une algèbre de termes $T(\Sigma, X)$ dont chaque élément représentera soit un automate temporisé modélisant une station, soit une variable d'un de ces automates ou encore le produit de tous les automates représentant l'ensemble du réseau. Par la suite nous décrirons des règles de réécritures qui mettront ces termes en relation et qui correspondront aux phases d'évolution du système.

On représente une station émettrice par un 7-uplet (`id`, `eta`, `dest`, `tatt`, `nbmess`, `cback`, `write`) avec

1. `id` est le numéro de l'automate,
2. `eta` est le numéro de la place courante avec $\text{eta} \in \{0, 1, \dots, 5\}$,
3. `dest` est le numéro de la place vers laquelle il est prévu de transiter une fois les contraintes d'horloges satisfaites,
4. `tatt` temps restant avant la prochaine transition. Ce champ représente la contrainte d'horloge $\mathbf{t} = \mathbf{T} + \mathbf{tatt}$ avec \mathbf{t} horloge locale propre à cet automate,
5. `nbmess` est le nombre de trames restant à émettre,

6. `cback` est le nombre de collisions depuis la dernière émission,
7. `write` est la variable entière codant le caractère à envoyer, la valeur 0 marque l'absence d'écriture.

On représente également la station centrale par un 6-uplet (`id,eta,tatt,nbmess,cback,write`) avec

1. `central_id` est le numéro de l'automate représentant la station centrale,
2. `eta` est le numéro de l'état courant avec $eta \in \{0, 1, \dots, 3\}$ de l'algorithme de la station centrale,
3. `tatt` est le temps restant avant la prochaine transition, représente la contrainte d'horloge $t = T + tatt$ avec t horloge locale propre à cet automate,
4. `nbmess` n'est pas utilisé par la station centrale,
5. `cback` champ également inutilisé par la station centrale,
6. `write` est la variable entière codant le caractère à envoyer, la valeur 0 marque l'absence d'écriture.

L'ensemble du système est représenté par le triplet (t_g, l, T) où t_g est un entier codant la date courante, et l est la liste des 7-uplets codant les émetteurs et la station centrale. T est le codage de la table d'accessibilité de la ressource de synchronisation correspondant au fait que deux stations peuvent s'entendre ou pas.

On précise qu'on représente une liste vide par `nil` et qu'on note la concaténation de liste par un point. Si l est une liste de termes codant une station émettrice et a un terme codant une station émettrice alors $a.l$ est la liste de termes l à laquelle on a rajouté le terme a en tête. De même $a.nil$ est la liste contenant le seul terme a .

Pour calculer la valeur lue sur chaque entrée de la ressource de synchronisation, on dispose d'un tableau `TMess` contenant autant d'entrées qu'il y a de stations et dont l'entrée i représente la valeur lue à l'entrée i de la ressource de synchronisation. `TMess` est un tableau de liste de caractère, correspondant à l'union des caractères écrits sur les entrées à partir desquelles l'entrée de lecture est accessible.

Algorithme de simulation du système.

Pour fixer le cadre de notre modèle, nous précisons quels sont les réseaux WIFI que nous modélisons et nous émettons sans perte de généralité les hypothèses suivantes qui vont nous permettre d'appliquer nos preuves de terminaison :

- La station 1 entend $n - 1$ autres stations, nommée $2, \dots, n$.
- Il y a k stations cachées pour la station 1, qu'on numérote $n + 1, \dots, n + k$.
- Toute station peut entendre la station centrale et réciproquement.
- La topologie du réseau ne change pas durant l'exécution du protocole.
- Chaque station à un nombre fixe de messages à émettre.

Pour simuler l'évolution de l'ensemble du système on exécute le produit synchronisé des différents automates temporisés communiquant via la ressource de synchronisation qui représente en terme d'accessibilité la topologie du réseau simulé.

Pour réaliser cela on accomplit les choses suivantes :

- Il faut définir une ressource de synchronisation représentant la topologie du réseau ainsi que l'alphabet des messages échangés par les diverses stations. Les valeurs lues sur les entrées de la ressource suivront les règles définies sur la figure 6.3 de la section 6.2.
- Il faut réserver une variable t_g représentant le temps et contenant la date globale.
- Il faut avoir à disposition un algorithme d'ordonnancement des divers automates temporisés qui permet d'exécuter les transitions de ces automates en fonction de la valeur de leur horloges, de la valuation des contraintes d'horloges en fonction du temps et de ce que retournent les fonctions de lecture des entrées de la ressource de synchronisation.

Pour simuler le comportement du système en fonction du temps, il suffit d'incrémenter le temps jusqu'à la date de la prochaine action. Pour faire cela on définit la relation suivante :

Définition 7.3.1 (Relation $>_{simul}$). La relation $>_{simul}$ est vérifiée si les automates temporisés représentant les stations émettrices satisfont les conditions suivantes :

- $\mathcal{A}_1 >_{simul} \mathcal{A}_2$ si
 - \mathcal{A}_1 et \mathcal{A}_2 ont tous les deux des messages à envoyer et la date de prochaine transition de \mathcal{A}_1 est inférieure ou égale à celle de \mathcal{A}_2
 - ou si \mathcal{A}_1 a des messages à émettre alors que \mathcal{A}_2 n'en a pas.
- Si \mathcal{A}_1 et \mathcal{A}_2 n'ont plus de message à envoyer alors $>_{simul}$ ne compare pas \mathcal{A}_1 et \mathcal{A}_2 .

Remarque 7.3.1. La relation $>_{simul}$ est une relation d'ordre partiel sur les automates temporisés. On appellera $>_{simul}$ l'ordre de simulation.

Pour représenter le système à une date donnée, on place chaque automate représentant une station dans une file d'attente triée selon l'ordre partiel $>_{simul}$. Une fois la liste des automates triée, il suffit de vérifier que le premier élément de la liste (`id,eta,tatt,nbmess,cback,write`) représente un automate qui possède des messages à envoyer. Pour vérifier cela, il suffit de faire le test suivant `nbmess==0`. L'ordre $>_{simul}$ n'est pas un ordre strict, car deux éléments s_1 et s_2 ayant au moins un élément à envoyer et un champ `tatt` contenant tous deux la même valeur ne vérifient ni $s_1 >_{simul} s_2$ ni $s_2 >_{simul} s_1$. À cause de cela, il existe des configurations du réseau qui peuvent être représentées de plusieurs manières. Ainsi s'il existe p stations ayant toutes au moins un message à émettre et telles que ces p stations ont la même valeur de champ `tatt`, alors il existe au moins $p!$ façons de représenter un tel système avec notre modèle.

Si cet automate ne possède aucun message à envoyer alors toutes les autres stations représentées par les automates de la liste n'ont plus de messages à envoyer. Cette propriété sert de critère de terminaison car la simulation est terminée quand toutes les stations représentées sont dans un état où elles n'ont plus de message à envoyer.

Dans le cas où le premier automate de la liste code une station pour laquelle il reste au moins un message à envoyer, il faut effectuer les opérations suivantes :

- Incrémenter la date globale t_g de `tatt`.
- Retrancher au champ `tatt` de chaque automate la valeur `tatt` du premier automate pour signifier que la prochaine action prévue se déclenchera `tatt` unités plus tôt une fois la date globale mise à jour.
- Faire transiter l'état du premier automate en fonction du caractère lu sur la ressource de synchronisation et des messages enregistrés depuis la dernière action.
- Calculer les valeurs lues sur les entrées de la ressource de synchronisation.
- Dans le cas où il l'automate écrit sur la ressource de synchronisation. il faut éventuellement interrompre certaines transitions programmées.

On réitère ces opérations après avoir réinsérer la premier élément dans la liste pour respecter l'ordre $>_{simul}$ et cela jusqu'à ce qu'on arrive dans une configuration où tous les automates ont émis l'ensemble des messages qu'ils avaient initialement à envoyer.

Nous allons maintenant décrire de quelle façon nous calculons les transitions et comment nous les codons avec des règles de réécriture. Une fois cela fait, nous expliquerons comment nous implémentons le produit synchronisé de l'ensemble des automates grâce à un système de réécriture probabiliste dont les choix sont conditionnés par une stratégie que nous expliciterons.

7.3.2 Faire transiter l'automate d'un émetteur en fonction des valeurs de la ressource de synchronisation

Maintenant que nous avons fixé la syntaxe des termes permettant de représenter les automates codant une station émettrice, il reste à écrire l'ensemble des règles de réécriture probabilistes permettant de décrire les transitions entre les différentes places des automates conformément à la figure 7.5. On rappelle que l'on détermine les valeurs lues sur chacune des entrées de la ressource de synchronisation grâce à l'algorithme de simulation. C'est le codage de la stratégie qui permettra de choisir une règle parmi plusieurs quand le choix existera. Cela permettra de sélectionner la règle de transition dans les cas où on détecte une collision et dans le cas où en on détecte pas.

Les transitions à partir de la place 0 sont codées par les règles de réécriture

suivantes ³ :

Nom	Règle
0-0	$(id, 0, 1, tatt, nbmess, cback, 0)$ $\rightarrow \{(id, 0, 1, DIFS, nbmess, cback, 0) : 1$
0-OCTS	$(id, 0, 0, tatt, nbmess, cback, 0)$ $\rightarrow \{(id, 0, 0, NAV, nbmess, cback, 0) : 1$
0-1	$(id, 0, 1, 0, nbmess, cback, 0)$ $\rightarrow \left\{ \begin{array}{l} (id, 1, 2, 1, nbmess, cback, RTS(id)) : \frac{1}{2^{\max(10, (cback))}} \\ \vdots \\ (id, 1, 2, k, nbmess, cback, RTS(id)) : \frac{1}{2^{\max(10, (cback))}} \\ \vdots \\ (id, 1, 2, 2^{\max(10, cback)}, nbmess, cback, RTS(id)) : \frac{1}{2^{\max(10, (cback))}} \end{array} \right.$

Les transitions à partir de l'état 1 sont codées par les règles suivantes :

Nom	Règle
1-2	$(id, 1, 2, 0, nbmess, cback, RTS(id))$ $\rightarrow \{(id, 2, 0, Timeout, nbmess, cback, 0) : 1$
1-0	$(id, 1, 2, tatt, nbmess, cback, RTS(id))$ $\rightarrow \{(id, 0, 0, DIFS, nbmess, cback+1, 0) : 1$
1-OCTS	$(id, 1, 2, tatt, nbmess, cback, RTS(id))$ $\rightarrow \{(id, 0, 0, NAV, nbmess, cback+1, 0) : 1$

Le seul tirage aléatoire du protocole a lieu lors de l'entrée dans la place 1. Il sert à calculer la contrainte d'horloge permettant de spécifier quel sera le temps de séjour dans la place 1 avant que la transition vers l'état 2 puisse se déclencher. Naturellement cette dernière se produit si l'automate lit le mot vide sur l'entrée de la ressource de synchronisation correspondante.

Une fois dans la place 2, il existe deux façons de transiter vers une autre place :

- La première correspond au cas où la station reçoit une trame CTS et transite vers la place 3,
- la seconde correspond au cas où la trame CTS n'est pas reçue au bout de Timeout unités de temps car elle a soit été parasitée ou bien la trame RTS émise lors de la transition de la place 1 vers 2 a été parasitée par une trame RTS émise par une autre station au même moment.

On code ces deux transitions avec les règles de réécriture probabilistes suivantes :

³On fait un abus de notation en marquant $backoff(cback)$ dans chaque partie du membre droit de la première règle de réécriture probabiliste, puisque le tirage n'a lieu qu'une seule fois et que cette valeur est la même partout où elle apparaît dans la règle.

Nom	Règle
2-3	$(id, 2, 3, 0, nbmess, cback, 0)$ $\rightarrow \{(id, 3, 4, MSG_TIME, nbmess, cback, MSG) : 1$
2-0	$(id, 2, 3, tatt, nbmess, cback, 0)$ $\rightarrow \{(id, 0, 0, nbmess, cback+1, 0) : 1$

On note que la vérification de la condition d'horloge $t < Timeout$ sera effectuée par la stratégie codant le comportement du simulateur du produit d'automate. Pour appliquer la règle [2-0]Co1, on applique en fait la règle de réécriture [2-0]. Seules les conditions de déclenchement changent.

Une fois entré dans la place numéro 3, l'automate modélise l'envoi d'un paquet de données par la station émettrice qui dure un temps noté `MSG_TIME`. Cette transition est modélisée grâce à la règle de réécriture suivante :

Nom	Règle
3-4	$(id, 3, 4, 0, nbmess, cback, MSG)$ $\rightarrow \{(id, 4, 0, Timeout, nbmess, cback, 0) : 1$

Une fois cette trame de données émise, la station doit attendre que la station centrale lui renvoie une trame d'acquiescement. La place 4 modélise cette phase d'attente. On attend la réception de la trame `ACK` au plus `Timeout` unités de temps. Le protocole propose un mécanisme de retour à la place 0 en cas de non réception de la trame `ACK`, c'est à dire quand la station ne l'a pas reçue au bout de `Timeout` unités de temps. Dans le cas contraire, l'automate évolue vers la place 5 marquant la réception de cette dernière trame. Pour simuler cette phase, on programme un retour en 0 au bout de `Timeout` unités de temps mais on transite vers la place 5 si on reçoit la trame `ACK`.

Nom	Règle
4-0	$(id, 4, 0, 0, nbmess, cback, MSG)$ $\rightarrow \{(id, 0, 1, DIFS, nbmess, cback+1, 0) : 1$
4-5	$(id, 4, 0, tatt, nbmess, cback, MSG)$ $\rightarrow \{(id, 5, 0, TACK, nbmess, cback, 0) : 1$

De l'état 5 on peut soit transiter vers l'état 0 pour modéliser le fait que l'acquiescement prouve qu'une erreur de transmission s'est produite. Dans ce cas on comptabilise une collision. Dans le cas où on ne détecte pas d'erreur, alors on évolue vers l'état 0 d'un automate clone de celui présenté ci dessus mais où le nombre de messages à envoyer est décrémenté.

La transition [4-0]Co1 est aussi codée simulée par l'application de la règle de réécriture 4-0 car elle effectue exactement les mêmes changements sur le terme de simulation. Seules les conditions d'application de cette règle dans ce dernier cas changent.

Les deux règles de réécriture modélisant ces transitions sont :

Nom	Règle
5-0	$(id, 5, 0, tatt, nbmess, cback, write)$ $\rightarrow \{(id, 0, 0, nbmess, cback+1, 0) : 1$
5-0succ	$(id, 5, 0, tatt, nbmess, cback, write)$ $\rightarrow \{(id, 0, 0, nbmess-1, cback, 0, 0) : 1$

Maintenant, il nous reste à expliquer comment on code la synchronisation via la ressource de synchronisation. On notera par $\rightarrow_{transit}$ le système de réécriture formé des règles 0-0, 0-0CTS, 0-1, 1-2, 1-0, 1-0CTS, 2-3, 2-0, 3-4, 4-0, 4-5, 5-0 et 5-0succ.

7.3.3 Codage de la synchronisation et simulation de l'ensemble du système

Nous allons introduire dans cette section les règles de réécritures nécessaires pour simuler le réseau WIFI représenté ainsi qu'une politique de simulation ϕ_{simul} qui appliquera ces règles dans un ordre précis et sur les sous termes adéquats. Cet ordre correspond à l'ordre des opérations de l'algorithme de simulation décrit dans la section 7.3.1.

Le produit de synchronisation utilise les règles de réécriture permettant de calculer les transitions de chaque automate, présentées dans la section précédente. On a besoin en plus d'un autre ensemble de règles pour effectuer le tri de la liste des automates suivant l'ordre $>_{simul}$ ainsi que la mise à jour des champs `tatt` susceptibles d'être changés lors de la transition de chaque automate et lors de chaque opération d'écriture.

De même on va définir un ensemble de règles de réécriture pour gérer les gardes sur les horloges ainsi que les contraintes sur les variables.

Enfin, nous expliciterons la stratégie ϕ_{simul} qui applique les règles de réécriture permettant de faire transiter les automates du produits quand il le faut. En effet, il est presque toujours possible de réécrire un automate du système alors que l'algorithme de simulation exige que ces opérations soient appliquées dans un ordre précis et que certains traitements sur les données soient faits entre deux pas de réécriture.

Techniquement, on peut appliquer les règles de réécriture probabilistes à de nombreux endroits et dans un ordre arbitraire. La stratégie ϕ_{simul} permet d'appliquer ces règles dans l'ordre qui correspond à l'exécution de l'algorithme de simulation et de désambigüiser les cas où on peut appliquer plusieurs règles lors d'une même étape de l'algorithme.

La stratégie va également permettre d'ordonnancer les événements liés aux collisions. En regardant les transitions sortant des places respectivement 1, 2, 4 on constate

que l'observation d'un message sur l'entrée de la ressource de synchronisation déclenche de façon préemptive les règles 1-2, 1-0CTS respectivement 2-3, 4-5 au lieu de celle prévue, c'est à dire 1-0, respectivement 2-0 et 4-0.

Précisons dès maintenant que si l est une liste d'automates temporisés alors le terme `tri_liste(l)` se réécrit en une liste d'automates temporisés triés selon l'ordre $>_{simul}$ sans que les automates de l ne soient modifiés. Comme $>_{simul}$ est un ordre partiel, il peut exister plusieurs listes représentant une même configuration du système. On gère ce problème en appliquant les règles de tri et les règles calculant l'évolution des différents automates en les appliquant de gauche à droite sur les éléments de la liste. L'algorithme donne le même résultats de simulation pour deux listes différentes représentant le même système.

Décrivons maintenant de manière exhaustive la politique d'application des règles de réécriture pour coder le simulateur dans son ensemble.

En entrée on dispose d'un terme représentant le réseau WIFI (t, l, T) . Il faut trier la liste l avant de commencer à appliquer les règles de réécriture permettant de simuler l'évolution du système. La première règle de réécriture que la politique ϕ_{simul} va appliquer est la règle suivante :

$$(t, l, T) \rightarrow \{(t, tri_liste(l), T) : 1$$

Une fois la liste triée, la stratégies ϕ_{simul} applique une règle de réécriture pour incrémenter la date t de la valeur contenue par le champ `tatt` du premier élément de la liste l . Puis ϕ_{simul} applique les règles permettant de retrancher au champ `tatt` des automates suivant le premier automate de la liste l la valeur du champ `tatt` du premier terme de la liste l . Plus exactement ϕ_{simul} applique la première règle du tableau ci dessous et applique l'une des règles suivante autant de fois que possible.

<code>(t, (id, etat, dest, tatt, nbmess, cback, write).l, T)</code>
$\rightarrow \{ (t+tatt, (id, etat, dest, 0, nbmess, cback, write).majdate(l, tatt), T) : 1$
<code>majdate((id, etat, dest, tatt, nbmess, cback, write).l, t)</code>
$\rightarrow \{ (id, etat, dest, tatt-t, nbmess, cback, write).majdate(l, t) : 1$
<code>majdate(nil, t) $\rightarrow \{nil : 1$</code>

À ce stade, le premier élément de la liste est dans la place indiquée par la valeur de son champ `etat`. De même il est possible que plusieurs termes d'émetteurs aient leur champ `tatt` qui s'évalue à 0. Dans ce cas ces émetteurs atteindront leur état courant en même temps et il est possible qu'ils écrivent sur la ressource de synchronisation de façon simultanée. Dans ce cas ils peuvent générer une collision. C'est pourquoi, on doit tout d'abord calculer la valeur lue sur chaque entrée de la ressource de synchronisation avant d'appliquer les règles de réécriture permettant de faire transiter les automates codant les stations émettrices. De même la détection d'un message par une station qui

est soit dans l'état 1, soit dans l'état 2 ou encore dans l'état 4 provoque une transition immédiate vers l'état 0 dans les deux premiers cas et vers 5 dans le dernier cas. On déclenche ces transitions quand on a détecté un problème, comme une collision.

Pour faire cela on va commencer par présenter l'algorithme de calcul de la valeur lue sur chaque entrée de la ressource de synchronisation en fonction de l'état des automates.⁴

Dans le but de simplifier l'écriture de cet algorithme on adopte les conventions de notation suivantes :

Si l est une liste alors $\#l$ est le nombre d'éléments de cette liste et si $i < \#l$ alors $l[i]$ est la valeur du i ème élément de la liste l .

Algorithme: Ordonnancement

Entrée:

(t, l, T): Terme de simulation avec t :entier; l :Liste de Terme de stations;
 T Matrice d'accessibilité.

Données:

$TMessages[\#l]$: Tableau de liste de mots de l'alphabet formé
 par les messages du protocole CSMA-CA,
 de la taille de la liste l .

i, j, n : Entiers.

Algorithme:

```

n=#l;
Pour i de 1 à n
  TMessages[i]=nil.
fin Pour.
i=1
(id,etat,dest,nbmess,cback,write)=l[i];

Faire
  Pour j de 1 à n
    Si (T[id,j]=1 et write!=0) alors
      TMessages[j]=write.TMessages[j];
    Fin Si;
  Fin Pour;
//On concatène les messages reçus par j avec celui
//envoyé par le station courante si j est accessible
//depuis la station courante et que la station

```

⁴Cet algorithme peut se coder sous la forme d'un système de réécriture, mais sa lecture est plus immédiate sous une forme impérative.

```

//courante envoie un message
i=i+1;
(id,etat,dest,tatt,nbmess,cback,write)=l[i]
Tant que (tatt==0);

//Mecanisme de préemption
Pour i de 1 à n
(id,etat,dest,tatt,nbmess,cback,write)=l[i];
Si ((id==0 ou id==1 ou id==2 ou id==4) et TMessages[id]!=nil) alors
l[i]=(id,0,tatt,nbmess,cback,write)
// Le temps d'attente avant prochaine execution est mis à
// zéro, pour effectuer une transition immediate.
Fin Si

```

Fin Algorithme

Propriété 7.3.1. La complexité de cet algorithme est linéaire en le nombre de stations simulées.

La stratégie applique à nouveau un tri de liste sur l en réécrivant le système de la façon suivante :

$$(t, l, T) \rightarrow \{(t, \text{tri_liste}(l), T) : 1$$

Une fois qu'on a calculé la valeur lue sur chaque entrée de la ressource de synchronisation, on peut faire transiter les termes représentant les automates simulant les stations WIFI en appliquant les règles 0-0, 0-OCTS, 0-1, 1-2, 1-0,1-OCTS, 2-3, 2-0, 3-4, 4-0, 4-5, 5-0, 5-0succ grâce à l'algorithme présenté ci après, dont le principe est assez simple.

On fait transiter tous les automates représentés par un terme dont le champ `tatt` s'évalue à 0. Ceux-ci sont toujours en début de liste. Il suffit de regarder ce que vaut l'évaluation de `TMessages[id]` correspondant à l'automate `id` lisant la ressource de synchronisation sur l'entrée `id` pour savoir si l'automate en question :

- Ne reçoit pas de message, c'est à dire quand `TMessages[id]=nil`.
- Reçoit un seul message, c'est à dire quand `TMessages[id]=a.nil`. avec `a` correspondant à un caractère représentant les trames définies dans le protocole CSMA-CA.
- À détecté une superposition de signaux correspondant à une collision quand `TMessages[id]=a.*.b.nil`.

On précise qu'on note l'application d'une règle de réécriture nommée `nom` sur le i ème sous élément de la liste l de la façon suivante `[nom]l[i]`. De même on note par `[*]l[i]` l'application de n'importe quelle règle qui « filtre » le terme à réécrire. On rappelle qu'on a nommé les règles de réécriture décrivant les différentes transitions de

sortie à partir d'un état. On remarque qu'à partir du moment où il existe un terme représentant un automate qui a au moins un message à envoyer, il existe au moins une règle de réécriture permettant de réécrire cet état.

Lorsqu'une seule règle peut s'appliquer, la stratégie l'applique. C'est ce qui se passe quand on est dans l'état 3. Dans les cas où l'état est soit 0, soit 1, soit 2, soit 4 ou encore 5, il faut décrire de quelle façon on choisit la règle de réécriture à appliquer.

Algorithme: Transition

Entrée:

(t,l,T) : terme de simulation

Variable:

i : entier

Début:

```

i=1;
(id,etat,dest,tatt,nbess,cback,write)=l[i];
Tant que (tatt==0)
  switch (etat)
    case 1:
      Si (T[id]==nil) Alors
        (t,l,T)=(t,[1-2]l[i],T);
      Sinon
        (t,l,T)=(t,[1-0]l[i],T);
      Fin Si
    break;

    case 2:
      Si (T[id]==CTS) Alors
        (t,l,T)=(t,[2-3]l[i],T);
      Sinon Alors
        (t,l,T)=(t,[2-0])l[i],T);
      Fin Si;
    break;

    case 3:
      (t,l,T)=(t,[3-4]l[i],T);
    break;

    case 4:

```

```

    Si (T[id]==ACK(id)) Alors
        (t,l,T)=(t,[4-5]l[i],T);
    Sinon Alors
        (t,l,T)=(t,[4-0]l[i],T);
    Fin Si
break;

case 5:
    Si (T[id]==Checksum_ok) Alors
        (t,l,T)=(t,[5-0Succ]l[i],T);
    Sinon Alors
        (t,l,T)=(t,[5-0]l[i],T);
break;

default:
    (t,l,T)=(t,[*]l[i],T);

Fin switch;
Fin

```

Dans les cas par défaut, une seule règle de réécriture « filtre » le terme, c'est pour cela que l'on peut appliquer l'opération `[*]l[i]`. Une fois arrivé à ce stade on recommence l'opération depuis le début.

Pour résumer, écrivons les principales étapes de la stratégie sous la forme d'un pseudo algorithme :

Algorithme: Stratégie `phi_simul`:

Terme `(t,l,T)`

1) Trier `l` selon l'ordre de simulation

```
(t,l,T)=(t,[*]tri_liste(l),T);
```

```
(id,etat,dest,tatt,nbmess,cback,write)=l[0];
```

Tant que `(nbmess>0)`

2) Ordonancement(`(t,l,T)`);

3) `(t,l,T)=(t,[*]tri_liste(l),T)`;

4) Transition(`(t,l,T)`); C'est ici qu'ont lieu les étapes de réécriture probabilistes.

5) `(id,etat,tatt,nbmess,cback,write)=l[0]`;

Fin Tant que

Nous avons à ce stade décrit quelles sont les règles de réécriture probabilistes que nous utilisons pour coder la simulation ainsi que la façon avec laquelle nous appliquons ces règles, c'est à dire la stratégie en régit l'application. Nous allons grâce à cela pouvoir appliquer les méthodes de preuve de terminaison presque sûre présentées dans le chapitre 5 pour montrer que le système de réécriture probabiliste codant la simulation termine presque sûrement et positivement sous la stratégie ϕ_{simul} .

7.4 Terminaison +a.s de la simulation du CSMA-CA

L'algorithme de la stratégie décrit dans la section précédente, applique dans son étape 4) les règles probabilistes codant les transitions des différents automates du produit synchronisé, c'est à dire les règles de $\rightarrow_{transit}$. On rappelle que $\rightarrow_{transit}$ est l'ensemble des règles de réécriture appliquée lors de l'étape 4 de la stratégie ϕ_{simul} . On va montrer d'une part qu'il existe une borne supérieure – dépendante de l'instance du problème modélisé – concernant la longueur des dérivations séparant l'application de deux règles de l'étape 4) et d'autre part que les règles de $\rightarrow_{transit}$ ont un drift négatif pour une certaine fonction V qui restera constante lorsque les autres règles seront appliquées.

En d'autres termes, nous allons construire une fonction V qui vérifie les conditions du théorème 5.3.3 avec la stratégie ϕ_{simul} .

Commençons par montrer qu'il existe bien une fonction évaluant les termes de simulation vers les réels positifs dont le drift est négatif lors de l'application des règles de $\rightarrow_{transit}$.

7.4.1 Construction d'une fonction certifiant la terminaison +a.s sous stratégie

Pour passer d'un état n vers un état $n + 1$ avec $n \in \{0, \dots, 3\}$, un automate doit satisfaire un certain nombre de conditions. Faillir à certaines de ces conditions peut mener cet automate à retourner dans l'état 0. Pour marquer cela on désire construire une fonction mesurant la distance entre l'état courant d'un automate et l'état où celui-ci a émis tous ses messages. Une telle fonction diminuera quand on passera de l'état n vers l'état $n + 1$.

Pour mesurer la distance séparant l'état courant de l'ensemble des états terminaux la fonction que nous allons construire dépendra de deux paramètres :

- le nombre de messages restant à envoyer pour chaque station,
- ainsi que la phase d'émission dans laquelle se situe chaque automate.

Cette fonction que nous nommons $V : T(\Sigma, X) \rightarrow \mathbb{R}_+$ évalue les termes de simulation vers l'ensemble des réels positifs sera dépendante de la somme sur chaque

automate du nombre de messages restant à envoyer et de la somme de l'évaluation de chaque terme par une fonction W que nous expliciterons par la suite.

Précisons que les fonctions `nbmess` et `etat` évaluent les termes représentant les émetteurs de la façon suivante :

- `nbmess((id,etat,dest,tatt,nbmess,cback,write))) = nbmess`
- `etat((id,etat,dest,tatt,nbmess,cback,write))) = etat`.

La forme générale de la fonction V est avec K une constante positive que nous allons calculer.

$$V((t, l[1] . l[2] \dots l[\#l], T) = K \times \sum_{i=1}^{\#l} \text{nbmess}(l[i]) + W(\text{etat}(l[i]))$$

Remarque 7.4.1. On peut immédiatement dire que toute permutation des éléments de la liste l , que toute modification de la valeur du champ `tatt` ainsi que des champs `tatt`, `dest`, `cback`, `write` de chaque émetteur laisse l'évaluation du terme de simulation par la fonction V inchangée.

Propriété 7.4.1. Ainsi, on peut d'emblée dire que l'application des règles de réécriture effectuée à toutes les étapes de la stratégie ϕ_{simul} sauf à l'étape numéro 4) laissent inchangée l'évaluation du terme de simulation par la fonction V .

Parmi les opérations qui modifient la valeur de la fonction V on distingue celle où un automate décrémente son nombre de message à envoyer, c'est à dire lorsqu'on applique la règle de réécriture `5-0succ`, des autres règles de $\rightarrow_{\text{transit}}$. Si on applique la règle de réécriture `5-0succ` sur le terme d'automate i alors `nbmess(l[i])` diminue de 1 et W passe de la valeur $W(5)$ à $W(0)$. Lorsqu'une autre règle $\rightarrow_{\text{transit}}$ est appliquée sur le i -ème automate de la liste l , on change uniquement la valeur de $W(\text{etat}(l[i]))$.

On va déterminer une fonction W ainsi qu'un ensemble conditions \aleph qui sont suffisantes pour vérifier les hypothèses du théorème 5.3.3. Les conditions de l'ensemble \aleph signifient que les règles de $\rightarrow_{\text{transit}}$ ont un temps moyen de sélection borné sous la stratégie ϕ_s et que la probabilité conditionnelle par rapport aux tirages précédents des `backoff` de chaque station satisfait trois propriétés :

- la probabilité qu'une station dans l'état 1 de l'algorithme du CSMA-CA transite vers l'état 2 est supérieure à une constante $\alpha > 0$ fixée,
- la probabilité qu'une station dans l'état 2 de l'algorithme du CSMA-CA transite vers l'état 3 est supérieure à une constante $\beta > 0$ fixée,
- la probabilité qu'une station dans l'état 4 de l'algorithme du CSMA-CA transite vers l'état 5 est supérieure à une constante $\gamma > 0$ fixée.

Écrivons de manière plus formelle ces conditions : Soit $(I_n)_n$ la suite des temps d'arrêts où I_n est la longueur de l'histoire correspondant à la n ème application d'une règle de $\rightarrow_{\text{transit}}$ –c'est bien un temps d'arrêt– sous la stratégie ϕ_{simul} . On note par

\mathcal{F}_n la tribu engendrée par tous les tirages des variables aléatoires `backoff` de chaque automate durant les n premiers pas de réécriture. On se positionne dans le cas où on doit réécrire le terme d'automate `(id,etat,0,dest,nbmess,cback,write)` qui est un sous terme du terme de simulation obtenu au bout d'une histoire de longueur I_n , c'est à dire au moment où la stratégie déclenche une règle de $\rightarrow_{transit}$ dans l'étape 4) de la stratégie ϕ_{simul} . Par $P([1-2](id,1,0,dest,nbmess,cback,write)|\mathcal{F}_{I_n})$ on note la probabilité que la stratégie ϕ_{simul} puisse appliquer la règle de réécriture [1-2] sur le sous terme d'automate `(id,1,0,nbmess,cback,write)` dans l'état 1, sachant \mathcal{F}_{I_n} . On peut en effet mesurer la probabilité d'un tel événement conditionnellement à \mathcal{F}_n car les écritures sont conditionnées par les tirages des valeurs des `backoff`.

$$(\aleph) \left\{ \begin{array}{l} (i) \quad \exists M \in \mathbb{R} \text{ tel que } E[I_{n+1}|\mathcal{F}_{I_n}] \leq M + I_n \\ (ii) \quad \exists \alpha > 0 \text{ tel que } p_1 = P([1-2](id,1,0,nbmess,cback,write)|\mathcal{F}_{I_n}) > \alpha \\ \quad \forall n \in \mathbb{N} \\ (iii) \quad \exists \beta > 0 \text{ tel que } p_2 = P([3-4](id,3,0,nbmess,cback,write)|\mathcal{F}_{I_n}) > \beta \\ \quad \forall n \in \mathbb{N} \\ (iv) \quad \exists \gamma > 0 \text{ tel que } p_3 = P([4-5](id,4,0,nbmess,cback,write)|\mathcal{F}_{I_n}) > \gamma \\ \quad \forall I_n \in \mathbb{N} \end{array} \right.$$

Propriété 7.4.2. Les conditions \aleph impliquent la terminaison en temps moyen fini du système de réécriture de la simulation sous la stratégie ϕ_{simul}

Soit h_{τ_n} une histoire non terminale de longueur τ_n obtenue en appliquant I_n pas de réécriture sous la stratégie ϕ_{simul} . La fonction W a été construite de telle façon que si les hypothèses de \aleph sont vérifiées alors le drift de V en h_{I_n} est inférieur à -1 .

Démonstration. La stratégie ϕ_{simul} sélectionne par définition une règle de $\rightarrow_{transit}$ au temps d'arrêt I_n si celui-ci est fini. D'après la première hypothèse de \aleph si $I_n = \infty$ cela signifie que le système de réécriture se trouve presque sûrement sur un état terminal car les règles de ϕ_{simul} ont un temps de sélection moyen borné pour ϕ_{simul} .

Soit π une dérivation commençant par h et suivant les choix de la stratégie ϕ_{simul} . $\Delta_{I_n, \phi_{simul}/h} V$ le drift en h_{I_n} de la fonction V lors de la prochaine application d'une des règles de $\rightarrow_{transit}$ sera égal au drift de la fonction W si ϕ_{simul} sélectionne une règle de $\rightarrow_{transit} / \{5-0succ\}$ et vaudra -1 si elle sélectionne la règle `5-0succ`.

On a construit la fonction W pour qu'elle satisfasse les contraintes⁵ suivantes :

$$\left\{ \begin{array}{l} \forall p_1 \geq \alpha \quad p_1 \times W(2) + (1 - p_1) \times W(0) \leq W(1) - 1 \\ \forall p_2 \geq \beta \quad p_2 \times W(3) + (1 - p_2) \times W(0) \leq W(2) - 1 \\ \forall p_3 \geq \gamma \quad p_3 \times W(5) + (1 - p_3) \times W(0) \leq W(4) - 1 \end{array} \right. .$$

Remarque 7.4.2. On utilise abusivement les notations p_1, p_2, p_3 pour rendre la lecture des calculs à suivre plus aisée, mais il s'agit bien de variables aléatoires et non de constantes réelles.

⁵avec l'abus de notation décrit dans la remarque 7.4.2

C'est à dire que si $\rightarrow_{transit}$ s'applique au prochain coup sur un terme d'automate dans la phase 1 de l'algorithme du CSMA-CA alors :

$$\Delta_{I_n, \phi_{simul}/h} V = p_1 \times W(2) + (1 - p_1) \times W(0) - W(1) \leq -1.$$

Si $\rightarrow_{transit}$ s'applique au prochain coup sur un terme d'automate dans la phase 2 de l'algorithme du CSMA-CA alors :

$$\Delta_{I_n, \phi_{simul}/h} V = p_2 \times W(3) + (1 - p_1) \times W(0) - W(2) \leq -1,$$

mais si $\rightarrow_{transit}$ s'applique au prochain coup sur un terme d'automate dans la phase 4 de l'algorithme du CSMA-CA alors :

$$\Delta_{I_n, \phi_{simul}/h} V = p_3 \times W(5) + (1 - p_1) \times W(0) - W(4) \leq -1.$$

Dans les cas des transitions entre les états 0 et 1, 3 et 4 le drift vaut exactement -1 , car on applique avec probabilité 1 une règle qui fait décroître W de 1.

La règle **5-0succ** est appliquée avec probabilité 1. En effet dans le protocole il existe un mécanisme de contrôle d'erreur de demande de retransmission pour palier au problèmes de parasites radio extérieurs au réseau. Mais dans notre modélisation nous travaillons avec des transmission sans erreurs. L'application de **5-0succ** sur l'un des termes d'automates fait décroître de 1 l'évaluation du terme de simulation par la fonction V . Nous explicitons ici la fonction W en supposant que les trois dernières conditions de l'ensemble \aleph existent :

On construit une fonction W satisfaisant les hypothèses précédentes :

$$\begin{aligned} K &= 1 + \frac{2\alpha\beta + \alpha + 2}{\alpha\beta\gamma} \\ W(0) &= K \\ W(1) &= K - 1 \\ W(2) &= \frac{\alpha K - 2}{\alpha} \\ W(3) &= \frac{\alpha\beta K - \alpha - 2}{\alpha\beta} \\ W(4) &= \frac{\alpha\beta K - \alpha - 2}{\alpha\beta} - 1 \\ W(5) &= \frac{\alpha\beta\gamma K - 2\alpha\beta - \alpha - 2}{\alpha\beta\gamma} = 1 \end{aligned}$$

Donc pour conclure : $\aleph \Rightarrow \Delta_{I_n, \phi_{simul}/h} V \leq -1$ et sous la stratégie ϕ_{simul} le système de réécriture termine positivement et presque sûrement.

□

Maintenant, on va montrer que pour toute liste représentant un système réel où les automates démarrent à l'état 0 en même temps, le système de réécriture termine en temps moyen fini sous la stratégie ϕ_{simul} .

Théorème 7.4.1 (Terminaison en temps moyen fini de la simulation à partir ϕ_{simul}).
Le système de réécriture codant la simulation atteint un état terminal au bout d'un temps moyen à partir des états initiaux représentant un système où toutes les stations sont dans l'état 0 et ont leur horloges initialisée à 0.

Notation 10. On note T_s l'ensemble des termes de simulation, c'est à dire des termes correspondant à des configuration réelles du système modélisé.

Démonstration. Nous allons maintenant montrer que les conditions \aleph sont vérifiées.

Preuve de la condition (i) de \aleph

Les étapes 1), 2), 3), 5) de l'algorithme décrivant la stratégie ϕ_{simul} appliquent des règles dans un ordre qui correspond à l'exécution d'algorithmes dont la complexité est polynomiale en la taille des instance des systèmes modélisés. Par hypothèse, la taille de instances est fixée au début de l'exécution et demeurant constante durant toute l'exécution, puisque par hypothèse, aucune station ne peut entrer ni quitter le réseau. Ainsi on peut majorer le nombre de pas de réécriture séparant deux applications d'une règle de $\rightarrow_{transit}$ par l'évaluation d'un polynôme à une variable prenant en entrée la taille du problème modélisé. Ce qui permet d'affirmer que ϕ_{simul} a un temps de sélection borné pour les règles de $\rightarrow_{transit}$, c'est à dire que la condition (i) de \aleph est satisfaite.

Maintenant prouvons qu'il existe bien une constante positive α telle que décrite dans les conditions de \aleph . Commençons à nous intéresser aux conditions nécessaires qui permettent à une station d'accéder à l'état 1 de l'algorithme du CSMA-CA. Pour arriver en 1 la station a du écouter la ressource de synchronisation durant DIFS unités de temps sans que celle-ci n'ait pu lire autre chose que le mot vide ϵ sur son entrée.

Nous allons exhiber quelques propriétés vérifiées par le protocole CSMA-CA qui vont nous être nécessaires par la suite. Nous allons notamment montrer qu'une seule station émettrice peut se trouver dans les phases 3, 4 et 5 à un instant donné et que les autres se trouvent dans soit dans la phase 0, ou soit dans la phase 1. Nous distingueront deux cas de figures, le premier est celui où tout se passe correctement et où toute les stations transitent de façon synchrones de l'état 0 à l'état 1 après avoir correctement reçu la trame ACK émise par la station centrale et qui marque la fin d'un cycle d'émission. Le second cas correspond au fait qu'une station peut transiter vers son état 2 au même moment que la station centrale émet la trame CTS et que certaines stations reçoivent une cette trame CTS alors que d'autre non. Dans ce cas particulier, le cycle d'émission peut se poursuivre, mais certaines stations peuvent émettre des trame RTS en même temps qu'une trame de donnée est envoyée à la station centrale. Ce phénomène entraîne un parasitage des messages ainsi transmis. Nous allons identifier les causes provoquant un tel phénomène pour pouvoir par la suite mesurer la probabilité qu'il se produise sachant les tirages des variables aléatoires réalisés par le passé.

Propriété 7.4.3. Si tous les automates du produit se trouvent dans la phase 0 de l'algorithme du CSMA-CA et que leur horloges τ est initialisée à 0 à la même date et que la station centrale se trouve également dans la phase 0 de l'algorithme qui gère

son comportement, alors tous les automates d'émetteurs arrivent à la même date dans la phase 1 de l'algorithme du CSMA-CA.

Démonstration. Tout les automates doivent attendre un temps constant $DIFS$ sans jamais lire un autre mot que ϵ sur leur entrée de la ressource de synchronisation pour passer de l'état 0 à l'état 1. Comme tous les automates représentant le comportement des stations sont dans l'état 0, aucun ne peut effectuer d'opération d'écriture, de même pour la station centrale qui demeure reste dans son état 0 puisqu'elle ne recevra pas la Trame RTS nécessaire pour transiter vers son état 1 puis effectuer sa transition vers son état 2 au terme de laquelle elle enverra une trame CTS à toutes les autres stations. \square

Propriété 7.4.4. Il y a au plus un automate d'émetteur dans les état 3, 4 ou 5.

Démonstration. Si un automate est entré dans l'état 3, cela signifie que la borne centrale lui a envoyé une trame CTS et que cette trame n'a pas été parasitée par une autre émission. La borne centrale passe de l'état 1 à l'état 2 pour émettre la trame CTS et attend la réception d'une trame MSG avant $Timeout$ unités de temps.

Aucun autre message CTS ne sera réexpédié avant que la station centrale ne soit revenue dans l'état 0 et ait reçu une trame RTS non parasitée contenant son identifiant.

Une fois entré dans l'état 3, l'automate d'émetteur émettra son message au terme de la transition qui le mènera vers 4, puis attendra de recevoir une trame d'acquittement. \square

Propriété 7.4.5 (Sources des collisions à partir de l'état 1). On distingue deux types de causes provoquant des collisions lorsque les stations attendent à partir de l'état 1 avec les horloges τ mise à 0 à la même date.

La première catégorie de collisions que l'on prend en compte est celle qui est provoquée par l'émission simultanée de plusieurs trames RTS par deux stations, par exemple 1 et 2, qui sont mutuellement accessible. Cela se produit lorsque la valeur calculée par l'algorithme du `backoff` de la station 1 est la même que celle renvoyée par l'algorithme du `backoff` de la station 2.

La seconde classe de collisions prise en compte est l'ensemble des collisions qui se produisent entre deux stations mutuellement cachées. Elle se produisent lorsque deux stations ont tirés des valeur de `backoff` suffisamment proches pour qu'on ait l'un des deux phénomènes suivants :

- Les deux trames RTS se parasitent,
- la deuxième trame RTS est émise en même temps que la trame CTS émise par la station centrale.

La seconde classe de collisions ne peut être détectée par une station émettrice lors de l'arrivée dans l'état 2, et n'a pas de conséquences sur le bon fonctionnement de

l'algorithme car le signal de la station cachée est trop faible pour parasiter celui de la station centrale.

Si une collision de la première sorte se produit alors la station centrale n'enverra pas de trame CTS. En effet, dans le premier cas de figure, les trames RTS se parasiteront mutuellement et la station centrale ne pourra pas les reconnaître. Par conséquent, elle ne reverra aucune trame CTS et les stations 1 et 2 retourneront dans l'état 0 au bout de Timeout unités de temps.

Propriété 7.4.6 (Synchronisation). Lorsqu'une transmission s'est bien déroulée, toutes les stations dans l'état 0 reçoivent la trame ACK dans le même « slot » de temps. Elles transiteront toute dans le même slot de temps vers la place 1.

Dans un certain cas les trames MSG et ACK peuvent être parasitées par l'émission de trame RTS. On va étudier quelle sont les causes de ces collisions.

Propriété 7.4.7 (Cause de parasitage des trames MSG et ACK). Si une station transite de la place 1 vers la place 2 en même temps que la station centrale émet une trame CTS(1) alors l'émission des trames MSG(1) et ACK(1) peut être parasitée.

Démonstration. Soient \mathbf{s}_1 et $\{\mathbf{s}_i\}_{i \in \{1, \dots, n\}}$ l'ensemble des stations du réseau. Si à la date t_1 il existe une station s_j qui transite vers la place 2 et la station centrale transite de la place 1 vers la place 2, alors les opérations d'écriture suivantes ont lieu simultanément sur les entrées de la ressource de synchronisation :

$$\begin{aligned} & \mathbf{w}(\text{central_id}, \text{CTS}(1)) \\ & \mathbf{w}(j, \text{RTS}(j)). \end{aligned}$$

Regardons quelles sont les valeurs contenues par les ressources de synchronisation et déduisons en de quelle manière vont se comporter les différentes stations :

1. Soit i tel que $T[j, i] = 1$, alors $\mathbf{r}(i) = \{\text{CTS}(1), \text{RTS}(i)\}$,
 - si $\text{etat}(s_i) = 0$ on applique la transition $[0-0] s_i$,
 - sinon si $\text{etat}(s_i) = 1$ on applique la transition $[1-0] s_i$.
2. Soit i tel que $T[j, i] = 0$, alors $\mathbf{r}(i) = \text{CTS}(1)$,
 - si $\text{etat}(s_i) = 0$ alors on applique la règle $[0-ONAV] s_i$,
 - sinon si $\text{etat}(s_i) = 1$ alors on applique la règle $[1-ONAV] s_i$.

Si la station 1 est accessible par la station j , c'est à dire $T[j, 1] = 1$, alors la station 1 va retourner dans sa place 0 car elle va lire $\{\text{CTS}(1), \text{RTS}(i)\}$ au lieu de CTS(1). Ces stations devront séjourner DIFS unités de temps avant de transiter vers l'état 1. Néanmoins l'ensemble des stations i telles que $T[j, i] = 0$ liront la trame CTS(1) et transiteront dans la phase 0 de l'algorithme du CSMA-CA et séjourneront NAV unités de temps dans l'état 0. Si $T[j, 1] = 0$ alors la station 1 reçoit bien la trame CTS et transite vers la section critique. Cependant la station i retourne dans l'état 0 après Timeout unités de temps. Ainsi cette dernière station ainsi que celles ayant

reçu $\{\text{CTS}(1), \text{RTS}(i)\}$ peuvent atteindre l'état 1 puis l'état 2 car elles n'entendent pas les émissions de la station 1. Dans ce cas les trames RTS qu'elles émettent peuvent parasiter la trame de donnée MSG où la trame ACK. \square

Par la suite on montrera qu'on peut mesurer la probabilité que de tels événements se produisent conditionnellement au tirages des variables aléatoires effectuées par le passé.

Prouvons (ii) de \aleph en calculant α .

On peut considérer sans pertes de généralités qu'on se positionne sur l'automate 1. On pose que l'automate 1 est accessible par $n - 1$ automates et est caché pour les k autres stations restantes.

Pour que l'automate puisse transiter de la place 1 vers la place 2, il faut et il suffit qu'à la date où on observe le système que la date de prochaine exécution de l'automate 1 soit la plus petite au sens large. Cela signifie qu'il faut que cette station émette sa trame RTS avant toutes les autres stations ou au pire au même moment que certaines stations parmi les $n - 1$ stations à partir desquelles la station 1 est accessible. Ces dites stations sont soit dans la place 0 ou dans la place 1.

Si on note par cback_i le paramètre de **backoff** du i -ème automate que l'automate 1 peut entendre, alors la probabilité que la station 1 transite de son état 1 vers son état 2 vaut :

$$\begin{aligned} P_{succ[1-2]} &= P\left(\forall i \text{ backoff}(\text{cback}_1) \leq \text{backoff}(\text{cback}_i) + P_i\right) \\ &\geq P\left(\bigcap_{i=2}^n \text{backoff}(\text{cback}_1) \leq \text{backoff}(\text{cback}_i)\right) \end{aligned}$$

Ici P_i représente le temps qu'il reste à la station i avant qu'elle n'atteigne l'état 1 quand elle est en phase d'attente dans l'état 0. Cette valeur dépend des exécutions précédente et vaut 0 lors d'un départ synchronisé.

Comme les tirage des **backoff** ont lieu de manière indépendante [IEE03] alors

$$P_{succ[1-2]} \geq \prod_{i=2}^n P\left(\text{backoff}(\text{cback}_1) \leq \text{backoff}(\text{cback}_i)\right)$$

On trouve facilement un minorant du membre droit de l'expression précédente.

$$P_{succ[1-2]} \geq \frac{1}{2^{10 \times n}}$$

On vient de trouver une valeur pour α , avec

$$\alpha = \frac{1}{2^{10 \times n}}$$

c'est à dire que la condition (ii) de \aleph est vérifiée.

Preuve de (iii) en calculant la constante β .

Calculons maintenant une borne inférieure β de la probabilité que s'applique la règle qui simule la transition de la place 2 vers la place 3. Expliquons tout d'abord sous quelles conditions une telle règle peut s'appliquer. Pour déclencher cette règle la station doit lire le symbole CTS(1) sur son entrée de la ressource de synchronisation après avoir séjourné au plus `Timeout` unités de temps dans la place 2 .

Une station dans l'état 2 a déjà émis une trame RTS et attend la réception d'une trame CTS. Néanmoins, s'il existe une station qui a émis une trame RTS en même temps que la station 1 alors la station centrale ne reconnaît pas les trames RTS et par conséquent ne renvoie pas la trame CTS. Cette station peut être soit cachée ou non pour la station 1.

Remarque 7.4.3. Il est possible qu'une station cachée émette une trame RTS au même moment que la station centrale émet la trame CTS(1). Dans ce cas précis deux messages sont émis simultanément sur le canal radio dans le cas du réseau réel et on effectue alors deux écritures simultanées sur deux entrées différentes de la ressource de synchronisation quand on simule ce phénomène. Dans le cas réel la station 1 recevra clairement la trame CTS(1), car l'intensité du signal émis par la station cachée sera trop faible pour parasiter celui émis par la station centrale. Dans le cas de la simulation la station 1 lira bien CTS(1) sur son entrée de la ressource de synchronisation, conformément à ce qui se produit dans le cas réel.

Donc l'ensemble des événements à mesurer est celui où le tirage de `backoff(cback1)` est inférieur aux autres tirages de `backoff(cbacki) + Pi` où P_i est le temps restant à attendre avant que la station i entre dans sa place 1 au moment où à lieu le tirage de `backoff(cback1)`, si elle n'y est pas déjà⁶. En effet une station i qui est encore dans son état 0 peut quand même émettre une trame RTS en même temps ou avant la station 1 si jamais `backoff(cbacki) + Pi` est plus petit que la valeur `backoff(cback1)`.

Mesurons et bornons inférieurement la probabilité que ces événement se produisent, à commencer par l'événement :

$$P_{1,i} = P\left(\text{backoff}(\text{cback}_1) < \text{backoff}(\text{cback}_i) + P_i\right)$$

$P_{1,i}$ est la probabilité que le tirage du `backoff` de la station 1 soit plus petit que temps nécessaire à la station i pour déclencher sa transition de la place 1 vers la place 2.

$$P_{1,i} = \sum_{j=1}^{2^{\max(\text{cback}_1, 10)}} P(\text{backoff}(\text{cback}_i) + P_i > j | \text{backoff}(\text{cback}_1) = j) \\ \times P(\text{backoff}(\text{cback}_1) = j)$$

⁶Les P_i sont \mathcal{F}_n mesurables.

avec :

$$P \left(\begin{array}{l} \text{backoff}(\text{cback}_i) + P_i > j \\ |\text{backoff}(\text{cback}_1) = j \end{array} \right) = \begin{cases} 1 & \text{Si } P_i \geq j \\ \frac{2^{\max(\text{cback}_i, 10) - j + P_i}}{2^{\max(\text{cback}_i, 10)}} & \text{Sinon} \end{cases}$$

Alors la probabilité que l'on applique avec succès la règle [2-3] une fois que l'automate 1 se trouve dans sa place 2 :

$$\begin{aligned} P_{succ[2-3]} &= P(\forall i \text{backoff}(\text{cback}_1 < \text{backoff}(\text{cback}_i)) + P_i) \\ &= \prod_{l=1}^{n+k} \sum_{j=1}^{\text{cback}_l} P_{1,j} \\ &\geq \frac{1}{2^{10 \times (n+k)}} \end{aligned}$$

En posant $\beta = \frac{1}{2^{10 \times (n+k)}}$ on a une borne inférieure strictement positive de la probabilité que la règle [2-3] s'applique sur le terme représentant l'automate 1. On vérifie bien la propriété (iii) de \aleph .

Prouvons (iv) en calculant la constante γ

Dans la remarque 7.4.3 on mentionne le fait qu'une station cachée puisse émettre une trame RTS en même temps que la station centrale émet la trame CTS à la station 1. La station cachée en question va ensuite retourner au bout de **Timeout** unités de temps vers la place 0 car elle ne recevra pas de trame CTS. Si jamais la nouvelle valeur de **backoff** qu'elle tirera est petite, alors il se peut que cette station émette une trame RTS qui viendra parasiter la réception du message émis par la station 1. Dans ce cas de figure, le message aura été brouillé et la station centrale n'émettra pas de trame ACK ou enverra une trame ACK comportant un mauvais checksum. Nous allons calculer la probabilité qu'un tel événement se produise et montrer que la probabilité de recevoir correctement une bonne trame ACK est strictement supérieur à zéro. Pour que ce phénomène ne se produise pas il faut et il suffit que la station 1 émette sa trame RTS deux unités de temps avant la date prévue pour l'émission d'une trame RTS pour toutes les stations cachées pour la station 1.

Voici l'événement dont on veut mesurer la probabilité

$$succ_{[4-5]} := \forall i \in \{n, \dots, n+k\} \quad \text{backoff}(\text{cback}_1) \leq \text{backoff}(\text{cback}_i) + P_i - 2$$

En effectuant un changements de variables $(P')_i = P_i - 2$ on pose

$$\begin{aligned} (P')_{1,i} &= \sum_{j=1}^{2^{\max(\text{cback}_1, 10)}} P \left(\begin{array}{l} \text{backoff}(\text{cback}_i) + (P')_i > j \\ |\text{backoff}(\text{cback}_1) = j \end{array} \right) \\ &\quad \times P(\text{backoff}(\text{cback}_1) = j) \end{aligned}$$

avec :

$$P \left(\begin{array}{l} \text{backoff}(\text{cback}_i) + (P')_i > j \\ |\text{backoff}(\text{cback}_1) = j \end{array} \right) = \begin{cases} 1 & \text{Si } P'_i \geq j \\ \frac{2^{\max(\text{cback}_i, 10) - j + (P')_i}}{2^{\max(\text{cback}_i, 10)}} & \text{Sinon} \end{cases}$$

On peut maintenant calculer un minorant de la probabilité que la règle [4-5] s'applique :

$$\begin{aligned} P_{succ_{[4-5]}} &= \prod_{l=n}^{n+k} \sum_{j=1}^{cback_l} (P')_{1,j} \\ &\geq \frac{1}{2^{10 \times k}} \end{aligned}$$

On prend donc $\gamma = \frac{1}{2^{10 \times k}}$ qui est une valeur strictement positive. On a prouvé que la condition est vérifiée. À ce stade, on a vérifié toutes les conditions de \aleph donc on a achevé la preuve du résultat énoncé. \square

Remarque 7.4.4 (À propos de α , β et γ). Dans ce cas d'étude il était simplement question de vérifier les conditions (ii), (iii) et (iv) de \aleph , c'est à dire montrer que la probabilité de sélectionner les "bonnes" règles était positive. C'est ce que nous venons de faire en effectuant des majorations très grossières. De meilleures bornes inférieures pourraient être calculées et permettraient d'améliorer la qualité des bornes supérieures concernant le nombre moyen de pas de réécriture nécessaires pour atteindre l'ensemble des états terminaux.

7.5 Conclusion

7.5.1 En résumé

Nous avons dans ce présent chapitre modélisé un réseau de stations WIFI communiquant grâce au protocole 802.11b. Ce protocole utilise un algorithme distribué probabiliste qui permet à chaque station de partager l'accès à la ressource radio sans mutuellement provoquer des collisions. Grâce à aux notions de produit synchronisé d'automates temporisés que nous avons introduit dans le chapitre 6.2 et de systèmes de réécriture probabilistes et de stratégies introduits dans les chapitres 4 et 5, nous avons pu modéliser de façon formelle le comportement d'un tel réseau.

Nous avons par la suite appliqué les résultats concernant la terminaison en temps moyen fini des systèmes de réécriture probabilistes sous stratégie pour montrer que le système modélisé atteignait un état terminal dans un temps moyen fini, et cela à partir d'états modélisant un départ synchronisé de toutes les stations.

En plus de prouver que le protocole CSMA-CA vérifie des propriétés de qualité de service et de vivacité, le travail présenté dans ce chapitre donne un aperçu des possibilités de modélisation de systèmes qui offrent la réécriture probabiliste et la réécriture probabiliste sous stratégies. Il met également en évidence le fait que l'on puisse prouver de façon formelle des propriétés de terminaison en temps moyen fini d'algorithmes probabilistes ou plus généralement l'accessibilité en temps moyen fini de classes d'états.

L'étude des protocoles probabilistes tels que le CSMA-CA ou le CSMA-CD ont été à l'origine d'une prolifique littérature et de nombreuses approches. Ces études répondent au besoin qu'ont les concepteurs de tels protocoles de pouvoir vérifier que

leurs programmes et algorithmes satisfont ou non à certaines propriétés de sûreté et de qualité de service. L'approche proposée par la communauté model-checking consiste à fournir un formalisme de modélisation de systèmes, un logiciel de « model-checking » qui vérifie si certaines propriétés ou formules sont satisfaites par le modèle donné en entrée. On peut citer deux « models checkers » APMC [HLMP03] et PRISM [KNP02, HKNP06, PRI], le premier permettant de vérifier de façon approchée la probabilité qu'une formule LTL monotone soit vérifiée sur une chaînes de Markov et le second permettant de vérifier des propriétés probabilistes exprimés sous la forme de formule PCTL ou CSL sur des modèles exprimés grâce aux automates temporisés probabilistes. Le premier utilise des méthode de Monte-Carlo pour approcher la probabilité de satisfaction de propriétés monotone alors que le second ramène le problème de satisfaction de formules à la résolution d'une instance du problème du « Stochastic Shortest Path » [BT91]. Ces deux méthodes ne peuvent s'appliquer que des systèmes où l'espace des états est fini. Cependant, les formalisme de description de modèle utilisé par les « model checkers » ont atteint une certaine maturité depuis l'introduction de ces outils [CE81, Eme81, QS82] et les logiciels développés permettent de vérifier des propriétés sur des systèmes concret. Néanmoins, de tels outils buttent toujours au problème de l'explosion combinatoire quand la taille des modèles augmentent. C'est ici que les outils développés dans cette thèse pourraient avoir leur rôle à jouer. En effet, les méthodes de preuve développés pour prouver la terminaison de programmes à base de règles de réécriture probabiliste [BG06] s'appliquent sur des espaces de termes dénombrables mais non nécessairement finis. La complexité de la preuve dans le dernier cas réside dans le fait de trouver une fonction d'évaluation des termes satisfaisant les propriétés du théorème 5.3.3.

Dans le cas d'étude présenté dans ce chapitre on obtient un résultat de terminaison en temps moyen fini indépendant du nombre de stations et dans le cas où des stations sont mutuellement cachées. Une étude du protocole CSMA-CA802.11b a également été réalisé sous le model checker PRISM, mais pour le cas où seules deux stations ont des messages à s'échanger. Ces résultats sont présentés dans [KNS02] et permettent de mettre en évidence le problème de l'explosion combinatoire de l'espace d'états que le « model checker » doit générer pour vérifier les formules données en entrée en même temps que le modèle. En effet, PRISM génère une chaîne de Markov concurrente correspondant à toute les exécutions possibles du modèle donné en entrée et le nombre d'état de cette chaîne de Markov concurrente croit exponentiellement avec la taille du modèle étudié. Malgré les excellentes optimisations utilisés par ce model checker, le problème de l'explosion combinatoire du nombre d'états générés en fonction de la taille du modèle demeure inévitable et empêche la validation de certains systèmes au delà d'une certaine taille en utilisant ces méthodes. PRISM comme les système de réécriture probabilistes permet de certifier que certains systèmes probabilistes modélisables sous la forme d'une chaîne de Markov finis vérifient des formules au bout

d'un « temps moyen fini », en l'occurrence des propriétés de terminaison. APMC, en utilisant des méthodes de Monte Carlo ne permet que d'obtenir des solutions approchées de propriétés quantitatives mais avec un coût algorithmique moindre qu'avec une génération exhaustive de tout les états possibles. Ces optimisations ont été obtenues entre autres grâce à l'utilisation d' ϵ -abstractions probabilistes [Pey03]. Ces deux approches complémentaire ont mené à la fusion des outils APMC et PRISM.

En définitive, le model checking probabiliste et la vérification de propriétés de terminaison de programmes à base de règles probabilistes sont complémentaires dans le sens où à l'heure actuelle il n'existe pas de méthode permettant au model checking de faire de la vérification de propriétés sur des systèmes à espace d'états infinis mais dénombrables et que la preuve de terminaison en temps moyen finis de système de réécriture probabiliste sous stratégie n'est pas encore automatisée. De même nous n'avons pas encore développé de méthode pour prouver des propriétés quantitatives sur les systèmes implémentables sous la forme de systèmes de réécriture probabilistes.

8

Codage des systèmes de réécriture probabilistes

Dans cette thèse, nous avons défini la notion de système de réécriture probabiliste et présenté une définition générale des stratégies de réécriture, comme étant un mécanisme de contrôle de l'application des règles de réécriture probabilistes. Dans ce chapitre, nous allons présenter une manière d'implémenter de tels systèmes en utilisant le langage TOM [Mor, BBK⁺06]. Ce langage a été développé pour permettre d'utiliser facilement les mécanismes de la réécriture dans des langages très répandus, tels le langage C et le langage Java. Pour exprimer les systèmes de réécriture probabilistes dans TOM, il faut pouvoir être capable de représenter une règle de réécriture probabiliste. De plus, pour générer une exécution d'un système de réécriture probabiliste, on a besoin d'un mécanisme d'expression de stratégie. Les outils intégrés à TOM permettent d'exprimer de façon intuitive l'application des règles de réécriture sur des termes et le contrôle de l'application de ces règles grâce à des stratégies. Pour définir l'espace des termes, TOM intègre un langage de description de syntaxe appelé Gom. Grâce à la combinaison TOM+Gom il est aisé de décrire des systèmes de réécriture ainsi que des stratégies de réécriture. Nous allons brièvement présenter dans la section 8.1 ces aspects dans le cas de systèmes de réécriture, puis dans la section 8.2 nous montrerons comment exprimer des règles de réécriture probabilistes ainsi que des stratégies. Le lecteur désireux d'en apprendre d'avantage sur les possibilités de TOM, et elles sont nombreuses, peut consulter la thèse d'Antoine Reilles [Rei06] et la page de TOM [Mor]. Le but de ce chapitre est de montrer que l'implémentation des systèmes de réécriture probabiliste est très aisée en utilisant le langage TOM ainsi que le langage de description de syntaxe Gom. De même nous montrerons qu'il est facile d'implémenter des stratégies de contrôle de la réécriture en utilisant ce langage.

8.1 Exprimer des systèmes de réécriture avec TOM

8.1.1 Définir l'espace des termes

Afin de définir l'espace des termes d'un système de réécriture $(T(\Sigma, X), \rightarrow)$, on peut utiliser le langage de spécification algébrique `Gom`. `Gom` permet de définir la syntaxe des termes qu'on désire manipuler et associe cette syntaxe à des « sortes ». De plus, `Gom` propose des « opérateurs » permettant de définir des ensembles de termes et des contraintes algébriques sur les termes décrits. Ce langage offre par ailleurs des opérateurs pour définir la syntaxe de nouvelles sortes à partir d'autres sortes déjà définies, ainsi qu'un mécanisme de modulaire permettant d'importer des sortes déjà définies dans des fichiers externes.

Considérons l'exemple suivant :

```
Exemple 8.1.1. %gom {
module network
imports String int

  abstract syntax
    Sender = sender(eta:int,dest:int,tatt:int,nbmess:int,cback:int,write:int)
    List = concSender(Sender*)
    Etat = etat(x:int, t:int,y>List)
}
```

Dans l'exemple ci-dessus, on déclare la syntaxe des sortes `Sender`, `List` et `Etat`, en utilisant le langage de description de syntaxe `Gom`. `Gom` permet d'importer des définitions de syntaxes, grâce aux modules. Toujours dans l'exemple ci-dessus on importe la définition des sortes `String` et `int`. Nous utilisons ces définitions de syntaxe afin de pouvoir définir la syntaxe des sortes `Sender`, `List` et `Etat`. Ainsi, en écrivant `Etat = etat(x :int, t :int,y :List)` on spécifie que la syntaxe associée à la sorte `Etat` est `: etat(x,t,y)` où `etat` est un constructeur et où `x,t` sont des variables de la sorte `int` et `y` a pour sorte `List`. L'un des avantages de `Gom` est qu'il permet de définir des constructeurs avec les mêmes domaines et co-domaines.

Le compilateur TOM traduit les programmes écrits en TOM+Java qui effectuent des opérations sur une algèbre de termes décrite en `Gom` vers le langage Java. Durant cette opération, le compilateur TOM génère les classes Java permettant de manipuler les sortes décrites en `Gom`, telles des objets Java. Ainsi, dans l'exemple 8.1.1 on a défini une sorte `List`. TOM crée alors un objet dont le type est `List`, et on pourra dans la suite du programme définir des variables de type `List`, pouvant être utilisées en paramètre de fonctions Java et également en tant que type de retour.

8.1.2 Gestion des termes par TOM

TOM utilise le partage maximal pour gérer les termes qu'il manipule. C'est à dire, qu'à un terme on associe une et une seule adresse mémoire. Ainsi, pour comparer l'égalité de deux termes, TOM compare les adresses de ces deux termes. Si celles-ci sont égales alors les deux termes sont égaux sinon les deux termes sont différents. Cette technique permet d'avoir des test d'égalité de termes en temps constant. Néanmoins, à chaque fois que l'on modifie un sous terme d'un terme, il faut reconstruire l'ensemble du terme dans un autre emplacement mémoire. Cette façon de faire nécessite d'avoir des mécanismes efficaces de gestion de la mémoire, notamment de « garbage collecting », chargé de libérer les objets et les termes stockés en mémoires qui ne sont plus référencés.

Afin de signaler à TOM qu'on construit un nouveau terme, les concepteurs de TOM ont choisit d'utiliser l'opérateur « backquote » « ` » pour spécifier qu'on est bien en train de construire un nouveau terme.

8.1.3 Manipuler les termes avec TOM

On va décrire ici les mécanismes de manipulation des termes que propose TOM et qui vont nous permettre d'implémenter les règles de réécriture probabilistes. TOM offre un mécanisme de filtrage permettant d'associer à certains motifs, c'est à dire à une famille de termes, des actions. On peut grâce à ce mécanisme représenter des règles de réécriture, qui associent aux instances d'un membre gauche les mêmes instances de leur membre droit.

Rappelons que si \mathcal{R} est un système de réécriture sur l'algèbre de termes $T(\Sigma, X)$, alors deux termes x et y de $T(\Sigma, X)$ sont en relation pour la relation de réécriture $\rightarrow_{\mathcal{R}}$ s'il existe une position $p \in Pos(x)$, une substitution σ telle que $x|_p = \sigma(l)$ et $y = r[\sigma(r)]_p$.

La vérification de l'existence du couple position-substitution s'appelle le problème du filtrage, ou du « pattern matching ». TOM offre une façon d'exprimer ce type de conditions et d'associer à un terme qui vérifie une condition, une action. Par exemple, en TOM, on peut associer à un terme qui est une instance du membre gauche d'une règle de réécriture l'action qui remplace ce terme par l'instance correspondante du membre droit de la même règle de réécriture.

La structure de contrôle dont on parle ici est la structure `%match`. Nous allons brièvement présenter son utilisation dans le cas où on travaille sur des signatures définies grâce à `Gom`. On peut comparer la structure `%match` à la classique structure `switch/case` définie dans les langages C et Java à ceci près qu'elle permet de manipuler des termes complexes au lieu de structures atomiques.

Exemple 8.1.2. Considérons que l'on travaille sur la signature définie en `Gom` dans l'exemple 8.1.1.

```
public List concList(List l, List m) {
  %match(List l, List m){
    concSender(C1*),concSender(C2*) -> {return 'concSender(C1*,C2*);
  }
}
```

Dans cet exemple on définit une fonction qui prend en paramètre deux variables de type `List`, dont le type a été défini à partir de la syntaxe décrite en Gom. Si le terme `l` est de la forme `concSender(C1*)` et le terme `m` de la forme `concSender(C2*)`, alors la fonction construit puis retourne terme `concSender(C1*,C2*)` –remarquez la présence de l’opérateur « backquote ».

L’exemple 8.1.2 effectue la concaténation de deux listes dont la sorte est `List`. En effet, à partir des termes contenu dans les constructeurs `concSender`, respectivement nommés `C1*` et `C2*`, un nouveau terme `concSender(C1*,C2*)` est construit, puis renvoyé par la fonction.

L’exemple 8.1.2 présente une manière de coder une règle de réécriture. TOM intègre une construction particulière `%rule` permettant de définir des règles de réécriture, mais `match` en définissant des couples motifs-actions est plus commode à utiliser quand on veut mettre un terme en relation avec une distribution de probabilité sur les termes.

Exemple 8.1.3. Considérons la structure `%match` suivante :

```
%match (Etat e) {
  etat(t,x,concSender(
  C1@sender(eta,tatt,nbmess,cback,write),C2*)) -> {
    if(nbmess>0 && t>0) {
      l2='nextstep(concSender(C1,C2*));
      %match(List l2){
        concSender(CC1,CC2*) -> {
          return simul ('etat(t-1,x+tatt,
                        sortList(concList(concSender(CC1),
                        majdate(CC2*,tatt)))) ));
        }
      }
    }
    else return e;
  }
}
```

Comme le montre l'exemple 8.1.3, on peut référencer les champs de chaque terme et conditionner les actions en fonction de la valeur à laquelle ces champs s'évaluent. De plus, il est possible d'imbriquer les structures `match` et de les appliquer sur des termes nouvellement construits.

Le fait de pouvoir conditionner les actions en fonction de l'évaluation des sous termes, permet de coder la réécriture conditionnelle et de définir des contraintes pour l'application des règles.

Exemple 8.1.4. Soit $f \in \Sigma^3$ et $g \in \Sigma^2$ deux fonctions et r_1, r_2 deux règles de réécriture telle que $r_1 := f(1, g(x, y), z) \rightarrow g(y, z)$ et $r_2 := f(a, b, g(x, x)) \rightarrow g(a, a)$. Nous allons coder cette règle de réécriture en utilisant les mécanismes de TOM.

```
%gom{
module exemple
imports Nat
  abstract syntax
    Term = Nat(intValue:int)
          | f(x:Term,y:Term,z:Term)
          | g(x:Term,y:Term)
}

// fin du module gom

public Nat reduction(Nat t) {
  %match(Term t) {
    f(Nat(1),g(x,y,z)) -> {return 'g(y,z)}
    f(a,b,g(x,x)) -> {return 'g(a,a)}
  }
  return t;
}
```

On vient ainsi de définir grâce à TOM une fonction qui permet d'appliquer un pas de réécriture du système de réécriture composé des deux règles r_1 et r_2 .

8.1.4 Les stratégies de réécriture sous TOM

TOM intègre un langage de stratégies [BBK⁺07, Rei06] dont la sémantique est claire et bien définie. Ces stratégies permettent de définir un cadre formel propre, rendant réaliste l'étude de propriétés qu'ont les programmes codés à base de règles de réécriture dont l'application est régie grâce à de telles stratégies. De plus, le langage de stratégie intégré à TOM permet de composer les opérateurs de stratégies et de

s'appliquer jusqu'à ce qu'un point fixe soit éventuellement atteint. Le langage de description de stratégies de parcours arborescent et d'application de règles disponible en TOM permet également de prendre en compte, dans une certaine mesure, le non déterminisme comme dans le langage de stratégie de ELAN [BKK⁺98].

On note que des mécanismes de stratégies permettent d'énumérer l'ensemble des sous-termes d'un terme et permettent également d'appliquer une stratégie sur un sous terme déterminé et de détecter l'échec ou le succès de l'application d'une stratégie de réécriture sur un sous terme.

8.2 Codage des systèmes de réécriture probabilistes avec TOM

Nous allons voir, que le langage TOM permet, lorsqu'il est utilisé avec Java, de coder les règles de réécriture probabilistes. Nous montreront qu'il en est de même pour les stratégies probabilistes.

8.2.1 Écriture des règles de réécriture probabilistes

Nous rappelons qu'une règle de réécriture probabiliste est un couple de la forme

$$l \rightarrow \mu \in T(\Sigma, X) \times \text{Dist}(T(\Sigma, X)),$$

où $\text{Dist}(T(\Sigma, X))$ est l'ensemble des distributions de probabilité sur l'espace des termes $T(\Sigma, X)$.

Pour implémenter une telle règle, nous utilisons toujours la construction `match` de la même façon que dans le cas non probabiliste, mais on construit le successeur du terme en appliquant un tirage aléatoire du successeur. Le langage Java fournit des classes et des fonctions de génération de nombres pseudo aléatoires de bonne qualité, permettant de calculer un grand nombre de distributions.

Prenons un cas très simple, celui d'une règle de réécriture probabiliste réécrivant le terme $s(x)$ en x avec probabilité un demi et en $s(s(x))$ avec probabilité un demi. Le codage d'une telle règle en TOM se fait de la façon suivante :

```
public Term rewrite(Term t) {
    int rnd;
    Random a = new Random;
    %match(Term t) {
        s(x) -> {
            rnd=a.nextInt(100);
            if (a%2==1)
                return 's(s(x));
```

```

    else
        return 'x;
    }
}
return t;
}

```

Plus généralement, si on désire implémenter une règle de la forme $l \rightarrow \mu$ avec μ une distribution sur \mathbb{Q} telle que

$$\begin{aligned} \mu(t_1) &= \frac{p_1}{q_1} \\ &\vdots \\ \mu(t_k) &= \frac{p_k}{q_k} \\ &\vdots \\ \mu(t_n) &= \frac{p_n}{q_n} \end{aligned}$$

alors, à partir d'un nombre tiré uniformément sur l'ensemble des entiers

$$\{0, \dots, \max = p_1 \times (q_2 \times \dots \times q_n) + \dots + p_k \times (q_1 \times q_{k-1} \times q_{k+1} \times \dots \times q_n) + \dots + p_n \times (q_1 \dots q_{n-1})\},$$

on calcule le choix du successeur en utilisant l'algorithme ci-dessous pour que le choix ainsi fait suive la distribution de probabilité μ :

```

public Term rewrite(Term t) {
    int rnd;
    Random a = new Random();
    int bound;
    %match(Term t) {
    l -> {
        rnd=a.nextInt(max);
        bound=p1*(q2*...*qn);
        if (a<bound)
            return 't1;
        bound=bound+p2*(q1*q3*...*qn)
        if (a<bound)
            return 't2;
            .
            .
            .
        bound=bound+pk*(q1*...*qk-1*qk+1*...*q_n)
        if (a<bound)
            return 'tk
    }
}

```

```

        .
        .
        .
    if (a<bound)
        return 'tn
    }
return t;
}

```

Il est également possible de réécrire un terme suivant le tirage d'une variable aléatoire implémenté en Java,

```

%gom {
imports
int

abstract syntax
Term = couple (x:int,y:int)
}

public Term rewrite(Term t) {
    int rnd;
    Random a = new Random;
    %match(Term t){
        couple(x,y) -> {
            return 'couple(y,a.nextInt(x));
        }
        return t;
    }
}

```

Dans le dernier cas, si $x \geq 1$, on met en relation le terme (x,y) avec la distribution qui charge uniformément l'ensemble des termes $\{(y,z) | z \in \{0, \dots, x-1\}\}$. En d'autres termes, on applique la règle de réécriture suivante :

$$(x,y) \rightarrow \begin{cases} (y,0) & : \frac{1}{x} \\ \dots & \\ (y,x-1) & : \frac{1}{x} \end{cases}$$

8.2.2 Appliquer les règles de réécriture probabilistes sous stratégie

On a défini les stratégies de réécriture comme étant un algorithme contrôlant l'application des règles de réécriture en relation avec le dernier terme d'une histoire. Le langage TOM a été en premier lieu développé comme un pré-processeur pour les langages C et Java permettant d'ajouter une primitive de filtrage [MRV03, Rei06]. Cette spécificité permet à elle seule de définir des règles de réécriture et de contrôler leur application grâce à des algorithmes écrits en Java .

8.3 Implémentation de quelques exemples

8.3.1 Marche aléatoire

Reprenons le système de réécriture de l'exemple 5.2.2,

$$r_3 := s(x) \rightarrow \begin{cases} x & : p_1 \\ s(s(x)) & : 1 - p_1 \end{cases}$$

$$r_4 := s(x) \rightarrow \begin{cases} x & : p_2 \\ s(s(x)) & : 1 - p_2 \end{cases}$$

et considérons que l'on veuille appliquer une fois sur deux la règle r_3 et une fois sur deux la règle r_4 . Nous allons présenter façon d'implémenter ce système de réécriture avec TOM sous la stratégie qui sélectionne alternativement les deux règles. on définit tout d'abord l'espace des termes :

```
package pproba;

import java.util.Random;
import pproba.ppeanoproba.rwalk.types.Nat;

public class Ppeanoproba {

    static Random a = new Random();

    %gom {
        module rwalk
        abstract syntax
        Nat = Succ(val:Nat)
            | Zero()
    }
```

Une fois l'espace de termes défini, on définit le programme TOM lui même,

```
public final static void main(String[] args) {
    Nat t = 'Succ(Succ(Succ(Succ(Zero()))));
    transit(t,0);
}
```

composé d'une fonction `main` et d'une fonction décrivant les transitions entre les termes dont les sortes ont été définies avec `Gom`.

```
public static Nat transit(Nat etat, int par) {

    /*
     * On applique alternativement deux règles de réécriture
     * La première réécrit Succ(val)->{ val: 3/4 | Succ(Succ(val)):1/4
     * La seconde réécrit Succ(val)->{ val:1/2 | Succ(Succ(val)):1/2
     */
    %match(etat) {
        Succ(val) -> {
```

Si la parité est impaire alors on applique la règle de réécriture r_3 ,

```
        if (par == 1) {
            if (a.nextInt(3)<=2)
                return transit('val,(par+1)%2);
            else
                return transit('Succ(Succ(val)),(par+1)%2);
        }
```

sinon on applique la règle r_4 .

```
        else {
            if (a.nextInt(3)<=1)
                return transit('val,(par+1)%2);
            else
                return transit('Succ(Succ(val)),(par+1)%2);
        }
    }
    return etat;
}
```

L'algorithme d'application des règles est bien une stratégie car la parité de la longueur des histoires parcourues est bien une fonction des histoires vers $\mathbb{Z}/2\mathbb{Z}$.

8.3.2 Réécriture sous stratégie aléatoire Markovienne

Nous considérons que nous travaillons sur le même espace de termes que dans l'exemple précédent. Cependant, au lieu d'appliquer alternativement les règles de réécriture probabilistes, nous allons appliquer la règle r_3 avec probabilité $2/3$ et r_4 avec probabilité $1/3$

```
public Nat transit (Nat etat){
  %match(etat) {
    Succ(val) -> {
      if (a.nextInt(2)<= 1) {
```

Si on tire une valeur entière uniformément sur $\{0, 1, 2\}$ et que la valeur réalisé est 0 ou 1, alors on applique la règle r_3 ,

```
      if (a.nextInt(3)<=2)
        return transit('val);
      else
        return transit('Succ(Succ(val)));
    }
  }
```

sinon si la valeur réalisée vaut 2 alors on applique la règle r_4 :

```
    else {
      if (a.nextInt(3)<=1)
        return transit('val);
      else
        return transit('Succ(Succ(val)));
    }
  }
}
return etat;
}
```

8.3.3 Réécriture sous une stratégie aléatoire non Markovienne

On va coder une stratégie de réécriture qui applique la règle r_3 avec probabilité $\frac{n-1}{n}$ pour toute histoire non terminale de longueur n . Pour faire cela, on redéfinit la fonction `transit` de la façon suivante :

```
public Nat transit (Nat etat, int l){
  %match(etat) {
    Succ(val) -> {
      if(a.nextInt(l)<= l-1) {
```

Si on tire une valeur entière uniformément sur $\{0, \dots, l\}$ et que la valeur réalisé est inférieur à $l - 1$ alors on applique r_3 ,

```

if(a.nextInt(3)<=2)
    return transit('val,l+1);
else
    return transit('Succ(Succ(val)),l+1);
}

```

sinon la valeur réalisée vaut l alors on applique la règle r_4 :

```

else {
    if (a.nextInt(3)<=1)
        return transit('val,l+1);
    else
        return transit('Succ(Succ(val)),l+1);
}
}
}
return etat;
}
}

```

8.4 Conclusion

Le langage TOM associé avec Gom et Java permet de représenter et de calculer des dérivations de systèmes de réécriture probabilistes sous stratégies et sous stratégies aléatoires. TOM et Gom adjoignent à Java les mécanismes de définitions syntaxe et de « pattern matching » nécessaires pour définir l'application de règles de réécriture sur un espace de termes. Java permet de contrôler l'application de ces règles de réécriture en offrant une bibliothèque de fonctions très riche qui comprend notamment des fonctions de génération de nombres pseudos-aléatoires.

Conclusion et perspectives

Contributions

Nous allons résumer en quelques points les principales contributions présentées dans ce mémoire de thèse :

Un modèle de systèmes de réécriture probabiliste

Nous avons présenté dans cette thèse un modèle de réécriture probabiliste qui étend le pouvoir d'expression de la réécriture classique en lui ajoutant un mécanisme d'expression de phénomènes aléatoires.

Définition de la notion de stratégie

De même nous avons adapté la notion de « politique » utilisée pour étudier les « processus de décision markoviens » à notre formalisme afin de permettre d'étudier les propriétés que vérifient ses traces d'exécution. Nous avons appelé cette adaptation « stratégie » car le contrôle d'application des règles de réécriture probabiliste qu'il apporte correspond aux stratégies de réécriture dans le cas des systèmes de réécritures.

Étude du problème de la terminaison en temps moyen fini

Nous nous sommes penchés sur la question de la terminaison de ces systèmes de réécriture probabilistes et avons apporté un ensemble de critères impliquant la terminaison avec probabilité 1 et d'autres conditions plus fortes impliquant cette fois-ci la terminaison en temps moyen fini de ces mêmes systèmes. Le cas de la terminaison en temps moyen fini a fait l'objet de notre attention, car il permet de mettre en évidence des propriétés de qualité de service, comme la terminaison en temps moyen fini de systèmes probabilistes réels comme certains protocoles de télécommunication.

Terminaison en temps moyen fini sous stratégie

Nous nous sommes également intéressés à l'étude de la terminaison en temps moyen fini des systèmes de réécriture probabilistes quand le choix de l'application des règles de réécriture probabiliste est conditionné par des stratégies. Nous présentons

des conditions suffisantes que doivent satisfaire les stratégies pour qu'elles puissent permettre à un systèmes de réécriture de terminer en temps moyen fini.

Application du modèle de système de réécriture probabiliste

Nous appliquons les modèles de systèmes de réécriture probabilistes à des cas d'étude concret. Nous avons d'une part utilisés le mécanisme de réécriture pour modéliser un produit synchronisé d'automates temporisés permettant de modéliser un réseau d'ordinateurs communiquant avec le protocole « WIFI ». Nous avons appliqué nos méthodes de preuve de terminaison en temps moyen fini sous stratégie pour montrer que la modélisation d'un tel réseau parvenait à un état terminal au bout d'un temps moyen borné, ce qui en l'occurrence signifie que l'algorithme distribué du WIFI parvient bien à faire émettre toute les données de chaque station au bout d'un temps moyen fini.

Implémentation du systèmes de réécriture probabiliste

Nous avons étudié certains aspects de l'implémentation des systèmes de réécriture probabiliste, dans un premier temps dans le prototype de langage ELAN4 [BGK05], puis dans le langage TOM. Nous avons également implémenté l'algorithme de simulation dont nous prouvons la terminaison en temps moyen fini dans le langage TOM.

Preuve formelle versus Model-Checking

L'étude des protocoles probabilistes tels que le CSMA-CA ou le CSMA-CD ont permis d'appliquer certaines méthodes de vérification automatiques et a servit de banc d'essais pour en valider de nouvelles . Ces études répondent au besoin qu'ont les concepteurs de tels protocoles de pouvoir vérifier que leurs programmes et algorithmes satisfont ou non à certaines propriétés de sûreté et de qualité de service. L'approche proposée par la communauté model-checking consiste à fournir un formalisme de modélisation de systèmes, un logiciel de « model-checking » qui vérifie si certaines propriétés ou formules sont satisfaites par le modèle donné en entrée. Bien que les technologies des logiciels de model-checking aient atteints une grande maturité depuis leur introduction [CE81, Eme81, QS82], ces outils buttent toujours sur le problème de l'explosion combinatoire quand la taille des modèles augmente.

C'est ici que les outils développés dans cette thèse pourraient avoir leur rôle à jouer. En effet, les méthodes de preuve développés pour prouver la terminaison de programmes à base de règles de réécriture probabiliste [BG06] s'appliquent sur des espaces de termes dénombrables mais non nécessairement finis. La complexité de la preuve dans le dernier cas réside dans le fait de trouver une fonction d'évaluation des termes satisfaisant les propriétés du théorème 5.3.3.

Dans le cas d'étude présenté on obtient un résultat de terminaison en temps moyen fini indépendant du nombre de stations et dans le cas où des stations sont mutuellement cachées. Une étude du protocole CSMA-CA802.11b a également été réalisé sous le model checker PRISM, mais pour le cas où seules deux stations ont des messages à s'échanger. Ces résultats sont présentés dans [KNS02] et permettent de mettre en évidence le problème de l'explosion combinatoire de l'espace d'états que le « model checker » doit générer pour vérifier les formules données en entrée en même temps que le modèle.

En définitive, le model checking probabiliste et la vérification de propriétés de terminaison de programmes à base de règles probabilistes sont complémentaires dans le sens où à l'heure actuelle il n'existe pas de méthode permettant au model checking de faire de la vérification de propriétés sur des systèmes à espace d'états infinis mais dénombrables et que la preuve de terminaison en temps moyen finis de système de réécriture probabiliste sous stratégie n'est pas encore automatisée. De même nous n'avons pas encore développé de méthode pour prouver des propriétés quantitatives sur les systèmes implémentables sous la forme de systèmes de réécriture probabilistes.

Perspectives

L'étude de ce modèle de réécriture probabiliste et son rapprochement avec d'autres formalismes de descriptions de systèmes probabilistes a soulevé un bon nombre d'envies et a posé beaucoup de questions.

- La première de ces envies, est de déterminer une relation d'équivalence permettant de comparer l'expressivité des systèmes de réécriture probabilistes, puis s'en servir pour représenter de façon concise les systèmes de réécriture équivalents modulo cette relation. La relation la plus intéressante à nos yeux est la bisimulation probabiliste introduite dans [SL95]. En effet elle permettrait de garantir que deux systèmes de réécriture peuvent générer les même dérivations.
- Définir un langage de stratégies probabilistes cohérent et capable d'exprimer des fréquences d'application de classe de règles et définir un ensemble d'opérateurs pour mesurer la probabilité de certaines classes d'événements.
- Intégrer ce langage de stratégies à TOM.
- Adapter des méthodes de preuves automatiques de terminaison déjà existantes dans le cas de la réécritures classique au cas probabiliste ouvrirait une voie intéressante pour l'étude de propriété de terminaison sur des systèmes à espace d'états infinis. Étudier l'adaptabilité de méthodes de preuve utilisant une version probabiliste des ordre de simplification, par exemple RPO, déboucherait peut être sur le développement d'outil de « vérification automatique » pouvant travailler sur des modélisations de systèmes à espace d'état dénombrable de cardinalité quelconque. Les travaux effectués dans cette thèse ont déjà permis

d'étendre certaines méthodes de preuve de terminaison par induction, concernant la terminaison de systèmes de réécriture sous stratégies. En effet, dans [Gna07], l'auteur définit la notion de terminaison presque sûre positive sous stratégies inductives, et donne des algorithmes permettant de vérifier automatiquement la terminaison presque sûre positive de systèmes de réécriture probabilistes sous ces stratégies. Ces résultats nous encouragent à orienter nos recherches dans cette voie, car il montrent que la preuve automatique de terminaison en temps moyen fini de systèmes de réécriture probabiliste sous stratégies peut être implémentée pour des classes de stratégies génériques, telles que celles présentées dans le papier cité précédemment.

Conclusion

Du point de vue de la communauté « réécriture » ce travail présente un nouveau formalisme qui met à la portée de la réécriture la possibilité d'exprimer des choix de successeurs suivant une loi de probabilité. Cela permet d'exprimer des systèmes de transitions probabilistes tels que les processus de décision Markoviens à espace d'état dénombrable.

Du point de vue de celui qui habituellement étudie les systèmes de transitions probabilistes, tels les processus de décision Markoviens, le formalisme de la réécriture probabiliste permet de définir des relations de transitions complexes de façon concise sur des systèmes à espace d'état infinis mais dénombrables et d'étudier des problèmes d'accessibilité de classes d'états pour certaines sous catégories de traces d'exécutions que nous avons appelé stratégie.

Enfin, en considérant le travail effectué dans le domaine de la preuve de terminaison de programme à base de règles de réécriture, il est permis d'espérer pouvoir adapter les méthodes de preuves déjà existantes pour prouver la terminaison en temps moyen fini de certains systèmes de réécriture probabilistes. Les résultats présentés dans cette thèse permettent déjà de certifier la terminaison en temps moyen fini de certains systèmes, mais au coût d'une analyse fine. Les critères de terminaison sous stratégie permettent cependant de faciliter la preuve de terminaison des systèmes de réécriture dans certains cas.

Index

- ϵ , 15
- CSMA-CA
 - exponential backoff, 130
 - Symboles
 - ACK, 129
 - CTS, 129
 - DIFS, 129
 - MSG, 129
 - NAV, 129
 - RTS, 129
 - SIGNAL, 129
 - TACK, 129
 - TCTS, 129
 - TRTS, 129
 - backoff, 129
 - MSG_TIME, 129
- WIFI, 141
- Algèbre de termes, 14
- Atome, 33
- Chaîne de Markov, 37
 - Apériodicité, 41
 - Composantes connexes, 39
 - Ergodique, 42
 - Irréductibilité, 39
 - Période, 41
 - Propriété de Markov fort, 38
- Clos sous Σ -opérations, 18
- Clos sous substitution, 18
- Compatibilité avec les Σ -opérations, 18
- Composition
 - Relations binaires, 10
- Conditions
 - \aleph , 149
 - Preuve de (i), 151
 - Preuve de (ii), 151, 154
 - Preuve de (iii), 155, 156
 - Preuve de (iv), 157
- Convergence
 - L^p , 32
 - En loi, 32
 - En probabilité, 32
 - Presque sûre, 31
- Couple état-action, 46
- Décroissance par substitution, 81
- Espérance
 - Conditionnelle, 34
 - Mathématique, 30
- Exécution
 - D'un PDM, 45
- Fonction
 - De coût, 47
 - De coût Terminale, 47
 - Indicatrice, 33
 - Monotone, 21
- Forme normale, 11
- Histoire, 59
- Image inverse, 13
- Indépendance, 29
- Instance
 - D'un terme, 17
 - D'une variable, 16
- Irréductibilité, 11
- Lemme

- Borel-Cantelli, 29, 31
- Limite
 - D'une suite de v.a, 29
 - D'une suite monotone d'événements, 29
- Loi
 - Conditionnelle, 33
 - D'une variable aléatoire, 28
- Martingale, 35
 - L^1 , 36
 - Finie, 36
- Mesure
 - Asymptotique, 38
 - Invariante, 39
- Model checker
 - « PRISM », 126
 - APMC, 53, 158
 - PRISM, 53
- Opérateur de Bellman, 48
- Ordre
 - De réécriture, 20
 - De réduction, 20
 - Polynomial, 23
 - Préfix, 15
- pCTL, 52
- pCTL
 - Syntaxe, 52
- Position racine, 15
- Préservation contextuelle, 80
- Probabilité
 - Conditionnelle, 33
 - Par rapport à une tribu, 34
- Problème du plus court chemin stochastique, 48
- Problème du plus court chemin stochastique
 - Instance négative, 48
 - Instance positive, 48
- Processus de décision Markovien, 45
- Processus de décision markoviens
 - Politique, 46
- Propriété de Markov, 37
- Réécriture
 - Système de, 19
 - Règle, 18
- Réductible, 11
- Résolution de SSP, 48
- Règle de réécriture
 - $\rightarrow_{transit}$, 141
 - 0-0, 0-0CTS, 0-1, 1-2, 1-0, 1-0CTS, 2-3, 2-0, 3-4, 4-0, 4-5, 5-0, 5-0succ, 141
 - Probabiliste, 71
- Redex, 19
- Relation
 - Confluence, 11
 - Convergente, 11
 - De réécriture, 19
 - De réécriture probabiliste, 72
 - De réduction, 17
 - Normalisante, 11
 - Propriété de Church-Rosser, 11
 - Terminaison, 11
- Signature, 14
- Simulation
 - CSMA-CA
 - $>_{simul}$, 137
 - backoff, 130
 - (id,eta,tatt,nbmess,cback,write), 136
 - nil, 136
 - pow, 130
 - random, 130
- Sous-martingale, 36
- Stratégie, 59
 - ϕ_{simul} , 141, 149
- Substitution, 16
- Successeur, 11

Surmartingales, 36
Système complet d'événements, 33
Système de réécriture probabiliste (PRS),
72
Système de réduction abstrait (ARS), 9

Temps d'arrêt, 36
Toyama, 25
Tribu
Cylindrique, 38
Engendrée par une variable aléatoire,
35

Well founded induction, 11

Table des figures

1	Relation de transition probabiliste	3
2.1	Une marche aléatoire avec plusieurs choix de successeurs	45
3.1	Une marche aléatoire non déterministe.	59
3.2	Pourquoi introduire la notion de terminaison positive et presque sûre.	62
4.1	Une règle de réécriture probabiliste	73
4.2	La relation de réécriture appliquée à un terme	73
4.3	La règle r_1 peut s'appliquer en deux position.	75
4.4	La règle r_1 appliquée à la position p_2	76
5.1	Simulation du système de réécriture probabiliste par un processus de décision Markovien	89
5.2	Comportement du PARS sous une stratégie aléatoire	89
5.3	Relation de réécriture codée dans 5.2.2	90
5.4	Comportement du PRS sous une stratégie Markovienne aléatoire	91
5.5	Comportement du PRS décrit dans l'exemple 5.2.3.	92
5.6	Trajectoire suivie par l'évaluation de termes successifs.	94
6.1	Un automate temporisé simple	107
6.2	Topologie d'un réseau de communication	109
6.3	Lecture et écriture sur la ressource de synchronisation	111
6.4	Synchronisation entre deux automates	114
6.5	0 : Le bureau du thésard qui va encore rendre ses documents en retard	115
6.6	Les bureaux intermédiaires.	115
6.7	Le grand chef.	116
6.8	Un exemple de deadlock.	117
6.9	Synchronisation entre deux automates.	118
6.10	Un automate temporisé synchronisable avec variables.	122
7.1	Représentation d'un réseau de stations avec borne centrale	124
7.2	Table des symboles du CSMA/CA	127

7.3	Phases d'un envoi d'un paquet de données par le protocole CSMA-CA en fonction du temps.	128
7.4	Comportement de la station centrale.	130
7.5	Automates des phases du CSMA-CA	132
7.6	Opérations effectuées lors des transitions de la figure 7.5.	132

Bibliographie

- [ACD93] R. ALUR, C. COURCOUBETIS & D. DILL – « Model checking in dense real time », in *Information and Computation*, vol. 104, 1993, p. 2–34.
- [AG97] T. ARTS & J. GIESL – « Proving innermost normalisation automatically », in *Proceedings 8th Conference on Rewriting Techniques and Applications*, Lecture Notes in Computer Sciences, vol. 1232, Springer Verlag, 1997, p. 157–171.
- [Bal01] G. BALBO – « Introduction to stochastic Petri nets », *Lecture Notes in Computer Science* **2090** (2001), p. 84–155.
- [BBK⁺06] É. BALLAND, P. BRAUNER, R. KOPETZ, P.-E. MOREAU & A. REILLES – « The Tom manual », 2006, Disponible à l’adresse <http://tom.loria.fr/soft/release-2.4/manual-2.4/index.html>.
- [BBK⁺07] — , « Tom : Piggybacking rewriting on java », in *Proceedings of the 18th Conference on Rewriting Techniques and Applications*, Lecture Notes in Computer Science, Springer-Verlag, 2007.
- [BDA95] A. BIANCO & L. DE ALFARO – « Model checking of probabilistic and non deterministic systems », in *Fondation of Software Technology and Théoretical Computer Science* (S. Verlag, éd.), Lecture Notes in Computer Science, 1995, p. 499–513.
- [Ber01] J. BERTOIN – « Probabilités : Cours de licence de mathématiques appliquées », Tech. report, Université de Paris VI, 2001, Disponible en ligne à l’adresse suivante : <http://www.proba.jussieu.fr>.
- [BF99] B. BÉRARD & L. FRIBOURG – « Automated verification of a parametric real-time program : The ABR conformance protocol », in *CAV* (N. Halbwachs & D. Peled, éd.), Lecture Notes in Computer Science, vol. 1633, Springer, 1999, p. 96–107.
- [BG05] O. BOURNEZ & F. GARNIER – « Proving positive almost sure termination », (Nara, Japan) (J. Giesl, éd.), Lecture Notes in Computer Science, vol. 3467, Springer, 2005, p. 323–337.
- [BG06] O. BOURNEZ & F. GARNIER – « Proving positive almost sure termination under strategies », in *RTA* (F. Pfenning, éd.), Lecture Notes in Computer Science, vol. 4098, Springer, 2006, p. 357–371.

- [BGK05] O. BOURNEZ, F. GARNIER & C. KIRCHNER – « Stratégies de réécriture probabilistes dans ELAN 4 », Tech. report, LORIA, 2005.
- [BH03] O. BOURNEZ & M. HOYRUP – « Rewriting logic and probabilities », in *Rewriting Techniques and Applications, 14th International Conference, RTA-03* (Valencia, Spain) (R. Nieuwenhuis, éd.), LNCS 2706, Springer, jun 2003, p. 61–75.
- [BHK03] C. BAIER, B. HAVERKORT & J.-P. KATOEN – « Model checking algorithm for continuous time Markov chains », in *IEEE Transactions on Software Engineering* (S. Verlag, éd.), vol. 29, July 2003, p. 524–541.
- [BK98] C. BAIER & M. KWIATKOWSKA – « Model checking for a probabilistic branching time logic with fairness », *DISTCOMP : Distributed Computing* **11** (1998).
- [BK02] O. BOURNEZ & C. KIRCHNER – « Probabilistic rewrite strategies. applications to ELAN », in *Rewriting Techniques and Applications, 13th International Conference, RTA-02* (Copenhagen, Denmark) (S. Tison, éd.), LNCS 2378, Springer, jul 2002, p. 252–266.
- [BKK⁺98] P. BOROVSANÝ, C. KIRCHNER, H. KIRCHNER, P.-E. MOREAU & C. RINGEISSEN – « An overview of ELAN », in *Proceedings of the second International Workshop on Rewriting Logic and Applications* (Pont-à-Mousson (France)) (C. Kirchner & H. Kirchner, éd.), vol. 15, <http://www.elsevier.nl/locate/entcs/volume16.html>, Electronic Notes in Theoretical Computer Science, 1998.
- [BKKR01] P. BOROVSANÝ, C. KIRCHNER, H. KIRCHNER & C. RINGEISSEN – « Rewriting with strategies in ELAN : a functional semantics », *International Journal of Foundations of Computer Science* **12** (2001), no. 1, p. 69–98.
- [BL98] P. BARBE & M. LEDOUX – *Probabilité*, Belin, 1998.
- [BN98] F. BAADER & T. NIPKOW – *Term rewriting and all that.*, Cambridge University Press, 1998.
- [BT91] D. BERTSEKAS & J. TSITSIKLIS – « An analysis of stochastic shortest path problems », *Mathematics of Operations Research*, vol. 16, 1991, p. 580–595.
- [Buc76] B. BUCHBERGER – « A theoretical basis for the reduction of polynomials to canonical forms », in *ACM SIGSAM Bulletin*, vol. 10, 1976, p. 19–29.
- [CE81] E. CLARKE & E. EMERSON – « Design and synthesis of synchronisation skeletons using branching time temporal logic. », in *Logic of programs : Workshop*, Lecture Notes in Computer Science, vol. 131, 1981.

-
- [CM00] M. CLAVEL & J. MESEGUER – « Reflection and strategies in rewriting logic », in *Electronic Notes in Theoretical Computer Science* (J. Meseguer, éd.), vol. 4, 2000.
- [CMMU] E. CONTEJEAN, C. MARCHÉ, B. MONATE & X. URBAIN – « The cime system version 2.02 », Disponible à l'adresse suivante : <http://cime.lri.fr>.
- [con] T. I. CONSORTIUM – « The csma/ca 802.11 working group homepage », <http://www.ieee802.org/11/>.
- [CY95] C. COURCOUBETIS & M. YANANAKAKIS – « The complexity of probabilistic verification », in *Journal of the ACM*, vol. 42, 1995, p. 857–907.
- [dA97] L. DE ALFARO – « Formal verification of probabilistic systems », Thèse, Stanford University, 1997.
- [dA99] — , « Computing minimum and maximum reachability times in probabilistic systems. », in *In Proc. of CONCUR 99.*, Lecture Notes in Computer Science, vol. 1664, 1999, p. 66–81.
- [DA00] L. DE ALFARO – « From fairness to chance », *Electronic Notes in Theoretical Computer Science* **22** (2000).
- [Dau89] M. DAUCHET – « Simulation of Turing machines by a left-linear rewrite rule. », in *RTA*, 1989, p. 109–120.
- [Der70] C. DERMAN – *Finite-state Markovian decision processes*, Academic Press, 1970.
- [DFH⁺05] M. DUFLOT, L. FRIBOURG, T. HÉRAULT, R. LASSAIGNE, F. MAGNIETTE, S. MESSIKA, S. PEYRONNET & C. PICARONNY – « Probabilistic model checking of the CSMA/CD protocol using PRISM and APCM », in *Proceedings of the 4th International Workshop on Automated Verification of Critical Systems (AVoCS'04)* (London, UK) (M. R. A. Huth, éd.), vol. 128, Electronic Notes in Theoretical Computer Science, no. 6, Elsevier Science Publishers, 2005, p. 195–214.
- [Eme81] E. EMERSON – « Branching time temporal logic and the design of correct concurrent programs », Thèse, Harvard University, 1981.
- [Fel68] W. FELLER – *An introduction to probability theory and its applications*, vol. 1, Wiley, 1968.
- [FGK02a] O. FISSORE, I. GNAEDIG & H. KIRCHNER – « Cariboo : An introduction based proof tool for termination with strategies », in *Proceedings of Fourth International Conference on Principles and Practice of Declarative Programming*, October 2002, p. 62–73.
- [FGK02b] — , « Outermost ground termination », in *Proceedings of Fourth International Workshop on Rewriting Logic and Its Applications*, Electronic

- Notes in Theoretical Computer Science, vol. 71, Elsevier Science Publisher, September 2002.
- [FGK03] O. FISSORE, I. GNAEDIG & H. KIRCHNER – « Simplification and termination of strategies in rule-based languages. », in *PPDP '03 : Proceedings of the 5th ACM SIGPLAN international conference on Principles and practice of declarative programming* (NY, USA), ACM Press, 2003, p. 124–135.
- [FGK04] —, « A proof of weak termination providing the right way to terminate », in *Theoretical Aspects of Computing – ICTAC 2004 : First International Colloquium*, Lecture Note in Computer Science, Springer, 2004.
- [FN85] G. FLORIN & S. NATKIN – « Les réseaux de Pétri stochastiques », in *TSI*, vol. 4, 1985, p. 143–160.
- [Gir04] A. GIROUX – *Mesure et intégration*, Université de Montréal, Département de Mathématiques et Statistiques, 2004, Disponible à l'adresse suivante <http://www.dms.umontreal.ca/~giroux/mesure.html>.
- [Gna07] I. GNAEDIG – « Induction for Positive Almost Sure Termination », in *Proceedings of the Ninth ACM-SIGPLAN International Symposium on Principles and Practice of Declarative Programming* (Wroclaw, Poland), ACM Press, July 2007, p. 167–177.
- [GWM⁺93] J. GOGUEN, T. WINKLER, J. MESEGUER, K. FUTATSUGI & J.-P. JOUANNAUD – « introducing obj », in *Application of Algebraic Specification using OBJ* (J. Goguen, éd.), Cambridge, 1993.
- [HKNP06] A. HINTON, M. KWIATKOWSKA, G. NORMAN & D. PARKER – « Prism : A tool for automatic verification of probabilistic systems », in *12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'06)*, (I. H. Hermanns & J. Palsberg, éd.), Lecture Notes in Computer Science, vol. 3920, Springer, 2006, p. 441–444.
- [HL78] G. HUET & D. LANKFORD – « On the uniform halting problem for term rewriting systems », Tech. report, Le Chesnay, 1978.
- [HLMP03] T. HÉRAULT, R. LASSAIGNE, F. MAGNIETTE & S. PEYRONNET – « Approximate probabilistic model checking », in *Verification, Model Checking, and Abstract Interpretation : 5th International Conference, VMCAI 2004 Venice, Italy, January 11-13, 2004 Proceedings* (G. L. Bernhard Steffen, éd.), vol. 2937, Springer Verlag, Berlin, Heidelberg, New York, 2003, p. 73–84.
- [HS85] S. HART & M. SHARIR – « Concurrent probabilistic programs, or : How to schedule if you must », *SIAM Journal on Computing* **14** (1985), no. 4, p. 991–1012.

-
- [HSP83] S. HART, M. SHARIR & A. PNUELI – « Termination of probabilistic concurrent program », *ACM Transactions on Programming Languages and Systems* **5** (1983), no. 3, p. 356–380.
- [IEE03] IEEE 802.11 WORKING GROUP – *Information technology telecommunication-exchange between systems-local and metropolitan area networks-specific requirement- part11 :wireless lan medium acces control (mac) and physical layer (phy) specification*, ANSI/IEEE Std 802.11, 1999 (Revised 2003).
- [Jac03] J. JACOD – « Chaînes de Markov, processus de Poisson et applications », Tech. report, Université de Paris VI, <http://www.proba.jussieu.fr>, 2003.
- [KGSS02] M. KWIATKOWSKA, N. GETHIN, R. SEGALA & J. SPROSTON – « Automatic verification of real-time systems with discrete probability distributions », in *Theoretical Computer Science*, vol. 282, 2002, p. 101–150.
- [KKV95] C. KIRCHNER, H. KIRCHNER & M. VITTEK – « Designing constraint logic programming languages using computational systems », in *Principles and Practice of Constraint Programming* (P. van Hentenryck & V.Saraswat, éd.), MIT Press, 1995, p. 131–158.
- [KNP02] M. KWIATKOWSKA, G. NORMAN & D. PARKER – « PRISM : Probabilistic symbolic model checker. », in *12th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation (TOOLS'02)*, Lecture Notes in Computer Science, vol. 2324, April 2002, p. 200–204.
- [KNS] M. KWIATKOWSKA, G. NORMAN & R. SEGALA – « Verifying quantitative properties of continuous probabilistic timed automata », in *CONCUR'2000-Concurrency Theory*, vol. 1877, p. 123–137.
- [KNS02] M. KWIATKOWSKA, G. NORMAN & J. SPROSTON – « Probabilistic model checking of the ieee 802.11 wireless local area network protocol », in *Workshop on Process Algebra and Performance Modelling and Probabilistic Methods in Verification* (H. Hermanns & R. Segala, éd.), Lecture Notes in Computer Science, vol. 2399, Springer-Verlag, 2002, p. 169–187.
- [KNS03] —, « Probabilistic model-checking of deadline properties in the ieee 1394 firewire root contention protocol », *Formal Aspects of Computing* **14** (2003), p. 295–318.
- [LP05] J. LACROIX & P. PRIOURET – « Probabilités approfondies, cours de master », Tech. report, Université de Paris VI, 2005, Disponible en ligne à l'adresse suivante <http://www.proba.jussieu.fr/>.
- [LR81] D. LEHRMAN & M. RABIN – « On the advantage of free choice : a symmetric and fully-distributed solution to the dining philosopher problem. »,

- in *8th Annual ACM Symp. on Principles of Programming Languages (POPL'81)*, 1981, p. 133–138.
- [Mes92] J. MESEGUER – « Conditional rewriting logic as unified model of concurrency », *Theoretical Computer Science* **96** (1992), no. 1, p. 73–155.
- [Mes04] S. MESSIKA – « Méthodes probabilistes pour la vérification des systèmes distribués », Thèse, École Normale Supérieure de Cachan, 2004.
- [Mol81] M. MOLLOY – « On the intergration of delay and throughput in distributed processing models », Thèse, University of Colombia, Los Angeles, California, September 1981.
- [Mor] P.-E. MOREAU – « The tom home page », <http://tom.loria.fr>.
- [MRV03] P.-E. MOREAU, C. RINGEISSEN & M. VITTEK – « A Pattern Matching Compiler for Multiple Target Languages », in *12th Conference on Compiler Construction* (G. Hedin, éd.), vol. 2622, 2003, p. 61–76.
- [Ouv99] OUVRAGE COLLECTIF – *Vérification de logiciels. techniques et outils du model-checking. ouvrage collectif*, Vuibert, 1999.
- [Pey03] S. PEYRONNET – « Model checking et vérification probabiliste », Thèse, LRI, 2003.
- [PRI] « PRISM web site » – www.cs.bham.ac.uk/~dxdp/prism.
- [Put94] M. PUTERMAN – *Markov decision processes - discrete stochastic dynamic programming*, Wiley series in probability and mathematical statistics, John Wiley & Sons, 1994.
- [PZ86] A. PNUELI & L. D. ZUCK – « Verification of multiprocess probabilistic protocols », *Distributed Computing* **1** (1986), no. 1, p. 53–72.
- [QS82] J. QUEILLE & J. SIFAKIS – « Specification and verification of programs in CESAR », in *5th International symposium on programming*, Lecture Notes in Computer Science, vol. 137, 1982.
- [Rei06] A. REILLES – « Réécriture et compilation de confiance », Thèse, Institut National Polytechnique de Lorraine, 2006.
- [Ros83] S. M. ROSS – *Stochastic process*, Wiley, John and sons, 1983.
- [Seg95] R. SEGALA – « Modelling and verification of randomized distributed real time systems », Thèse, Massachusetts Institute of Technology, 1995.
- [SL95] R. SEGALA & N. LYNCH – « Probabilistic simulations for probabilistic process », in *Nordic Journal of Computing*, vol. 2, 1995, p. 250–273.
- [Toy87] Y. TOYAMA – « Counterexamples to termination for the direct sum of term rewriting systems. », in *Information Processing Letters*, vol. 25, 1987, p. 141–143.

-
- [Var85] M. VARDI – « Automatic verification of concurrent finite-state programs », in *26th Annual Symposium on Foundations of Computer Science*, 1985, p. 327–338.
- [VM98] G. V.A.MALYSHEV & M. MENSHIKOV – *Topics in the constructive theory of countable Markov chains*, Cambridge University Press, 1998.

AUTORISATION DE SOUTENANCE DE THESE
DU DOCTORAT DE L'INSTITUT NATIONAL
POLYTECHNIQUE DE LORRAINE

o0o

VU LES RAPPORTS ETABLIS PAR :

Madame Catuscia PALAMIDESSI, Directeur de Recherche, INRIA, Palaiseau

Monsieur Laurent FRIBOURG, Directeur de Recherche, Ecole Normale Supérieure de Cachan, Cachan

Le Président de l'Institut National Polytechnique de Lorraine, autorise :

Monsieur GARNIER Florent

à soutenir devant un jury de l'INSTITUT NATIONAL POLYTECHNIQUE DE LORRAINE,
une thèse intitulée :

"Terminaison en temps moyen fini de systèmes de règles probabilistes"

NANCY BRABOIS
2, AVENUE DE LA
FORET-DE-HAYE
BOITE POSTALE 3
F - 5 4 5 0 1
VANDŒUVRE CEDEX

en vue de l'obtention du titre de :

DOCTEUR DE L'INSTITUT NATIONAL POLYTECHNIQUE DE LORRAINE

Spécialité : « **Informatique** »

Fait à Vandoeuvre, le 06 septembre 2007

Le Président de l'I.N.P.L.,

F. LAURENT



Résumé

Nous avons dans cette thèse cherché à définir un formalisme simple pour pouvoir modéliser des systèmes où se combinent des phénomènes non-déterministes et des comportements aléatoires. Nous avons choisi d'étendre le formalisme de la réécriture pour lui permettre d'exprimer des phénomènes probabilistes, puis nous avons étudié la terminaison en temps moyen fini de ce modèle. Nous avons également présenté une notion de stratégie pour contrôler l'application des règles de réécriture probabilistes et nous présentons des critères généraux permettant d'identifier des classes de stratégies sous lesquelles les systèmes de réécriture probabilistes terminent en temps moyen fini. Afin de mettre en valeur notre formalisme et les méthodes de preuve de terminaison en temps moyen fini, nous avons modélisé un réseau de stations WIFI et nous montrons que toutes les stations parviennent à émettre leurs messages dans un temps moyen fini.

Mots clefs : Réécriture, stratégies, terminaison, terminaison en temps moyen fini, terminaison presque sûre, terminaison presque sûre positive, choix non-déterministes, choix aléatoires, réécriture probabiliste, méthodes formelles.

Abstract

In this thesis we define a new formalism that allows to model transition systems where transitions can be either probabilistic or non deterministic. We choose to extend the rewriting formalism because it allows to simply express non-deterministic behavior. Latter, we study the termination of such systems and we give some criteria that imply the termination within a finite mean number of rewrite steps. We also study the termination of such systems when the firing of probabilistic rules are controlled by strategies. In this document, we use our techniques to model the WIFI protocol and show that a pool of stations successfully emits all its messages within a finite mean time.

Keywords: Rewriting, strategies, termination, bounded mean time termination, positive almost sure termination, non-determinism, random choices, probabilistic rewriting, formal methods.