

وزارة التعليم العالي و البحث العلمي

BADJI MOKHTAR ANNABA UNIVERSITY

Université Badji Mokhtar- Annaba



جامعة باجي مختار - عنابة

Faculté des Sciences de l'Ingéniorat  
Département d'Informatique

## THESE

Présentée en vue de l'obtention du diplôme de

**Doctorat en Sciences**

### Un environnement de model-checking pour les systèmes embarqués probabilistes

**Filière :** Informatique

**Spécialité :** Informatique Industrielle

Par :

**Assia FERROUM**

Devant le jury :

|            |                    |     |                                   |
|------------|--------------------|-----|-----------------------------------|
| Président  | : Kimour Med Tahar | Pr  | Université Badji Mokhtar - Annaba |
| Rapporteur | : Boudour Rachid   | Pr  | Université Badji Mokhtar - Annaba |
| Examineur  | : Redjimi Mohamed  | Pr  | Université 20 Août 1955 - Skikda  |
| Examineur  | : Bouras Zinedine  | MCA | ESTI - Annaba                     |

**Année :** 2017

Thèse préparée au laboratoire LASE, Université Badji Mokhtar - Annaba



# Remerciements

Louange à mon Seigneur *ALLAH* qui m'a doté de la merveilleuse faculté de raisonnement. Louange à mon Créateur qui m'a incité à acquérir le savoir. C'est à mon Dieu que j'adresse toute ma gratitude en premier lieu.

En second lieu, je tiens à exprimer toute ma reconnaissance et mon profond respect à mon encadreur, Pr *BOUDOUR Rachid* qui a toujours été disponible malgré ses nombreuses occupations, et qui m'a initié au model-checking, domaine très intéressant qui regroupe beaucoup de concepts et de théories avec lesquels j'ai toujours aimé travailler, en mathématique et en informatique théorique.

Je voudrais remercier Pr *Kimour Med Tahar* de m'avoir fait l'honneur de présider mon jury. Je veux également exprimer toute ma gratitude au Pr *Redjimi Mohamed* et au Dr *Bouras Zinedine* pour avoir accepté d'être examinateurs de cette thèse.

Je ne saurais terminer ces remerciements sans un énorme merci à tous les membres de ma famille qui m'ont apporté leur soutien moral tout au long de ce travail.

Enfin, je tiens à remercier toutes les personnes qui m'ont aidé de près ou de loin afin d'achever ce travail.

À tous un sincère et chaleureux merci !

# Dédicaces



*Je remercie Dieu tout puissant de m'avoir donné le courage et la force pour accomplir ce travail que je dédie :*

*À toute ma famille.*

*À tous mes enseignants depuis le primaire jusqu'à l'université ;*

*À toutes mes amies et mes collègues sans exception.*

*Assia*

---

# ملخص

التحقق النموذجي الحتمي لديه نماذج، خوارزميات وأدوات ضد الفضاءات الكبيرة جدا. و من ثم يمكن التحقق من خاصية منطقية على فضاء حالات كبير جدا لاختبار أنظمة بحجمها الحقيقي. وهذا ليس هو الحال بالنسبة للنظم الاحتمالية. الهدف المحدد في هذه الأطروحة هو إثراء بيئة النظام الاحتمالي من خلال الوسائل التقنية التي تسهل النمذجة والصياغة والتحقق من صحة خصائص الأنظمة المدمجة. وبالتالي فإن العمل المطروح في هذه الرسالة يقترح بعض التمديدات في هذا الاتجاه: نموذج HMM كنموذج جديد في التحقق الاحتمالي، ثم برمجة خوارزمية المنطق POCTL من أجل التحقق من صحة الخصائص على هذا النموذج، ثم تصميم وبرمجة في Netbeans الأداة الاحتمالية المسماة «HMM\_Model-Checker»، وأخيرا انجاز أرضية مفتوحة للتحقق الاحتمالي وذلك بدمج مجموعة من أفضل الأدوات المعروفة لاقتراح إلى المستخدم الأداة المناسبة بأقل جهد. وقد تمت تجربة هذه الأرضية على حالة حقيقية على التليسكوب HST ونتائج واعدة تم الحصول عليها مقارنة بالدراسات الموجودة.

**كلمات مفتاحية:** التحقق الاحتمالي للنظم، أنظمة مدمجة، HMM، POCTL، HST.

---

# Résumé

Le model checking déterministe dispose de modèles, d'algorithmes et d'outils efficaces contre l'explosion combinatoire des états. Il est alors envisageable de vérifier une propriété logique sur un espace d'états très grand pour tester des systèmes en vraie grandeur. Cela n'est pas encore le cas pour des systèmes probabilistes. L'objectif défini dans cette thèse est d'enrichir l'écosystème probabiliste par des moyens techniques rendant la modélisation, la formulation et la validation des propriétés des systèmes embarqués plus aisée. Ainsi le travail véhiculé dans cette thèse, propose quelques extensions dans cette direction : le modèle de Markov caché (HMM) comme nouveau modèle en vérification probabiliste, puis une implémentation d'un algorithme pour POCTL afin de permettre la vérification des propriétés sur ce modèle HMM, ensuite une conception et une implémentation dans l'environnement Netbeans d'un model checker probabiliste baptisé «HMM\_Model-Checker», enfin une réalisation d'un environnement de vérification probabiliste ouvert intégrant une panoplie de model-checkers des plus connus pour suggérer à l'utilisateur l'outil approprié à moindres efforts. Cet environnement a été expérimenté sur un cas réel le télescope spatial Hubble HST et les résultats obtenus comparables à l'état de l'art sont prometteurs.

**Mots clés :** Model checker probabiliste, Systèmes embarqués, HMM, POCTL, HST.

---

# Abstract

The deterministic model checking has efficient models, algorithms and tools against the combinatorial explosion of states. It is then possible to verify a logical property on a very large state space to test systems in full size. This is not yet the case for probabilistic systems. The objective defined in this thesis is to enrich the probabilistic ecosystem by technical means making it easier to model, formulate and validate the properties of embedded systems. Thus the work conveyed in this thesis proposes some extensions in this direction: the hidden Markov model (HMM) as a new model in probabilistic verification, then an implementation of an algorithm for POCTL in order to allow the verification of the properties on this model HMM , then a design and implementation in the Netbeans environment of a probabilistic model checker named "HMM\_Model-Checker", finally a realization of an open probabilistic verification environment integrating a set of best-known model-checkers to suggest to the user the right tool for less effort. This environment has been experimented on a real case Hubble Space Telescope HST and the results obtained comparable to the state of the art are promising.

**Key words:** Probabilistic model checking, Embedded systems, HMM, POCTL, HST.

---

# Table des matières

|                                                          |               |
|----------------------------------------------------------|---------------|
| Remerciements.....                                       | i             |
| Dédicaces.....                                           | ii            |
| ملخص.....                                                | iii           |
| Résumé.....                                              | iv            |
| Abstract.....                                            | v             |
| Table des matières.....                                  | vi            |
| Liste des tableaux.....                                  | x             |
| Table des figures.....                                   | xi            |
| <br><b>Chapitre 1 : Introduction Générale.....</b>       | <br><b>13</b> |
| 1.1. Motivations.....                                    | 14            |
| 1.2. Les méthodes formelles.....                         | 15            |
| 1.3. Model-checking.....                                 | 16            |
| 1.3.1. Processus du Model-checking.....                  | 17            |
| 1.3.2. Forces et faiblesses.....                         | 17            |
| 1.3.3. Applications .....                                | 18            |
| 1.4. Objectifs.....                                      | 19            |
| 1.5. Contributions.....                                  | 20            |
| 1.6. Structure de la thèse.....                          | 20            |
| <br><b>Première Partie : État de l'art.....</b>          | <br><b>22</b> |
| <br><b>Chapitre 2 : Model-checking probabiliste.....</b> | <br><b>23</b> |
| 2.1. Introduction.....                                   | 24            |
| 2.2. Formalismes de modélisation .....                   | 24            |
| 2.2.1. Chaînes de Markov à temps discret .....           | 25            |

---

|                                                                                       |               |
|---------------------------------------------------------------------------------------|---------------|
| 2.2.2. Processus de décision de Markov .....                                          | 25            |
| 2.2.3 Chaînes de Markov à temps continu.....                                          | 26            |
| 2.2.4. Chaînes de Markov Cachées.....                                                 | 27            |
| 2.3. Formalismes de spécification.....                                                | 30            |
| 2.3.1. Quelques propriétés.....                                                       | 31            |
| 2.3.2. La logique PCTL.....                                                           | 31            |
| 2.3.2.1. La syntaxe de PCTL.....                                                      | 31            |
| 2.3.2.2. La sémantique de PCTL.....                                                   | 33            |
| 2.3.3. La logique POCTL.....                                                          | 34            |
| 2.3.3.1. La syntaxe de POCTL.....                                                     | 34            |
| 2.3.3.2. La sémantique de POCTL.....                                                  | 35            |
| 2.4. Algorithmes de model-checking probabiliste.....                                  | 36            |
| 2.5. Conclusion.....                                                                  | 39            |
| <br><b>Chapitre 3 : Outils de model-checking probabiliste.....</b>                    | <br><b>40</b> |
| 3.1. Introduction.....                                                                | 41            |
| 3.2. Model-checkers probabilistes.....                                                | 41            |
| 3.2.1. PRISM.....                                                                     | 42            |
| 3.2.2. APMC.....                                                                      | 42            |
| 3.2.3. ETMCC.....                                                                     | 43            |
| 3.2.4. MRMC.....                                                                      | 43            |
| 3.2.5. YMER.....                                                                      | 43            |
| 3.2.6. VESTA.....                                                                     | 44            |
| 3.2.7. UPPAAL – PRO.....                                                              | 44            |
| 3.3. Comparaisons.....                                                                | 45            |
| 3.4. Conclusion.....                                                                  | 49            |
| <br><b>Deuxième partie : Contributions.....</b>                                       | <br><b>50</b> |
| <br><b>Chapitre 4 : Contribution1: L’outil de vérification HMM_Model-Checker.....</b> | <br><b>51</b> |
| 4.1. Introduction.....                                                                | 52            |

---

---

|                                                         |    |
|---------------------------------------------------------|----|
| 4.2. Contexte de notre approche.....                    | 52 |
| 4.3. Modélisation.....                                  | 53 |
| 4.4. Spécification des propriétés.....                  | 54 |
| 4.4.1. Identification d'un ensemble d'états.....        | 54 |
| 4.4.2. Syntaxe de base.....                             | 55 |
| 4.4.3. L'opérateur P.....                               | 56 |
| 4.4.4. Les propriétés de chemins.....                   | 57 |
| 4.4.5. Les propriétés de chemins « next ».....          | 57 |
| 4.4.6. Les propriétés de chemins « Until ».....         | 58 |
| 4.4.7. Les propriétés de chemins « Bounded Until »..... | 58 |
| 4.4.8. Spécification des propriétés.....                | 59 |
| 4.5. Model-checking pour POCTL sur un HMM.....          | 59 |
| 4.5.1. Le calcul de $P_s(\psi)$ .....                   | 61 |
| 4.5.2. Exemples explicatifs.....                        | 65 |
| 4.6. Vue d'ensemble sur HMM_Model-Checker.....          | 73 |
| 4.7. Architecture du HMM_Model-Checker.....             | 73 |
| 4.8. Environnement de développement.....                | 74 |
| 4.9. Matériel utilisé.....                              | 75 |
| 4.10. Description générale du HMM_Model-Checker.....    | 75 |
| 4.11. Présentation du HMM_Model-Checker.....            | 78 |
| 4.11.1. L'interface graphique.....                      | 78 |
| 4.11.2. Fonctionnement du système.....                  | 79 |
| 4.12. Conclusion.....                                   | 91 |

## **Chapitre 5 : Contributions 2 et 3 : L'environnement de vérification avec Étude de cas.92**

|                                             |    |
|---------------------------------------------|----|
| 5.1. Introduction.....                      | 93 |
| 5.2. Cadre conceptuel du système.....       | 93 |
| 5.3. Architecture du système.....           | 93 |
| 5.4. Le langage de modélisation UML.....    | 95 |
| 5.4.1. Diagramme des Cas d'utilisation..... | 95 |
| 5.4.2. Diagramme d'activité.....            | 95 |
| 5.4.3. Diagramme de séquence.....           | 98 |

---

|                                                        |         |
|--------------------------------------------------------|---------|
| 5.5. Environnement du développement.....               | 99      |
| 5.6. Description générale de l'environnement.....      | 100     |
| 5.7. Présentation de l'environnement.....              | 100     |
| 5.8. Les interfaces graphiques.....                    | 101     |
| 5.9. Étude de cas.....                                 | 104     |
| 5.9.1. Le télescope spatial Hubble.....                | 104     |
| 5.9.2. Le Système réel.....                            | 105     |
| 5.9.3. Modélisation avec le modèle HMM.....            | 106     |
| 5.9.4. Choix des propriétés à vérifier.....            | 110     |
| 5.9.5. Vérification des propriétés.....                | 111     |
| 5.9.6. Résultats et discussions.....                   | 120     |
| 5.10. Conclusion.....                                  | 121     |
| <br><b>Conclusion Générale &amp; Perspectives.....</b> | <br>123 |
| <br>6.1. Conclusion.....                               | <br>124 |
| 6.2. Perspectives.....                                 | 126     |
| <br><b>Glossaire.....</b>                              | <br>127 |
| <br><b>Références Bibliographiques.....</b>            | <br>129 |

---

# Liste des tableaux

|                                                         |    |
|---------------------------------------------------------|----|
| Tableau 3.1- Vue d'ensemble d'outils.....               | 45 |
| Tableau 3.2- Plateformes supportées par les outils..... | 46 |
| Tableau 3.3- Modèles supportés par les outils.....      | 46 |
| Tableau 3.4- Logiques supportés par les outils.....     | 47 |
| Tableau 3.5- Structures de données des outils.....      | 47 |
| Tableau 3.6- Algorithmes utilisés par les outils.....   | 48 |
| Tableau 4.1- Caractéristiques du matériel utilisé.....  | 75 |

---

# Table des figures

|                                                                                   |    |
|-----------------------------------------------------------------------------------|----|
| Figure 2.1- Mise à jour des états de croyance.....                                | 29 |
| Figure 2.2- La syntaxe de la logique PCTL.....                                    | 32 |
| Figure 2.3- La sémantique de PCTL.....                                            | 33 |
| Figure 2.4- La syntaxe de la logique POCTL.....                                   | 34 |
| Figure 2.5- La sémantique de POCTL.....                                           | 35 |
| Figure 4.1- Algorithme de Model checking pour POCTL sur HMM.....                  | 60 |
| Figure 4.2- Algorithme Prob0.....                                                 | 62 |
| Figure 4.3- Algorithme Prob1.....                                                 | 63 |
| Figure 4.4- Le modèle HMM de l'exemple explicatif 1.....                          | 65 |
| Figure 4.5- La matrice P de l'exemple explicatif 1.....                           | 66 |
| Figure 4.6- La matrice L de l'exemple explicatif 1.....                           | 66 |
| Figure 4.7- La matrice B de l'exemple explicatif 1.....                           | 66 |
| Figure 4.8- Le vecteur $\pi$ de l'exemple explicatif 1.....                       | 67 |
| Figure 4.9- Le modèle HMM de l'exemple explicatif 2.....                          | 69 |
| Figure 4.10- La matrice P de l'exemple explicatif 2.....                          | 70 |
| Figure 4.11- La matrice L de l'exemple explicatif 2.....                          | 70 |
| Figure 4.12- La matrice B de l'exemple explicatif 2.....                          | 71 |
| Figure 4.13- Le vecteur $\pi$ de l'exemple explicatif 2.....                      | 71 |
| Figure 4.14- Vue d'ensemble de HMM_Model-Checker.....                             | 73 |
| Figure 4.15- Architecture du HMM_Model-Checker.....                               | 74 |
| Figure 4.16- Schéma de développement du HMM_Model-Checker.....                    | 77 |
| Figure 4.17- L'interface graphique.....                                           | 78 |
| Figure 4.18- Barre des menus.....                                                 | 79 |
| Figure 4.19- Barre d'outils.....                                                  | 79 |
| Figure 4.20- Le menu Fichier.....                                                 | 79 |
| Figure 4.21- Le menu Édition.....                                                 | 80 |
| Figure 4.22- Le menu Modèle.....                                                  | 80 |
| Figure 4.23- Le sous menu HMM.....                                                | 80 |
| Figure 4.24- La fenêtre Ouvrir HMM.....                                           | 81 |
| Figure 4.25- La boîte de dialogue pour choisir les paramètres du nouveau HMM..... | 81 |

---

|                                                                                          |     |
|------------------------------------------------------------------------------------------|-----|
| Figure 4.26- Exemple d'un nouveau HMM avec quatre états.....                             | 82  |
| Figure 4.27- La boîte de dialogue donnant les informations associées à un état.....      | 83  |
| Figure 4.28- La fenêtre Sauvegarder HMM.....                                             | 84  |
| Figure 4.29- Le sous menu Séquences-observations.....                                    | 84  |
| Figure 4.30- La boîte de dialogue pour choisir le type des observations.....             | 85  |
| Figure 4.31- La fenêtre Ouvrir Séquences.....                                            | 85  |
| Figure 4.32- La fenêtre Séquences générées par nouveau HMM.....                          | 86  |
| Figure 4.33- Message donnant la probabilité d'une séquence d'observations du HMM.....    | 86  |
| Figure 4.34- La fenêtre Sauvegarder Séquences.....                                       | 87  |
| Figure 4.35- Le sous menu Séquences-état.....                                            | 87  |
| Figure 4.36- La PPSE des séquences générées par nouveau HMM[0].....                      | 88  |
| Figure 4.37- Le menu Propriété.....                                                      | 88  |
| Figure 4.38- La fenêtre de vérification d'une formule POCTL.....                         | 89  |
| Figure 4.39- La fenêtre « Aide ».....                                                    | 90  |
| Figure 5.1- Architecture de notre environnement de vérification.....                     | 94  |
| Figure 5.2- Diagramme des Cas d'utilisation.....                                         | 96  |
| Figure 5.3 : diagramme d'activité.....                                                   | 97  |
| Figure 5.4- Diagramme de séquence (utilisateur).....                                     | 98  |
| Figure 5.5- Diagramme de séquence (Administrateur).....                                  | 99  |
| Figure 5.6- L'interface Home.....                                                        | 101 |
| Figure 5.7- L'interface de recherche d'un vérificateur.....                              | 101 |
| Figure 5.8- L'interface de recherche d'un vérificateur (afficher modèle et logique)..... | 102 |
| Figure 5.9- L'interface de recherche d'un vérificateur (résultat de recherche).....      | 102 |
| Figure 5.10- L'interface des outils.....                                                 | 103 |
| Figure 5.11- L'interface de M.A.J.....                                                   | 103 |
| Figure 5.12- Le modèle HMM du télescope spatial Hubble (HST).....                        | 106 |
| Figure 5.13- La matrice L du HMM du télescope HST.....                                   | 108 |
| Figure 5.14- La matrice P du HMM du HST.....                                             | 109 |
| Figure 5.15- La matrice $\mu$ du HMM correspondante.....                                 | 109 |
| Figure 5.16- Le vecteur $\pi$ du HMM du HST.....                                         | 110 |
| Figure 5.17- Résultat de la vérification de la propriété 1 en utilisant notre outil..... | 118 |
| Figure 5.18- Résultat de la vérification de la propriété 2 en utilisant notre outil..... | 119 |
| Figure 5.19- Résultat de la vérification de la propriété 3 en utilisant notre outil..... | 120 |

---

---

# Chapitre 1

## Introduction Générale

---

## 1.1. Motivations

Les systèmes embarqués sont de plus en plus utilisés dans de nombreux domaines d'application : l'industrie des télécommunications, l'électronique grand public, l'automobile et l'aéronautique, ... Ils font partie de notre vie quotidienne via internet ou via toutes sortes de systèmes intégrés comme les appareils électroménagers, les smartphones, les téléviseurs intelligents, les montres, etc.

Il est important de s'assurer de la sûreté de fonctionnement de tous ces systèmes avant de les mettre en œuvre, en particulier les applications mettant en danger la santé publique. Dans ces applications, outre leur dangerosité en cas de mauvais fonctionnement, elles impliquent un préjudice économique important, difficile à supporter parfois pour les personnes voire les états.

Dans un système embarqué complexe, non seulement la correction fonctionnelle, la vivacité et la synchronisation sont pertinentes, mais aussi des propriétés quantitatives comme la consommation de ressources (par exemple la puissance), la fiabilité (par exemple, les messages perdus, l'espérance de vie) et la performance (par exemple, le débit) [1].

La liste des catastrophes connues de tels systèmes est très longue. Un des exemples célèbres de systèmes critiques aux conséquences néfastes est la machine de radiothérapie Therac-25 développée conjointement par l'Atomic Energy of Canada Limited (AECL) et par la société française CGR MeV. Entre 1985 et 1987, la machine Therac-25 fut impliquée dans six accidents durant lesquels des patients reçurent des doses massives de radiation. Au moins cinq patients décédèrent des suites de l'irradiation. Une erreur dans la programmation du logiciel conduisait dans certaines situations à utiliser erronément la machine dans un mode Rayon-X au lieu du mode faisceau d'électrons, ce qui envoyait 125 fois plus de radiations que la dose prescrite par le médecin. Les coûts peuvent être catastrophiques en termes de vies humaines mais aussi en termes de coût économique et d'image de la marque. Un autre échec célèbre est le vol 501 inaugural de la fusée Ariane-5 qui a eu lieu le 4 juin 1996. La fusée a explosé 37 secondes après le décollage à la suite d'une panne du système de navigation. L'incident n'a fait aucune victime car la fusée n'était pas habitée. Les enquêtes ont montré que l'incident était dû à une erreur dans les appareils informatiques utilisés par le pilote automatique qui a

provoqué la destruction de la fusée ainsi que la charge utile c'est-à-dire 4 sondes de la mission Cluster [2].

Un troisième exemple de bogue est le rappel de 2.5 millions de véhicules par le constructeur automobile Honda en octobre 2011. Le rappel était dû à un problème informatique et électronique au niveau de la commande permettant d'actionner les vitres. La saleté qui s'accumulait sur les vitres endommageait les circuits et provoquait un incendie. L'erreur a coûté chère car il a fallu rappeler les véhicules défectueux. L'erreur a ainsi endommagé la réputation de la marque Honda [3].

Prenons plutôt un exemple beaucoup plus récent. Découvert il y a peu par des universitaires finlandais et rendu public par Google, un bogue appelé heartbleed permet de récupérer des informations confidentielles de nombreux services web connus tels que Yahoo, Flickr, Doodle ou encore Zimbra. Ce bogue se situe dans l'implémentation de OpenSSL, le protocole de sécurité le plus utilisé sur internet, et peut être exploité par n'importe qui. Les résultats d'audit semblent montrer que certains « attaquants » peuvent avoir exploité la faille pendant au moins cinq mois avant qu'elle ne soit officiellement découverte et corrigée en avril 2014 [4].

Pour pallier à ces problèmes, une solution en vogue est la vérification formelle, pour établir la conformité avec les exigences requises. Cette conformité est fréquemment abordée à l'aide de techniques telle que la vérification de modèles ou model-checking [5], objet de notre travail.

## 1.2. Les méthodes formelles

L'objectif de la vérification du système est d'établir si la conception ou le produit considéré satisfait certaines propriétés. La base de la vérification est une spécification qui prescrit ce que le système doit faire. Si le système satisfait à toutes les propriétés spécifiées, il est considéré comme correct, sinon il est défectueux. La correction est toujours relative à la spécification ; Ce n'est pas une propriété absolue d'un système. Lorsque nous disons qu'un système est correct, il est considéré comme correct pour une spécification [1].

Les méthodes formelles permettent de définir mathématiquement et rigoureusement les propriétés de fonctionnement d'un système. Elles définissent la syntaxe et la sémantique formelles du système. Elles s'appliquent à n'importe quelle phase du cycle de développement

d'un logiciel. L'utilisation de la logique mathématique et la rigueur de ces méthodes permettent de garantir l'impossibilité de comportements indésirables dans un système ou de garantir l'existence de comportements souhaitables [6].

Au cours de la dernière décennie, la recherche poussée dans le domaine des méthodes formelles a pris une place très importante conduisant ainsi au développement de plusieurs techniques de vérification très puissantes. On distingue trois classes de techniques de vérification [7] :

- Les techniques basées sur la preuve de théorème qui utilisent un modèle mathématique du système. Le principe d'une telle technique consiste à déterminer un ensemble d'axiomes (théorème de base), et étudier la capacité du modèle mathématique (démonstrateur) à construire une preuve de théorème.
- Le model checking est la technique de vérification la plus utilisée. Elle a été appliquée avec succès dans plusieurs domaines notamment le développement des protocoles de communication.
- La technique de synthèse de contrôleur consiste à déterminer un programme dont la synchronisation avec un système non totalement spécifié, vérifie les propriétés attendues du système.

Dans ce travail, nous nous intéressons au model-checking, et plus précisément à l'utilisation de cette technique pour la vérification de systèmes probabilistes.

### 1.3. Model-checking

Le model-checking [8] vérifie qu'un modèle satisfait une propriété exprimée dans différents formalismes. Le système est décrit par un modèle d'état fini et les propriétés sont exprimées en logique temporelle. La vérification des modèles vérifie formellement si le modèle satisfait une propriété par une recherche exhaustive automatique sur l'espace d'états. Un vérificateur de modèle est un outil logiciel qui, selon la description d'un modèle  $M$  et d'une propriété  $\emptyset$ , décide si  $M$  satisfait  $\emptyset$ . Le vérificateur modèle retourne Vrai si la propriété est satisfaite, sinon

il renvoie Faux, et fournit un contre-exemple. Il peut également fournir un témoin lorsque la propriété est satisfaite [9].

### 1.3.1 Processus du model-checking

En pratique, le model checking est un processus en trois phases :

1. Modéliser le système (système de transitions  $M$ ) et les spécifications (formule temporelle  $\emptyset$ ).
2. Vérifier si  $M$  satisfait  $\emptyset$  ou non. Sinon, retourner un contre-exemple.
3. Analyser les résultats obtenus :
  - (a) Si oui, le modèle  $M$  est sûr. Fin. Attention : le système réel est-il sûr pour autant ?
  - (b) Sinon, rejouer le contre-exemple sur le système réel.
    - i. Si c'est un vrai bug, avertir le concepteur et attendre qu'il corrige. Fin.
    - ii. Si le bug vient de notre modélisation, repartir à (1) en raffinant le modèle grâce au faux bug.

### 1.3.2 Forces et faiblesses

Parmi les avantages du model-checking, nous pouvons citer : [3]

- Il se base sur des fondements mathématiques solides comme la théorie algorithmique des graphes, les structures de données et la logique formelle.
- Il est facilement intégrable durant les cycles de développement d'un logiciel (phase d'analyse, de développement ou de test) et il permet aussi la réduction du temps de développement.
- Il fournit une information de diagnostic dans le cas où une propriété n'est pas vérifiée ; ce qui peut être utile pour déboguer.

- Il jouit d'un intérêt croissant du milieu industriel. Plusieurs compagnies dans le domaine du matériel informatique, du logiciel et des services informatiques comme la compagnie Bell et IBM ont désormais des laboratoires où le model-checking est utilisé.

Parmi les faiblesses du model-checking, nous pouvons citer :

- Le model-checking vérifie un modèle de système et non pas le système réel ou son prototype. Le résultat dépend par conséquent de la qualité du modèle.
- Le model-checking souffre du problème d'explosion combinatoire : le nombre de chemins à explorer pour analyser avec précision le comportement d'un système peut devenir rapidement grand et par conséquent impossible à parcourir.
- Le model-checking ne garantit pas un résultat exact dans le cas probabiliste. Dans le cas classique (model-checking non probabiliste), on a un résultat exact sauf s'il y a des fautes de programmation du model checker. Dans le cas probabiliste, nous devons représenter les probabilités sous forme de nombres réels en utilisant l'arithmétique à point flottant utilisée par les ordinateurs actuels, telle que décrite par la norme IEEE 754. L'arithmétique à virgule flottante ne permettant de représenter les nombres réels que jusqu'à une certaine précision (15 chiffres après la virgule), un nombre qui ne peut être représenté avec cette précision est arrondi. La conversion d'un réel en un nombre représentable génère de l'instabilité numérique et peut avoir un impact sur les probabilités calculées par le model-checker.

### 1.3.3. Applications

La technique du model-checking a été largement appliquée dans plusieurs domaines. Son intérêt a été particulièrement bien montré pour la vérification des protocoles de communication, les applications multimédia, et la sécurité informatique. Le model-checking a pour avantage de fournir un contre exemple lorsqu'il détecte une violation de la propriété ce qui facilite la compréhension et la correction des erreurs. Dans la littérature, plusieurs travaux montrent la polyvalence et l'apport indéniable des techniques de vérification dans la conception de systèmes. En effet, à l'aide de cette technique, il a été prouvé l'exactitude d'une

variété de protocoles de logiciels, notamment, des protocoles modernes d'e-commerce. Cette technique a également permis de détecter des bugs dans les systèmes de réservation en ligne des compagnies aériennes. Elle a été utilisée également pour vérifier un module d'exécution du contrôleur de la sonde spatiale "Deep Space 1" en identifiant un défaut de conception majeur. Ce défaut n'a pas été détecté pendant la phase de test et a causé un blocage lors d'un vol expérimental de la sonde à 96 millions de kilomètres au-dessus de la Terre. [7]

## 1.4. Objectifs

L'objectif principal de notre thèse consiste à aller à un environnement de vérification des systèmes probabilistes. Notre objectif principal se subdivise en trois sous-objectifs :

- Introduire pour la première fois le formalisme mathématique « HMM » dans le domaine de la vérification probabiliste à travers la réalisation d'un outil qui le supporte, car d'une part, les HMMs sont des outils statistiques d'apprentissage simple, riches en propriétés, fondés sur des bases mathématiques solides et ils permettent de modéliser divers systèmes probabilistes à partir d'un ensemble d'observations et dont le nombre d'états du système peut ne pas être connu. Et d'autre part, leur utilisation dans le domaine informatique est très répandue.
- Enrichir les model-checkers probabilistes existants par un outil dédié au modèle HMM comme un nouveau formalisme de modélisation et de la logique POCTL, comme un nouveau formalisme de spécification. A notre connaissance, il n'existe aucun model-checker probabiliste connu utilisant le HMM et le POCTL.
- Concevoir et implémenter un environnement de model-checking probabiliste qui rassemble tous les formalismes de modélisation existants.
- La réalisation des études de cas sur des systèmes embarqués réels pour valider notre travail.

## 1.5. Contributions

Les contributions apportées dans le cadre de cette thèse peuvent se résumer comme suit : (1) Une conception et une implémentation dans l'environnement Netbeans de l'outil appelé « HMM\_Model-Checker » pour permettre la vérification des propriétés sur le modèle HMM. (2) Une conception et une implémentation d'un environnement de vérification des systèmes probabilistes. (3) Ainsi qu'une étude de cas d'un système embarqué réel qui est le télescope spatial *Hubble* (en anglais, *Hubble Space Telescope* ou *HST*).

## 1.6. Structure de la thèse

Nous avons structuré cette thèse en deux parties.

La première partie concernant l'état de l'art est composée de deux chapitres :

Le chapitre 2 intitulé « Model-checking probabiliste » dresse un état de l'art relatif au model-checking probabiliste. Cette technique nous semble offrir, un très bon compromis entre : la couverture des différentes exécutions du système, la vérification proprement dite, le haut niveau de compétence requis pour son utilisation, et la simplicité de sa mise en œuvre. Nous avons présenté aussi dans ce chapitre les différents formalismes de modélisation probabiliste les plus connus : DTMC, CTMC et MDP ainsi que le modèle adopté dans notre approche HMM, ainsi que les logiques probabilistes : PCTL et POCTL.

Le chapitre 3 fait un survol de sept outils de model-checking probabiliste, les plus connus à ce jour.

La deuxième partie s'intéresse à nos contributions, est divisée en deux chapitres :

Le chapitre 4 intitulé « Contribution 1: L'outil de vérification HMM\_Model-Checker » est dédié à la conception de notre outil HMM\_Model-Checker : son architecture, l'étape de modélisation à base de HMM, la spécification des propriétés en logique POCTL, ainsi que la présentation d'algorithme de model-checking pour POCTL, suivies des détails d'implémentation dans l'environnement Netbeans.

Le chapitre 5 intitulé « Contributions 2 et 3 : L’environnement de vérification avec étude de cas » est dédié à la conception de notre environnement avec ses détails d’implémentation, suivies des résultats expérimentaux obtenus à partir d’une étude de cas d’un système embarqué réel qui est le télescope spatial Hubble (HST) : la présentation du système HST, la construction de son modèle HMM, le choix des propriétés à vérifier, la spécification de ces propriétés choisies en logique POCTL, la vérification manuelle et automatique des propriétés choisies et en fin comparaisons de nos résultats.

Une conclusion achève cette thèse, mettant en avant les principaux aspects de notre travail avec des perspectives de nos futurs travaux.

---

# **Première Partie**

## **État de l'art**

---

---

# **Chapitre 2**

## **Model-checking probabiliste**

---

## 2.1. Introduction

La vérification formelle basée sur la logique temporelle a été prolongée avec succès au domaine des modèles probabilistes. Dans ce chapitre, nous introduisons les notions de base associées au model-checking probabiliste.

Le model checking probabiliste est une technique de vérification formelle automatique pour l'analyse de systèmes qui présentent un comportement stochastique. La technique est similaire au model-checking classique. La différence majeure est que le modèle probabiliste contient des informations supplémentaires sur la probabilité ou le calendrier des transitions entre états, ou pour être plus précis, il peut modéliser le comportement stochastique.

Ce chapitre donne une introduction au model-checking probabiliste, il élabore des concepts de base pertinents pour cette thèse, à commencer par les principaux modèles probabilistes utilisés par la plupart des model checkers existants à savoir : les chaînes de Markov à temps discret (*Discrete-Time Markov Chains* ou DTMC), les processus de décision de Markov (*Markov Decision Processes* ou MDP) ainsi que les chaînes de Markov à temps continu (*Continuous-Time Markov Chains* ou CTMC), avec la logique PCTL (*Probabilistic Computation Tree Logic*) utilisée pour la spécification des propriétés pour les DTMC et les MDP.

Nous présentons aussi dans ce chapitre, le modèle probabiliste HMM que nous allons introduire pour la première fois au domaine de la vérification probabiliste dans ce travail, à travers la réalisation d'un model checker le supportant (voir chapitre 4) et la logique temporelle probabiliste correspondant POCTL (*Branching-time temporal logic Probabilistic Observation*) qui est une prolongation de la logique PCTL, et finalement une vue générale sur les algorithmes de model-checking probabiliste.

## 2.2. Formalismes de modélisation

Les types de modèles stochastiques les plus utilisés dans les model-checkers probabilistes les plus connus (chapitre 3) sont : (DTMC) [8, 10], (MDP) [10] et

(CTMC) [10]. Nous présentons aussi dans cette section le type que nous allons adopter dans notre approche qui est le (HMM) [11].

### 2.2.1. Chaînes de Markov à temps discret

Les chaînes de Markov à temps discret [8, 10] ou DTMC (de l'anglais *Discrete-Time Markov Chain*) sont le modèle le plus simple représentant des systèmes probabilistes. Ce sont des systèmes de transitions où les transitions sont étiquetées par la probabilité de passer d'un état à un autre.

**Définition :** Soit  $AP$  (de l'anglais *Atomic Proposition*) un ensemble fixe de propositions atomiques. Une chaîne de Markov à temps-discret est un tuple  $M = (S, s_0, P, L)$  tel que :

- $S$  est un ensemble fini non vide d'états,
- $s_0 \in S$  est l'état initial,
- $P : S \times S \rightarrow [0, 1]$  est une fonction qui assigne une probabilité à chaque transition de telle sorte que pour tout  $s \in S$ , on a :

$$\sum_{s' \in S} P(s, s') \in \{0, 1\}$$

- $L : S \rightarrow 2^{AP}$ , est une fonction qui assigne à chaque état  $s \in S$ , un ensemble de propositions atomiques  $L(s)$  valides dans  $s$ .

### 2.2.2. Processus de décision de Markov

Les DTMC ne permettent pas de modéliser le non-déterminisme. Les processus de décision de Markov [10] sont une généralisation des DTMC qui permettent à la fois un choix probabiliste et non déterministe.

Comme pour les DTMC, le choix probabiliste sert à modéliser et quantifier les sorties possibles d'une action aléatoire tandis que le non-déterminisme permet de décrire par exemple des systèmes concurrents, i.e., des systèmes fonctionnant en parallèle.

**Définition :** Soit  $AP$  un ensemble fixe de propositions atomiques. Un processus de décision de Markov ou MDP (de l'anglais *Markov Decision Process*) est un tuple  $M = (S, l_{init}, Act, P, L)$  tel que :

- $S$  est un ensemble fini non vide d'états,
- $l_{init} : S \rightarrow [0; 1]$  est la distribution initiale telle que :

$$\sum_{s \in S} l_{init}(s) = 1$$

- $Act$  est un ensemble d'actions,  $P : S \times Act \times S \rightarrow [0, 1]$  est la fonction des transitions de probabilité telle que pour tout état  $s \in S$  et action  $\alpha \in Act$  :

$$\sum_{s' \in S} P(s, \alpha, s') \in \{0, 1\}$$

- $L : S \rightarrow 2^{AP}$ , une fonction d'étiquetage.

### 2.2.3. Chaînes de Markov à temps continu

Les chaînes de Markov à temps continu (*Continuous- Time Markov Chains* ou CTMC) [10], étend aussi les DTMC mais d'une manière différente. Pour les DTMC, une transition correspond à une étape discrète d'un point de vue temporel. Dans un CTMC, les transitions peuvent être exécutées en temps réel. Chaque transition est étiquetée d'un coût, définissant le délai qui s'écoule avant qu'elle ne soit exécutée.

**Définition :** Soit  $AP$  un ensemble de propositions atomiques. Une *chaîne de Markov à temps continu* est un tuple  $M = (S, s_0, R, L)$  où :

- $S$  est un ensemble fini d'états,
- $s_0$  est l'état initial,
- $R : S \times S \rightarrow R_{\geq 0}$  est la matrice des taux de transition,
- $L : S \rightarrow 2^{AP}$  est une fonction d'étiquetage.

### 2.2.4. Modèles de Markov Cachés

Les modèles de Markov cachés (*Hidden Markov Model* ou HMM) [11] ont été développés vers la fin des années 60 et sont très importants pour beaucoup d'applications, particulièrement la reconnaissance de la parole [12], la reconnaissance de caractères [13], l'analyse biologique [14], et les problèmes de classification de protéines [15]. Récemment, les HMMs suscitent une attention importante dans le contexte de la modélisation des canaux de communication « *communication channel modelling* » [16] et des propriétés QoS dans les réseaux sans fil [17, 18].

Un HMM [11] peut être vu comme un automate fini dont les transitions sont pondérées par des probabilités, dites probabilités de transition. De plus, chaque état de l'automate peut "émettre" des symboles d'un alphabet selon une loi de probabilité donnée que l'on appelle loi de probabilité d'émission. L'adjectif "caché" provient du fait qu'une suite de symboles générée par un HMM ne permet pas de savoir par quelle suite d'états le HMM est passé pour la générer, nous disons par conséquent que les états du HMM sont "cachés".

Un HMM est un processus doublement stochastique avec un processus sous-jacent qui n'est pas observable. Ce dernier est uniquement observé à travers un autre ensemble de processus stochastiques qui produit la séquence des symboles [11]. En donnant une séquence d'observations, nous ne pouvons pas connaître exactement l'état occupé mais nous allons connaître la distribution de probabilité sur l'ensemble des états, cette information est captée par ce que nous appelons état **de croyance** [18].

**Définition :** Soit  $AP$  un ensemble de propositions atomiques. Un *modèle de Markov caché* est un sextuplet  $H = (S, P, \theta, \mu, \pi, L)$  où :

- $S = \{s_1, s_2, \dots, s_n\}$  est un ensemble fini de  $n$  états. L'état courant du HMM à l'instant  $t$  est noté  $q_t$  ( $q_t \in S$ ) ;
- $P$  est une matrice de probabilités de transitions entre les états de la chaîne :  

$$\forall i, j \in [1 \dots n], \forall t \in [1 \dots T] : a_{ij} = P(i, j) = p(q_{t+1} = s_j \mid q_t = s_i) \text{ avec : } a_{ij} \geq 0, \forall i, j \text{ et } \sum_{j=1}^n a_{ij} = 1;$$
- $\theta = \{O_1, O_2, \dots, O_M\}$  est l'ensemble des  $M$  symboles observables dans chaque

- état. Un élément  $V_t$  de  $\theta$  désigne un symbole observé à l'instant  $t$  ;
- $\mu$  est la matrice de distribution des probabilités d'observations :  $\mu_j(k)$  est la probabilité d'observer le symbole  $O_k$  quand le modèle se trouve dans l'état  $j$ , soit :  $\mu_j(k) = P(V_t = O_k \mid q_t = s_j)$ ,  $1 \leq j \leq n$ ,  $1 \leq k \leq M$ ,  $\mu_j(k) \geq 0 \quad \forall j, k$  et  $\sum_{k=1}^M \mu_j(k) = 1$  ;
  - $\pi = \{\pi_i\}$ ,  $i = 1, \dots, n$  ; est le vecteur de distribution des probabilités de transitions initiales ;  $\pi_i$  représente la probabilité que l'état de départ du modèle est l'état  $i$ , soit :  $\pi_i = P(q_1 = s_i)$ ,  $1 \leq i \leq n$  avec :  $\pi_i \geq 0 \quad \forall i$  et  $\sum_{i=1}^n \pi_i = 1$  ;
  - $L : S \rightarrow 2^{AP}$  est une fonction d'étiquetage.

**Les états de croyance** : L'observation dépend stochastiquement et exclusivement de l'état courant, en général la même observation pourrait être émise par plusieurs états différents, par conséquence, nous sommes incertains au sujet de l'état courant mais nous pouvons rassembler l'historique des observations dans un état de croyance (ou état d'information) qui est une distribution sur  $S$ .

Un état de croyance n'est pas réellement un état de HMM, c'est une façon de décrire ce que nous savons au sujet de l'état donnant l'historique des observations. L'ensemble de tous les états de croyance possibles est appelé espace de croyance, et est dénoté par  $C$ .

Nous utilisons  $s^t$  avec  $s^t \in S$  pour indiquer l'état au temps  $t$ ,  $O^t \in \theta$  pour indiquer l'observation au temps  $t$  et  $c^t$  pour indiquer l'état de croyance à l'instant  $t$ . Soit  $O_i \in \theta$  où  $i = 1, 2, \dots, t$ . L'état de croyance  $c_t$  à l'instant  $t$  est la distribution de  $S$  à l'instant  $t$  donnant l'historique de l'observation  $o_0, o_1, \dots, o_t$  :  $c^t(s) = p(s^t = s \mid O^0 = o_0, O^1 = o_1, \dots, O^t = o_t, H)$ ,  $\forall s \in S$  [18].

Maintenant en donnant l'historique des observations  $o_0, o_1, \dots, o_t$ , la question est comment calculer l'état de croyance  $c_n$  : l'état de croyance au temps 0 dépend seulement de la distribution initiale et la première observation, l'état de croyance au temps  $t$  prise toute notre information sur le passé, comme résultat nous pouvons calculer l'état de croyance courant  $c_t$  en basant sur l'état de croyance précédent  $c_{t-1}$  et l'observation courante  $o_t$  ; comme illustre la figure 2.1.

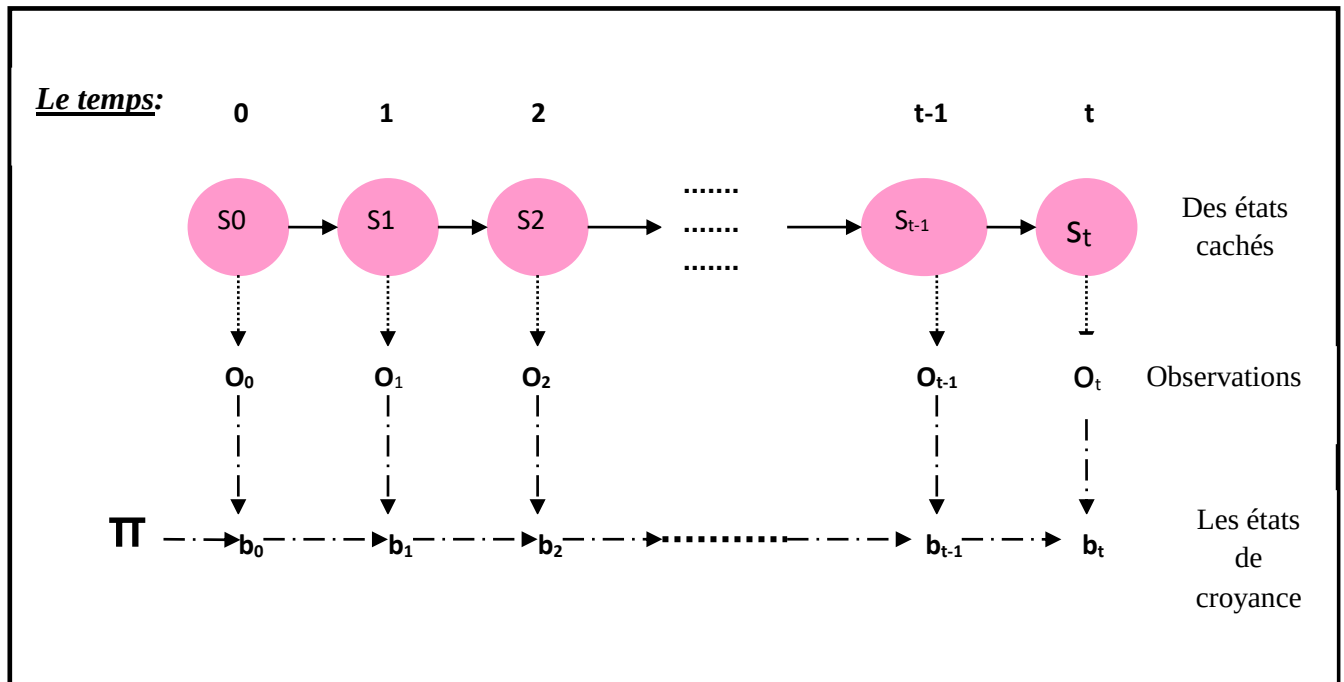


Figure 2.1- Mise à jour des états de croyance [18]

- Les états reliés par des flèches continues entre eux représentent les états cachés d'une exécution.
- Les flèches pointillées entre les états et les observations indiquent que l'observation  $O_t$  est produite par l'état  $s_t$  accordé à la fonction d'observation  $\mu$ , comme cas particulier  $C_0$  est en fonction de  $O_0$  et la distribution initiale  $\pi$ .

**Les trois problèmes des HMMs :** La réalisation par le modèle HMM  $H$  d'un processus Markovien de durée  $T$  est décrite par :

- $Q = q_1 q_2 \dots q_T$ , un chemin *a priori* caché parmi les  $N$  états,
- $O = O_1 O_2 \dots O_T$ , une suite d'observations appartenant à l'alphabet de  $M$  symboles.

À partir d'une suite d'observations  $O$  supposées émises par un modèle, trois problèmes de base doivent être résolus à fin de pouvoir exploiter le modèle  $H$  :

- **Le problème d'évaluation** : L'évaluation de la probabilité que la suite des observations ait été émise par un modèle. Lorsque plusieurs modèles existent, cette évaluation permet le choix du modèle le plus probable. Ce problème est efficacement résolu par l'algorithme *Forward* [11].
- **Le problème de décodage ou de reconnaissance** : La recherche de la séquence d'états d'un modèle ayant produit la séquence des observations, c'est à dire la recherche de la séquence cachée de plus forte probabilité et qui explique mieux ces observations. Pour résoudre ce problème efficacement, nous utilisons l'algorithme de *Viterbi* [11].
- **Le problème de re-estimation ou d'apprentissage** : L'apprentissage des paramètres d'un modèle. À partir d'un modèle donné *a priori* et d'observations supposées émises par ce modèle, nous cherchons les probabilités de transition et d'émission maximisant la vraisemblance des observations. La maximisation de la vraisemblance d'un modèle probabiliste est aujourd'hui effectuée par des algorithmes efficaces, comme l'algorithme *EM* (*Expectation Maximization*) ou l'algorithme de *Baum-Welch* [11].

## 2.3. Formalismes de spécification

Les model-checkers probabilistes les plus connus utilisent les logiques qui permettent de spécifier et de vérifier les systèmes probabilistes. Ces logiques sont : PCTL [19], CSL [20], PRCTL [21], CSRL [22, 23], QuaTE<sub>x</sub> [24], aCSL [25]. Dans le chapitre 3, nous verrons quelle logique utiliser par chaque model-checker.

Dans cette section, nous présentons les différents types de propriétés, puis la logique PCTL [19] étendue à la logique POCTL [18] utilisée par notre vérificateur formel (chapitre 4).

### 2.3.1. Quelques propriétés

Nous décrivons quelques propriétés courantes que nous allons utiliser à des fins de vérification.

- **Propriété d'atteignabilité** : énonce qu'une certaine situation peut être atteinte.
- **Propriété de sûreté** : affirme que rien de mauvais ne peut se produire.
- **Propriété de vivacité** : affirme que quelque chose de bon peut inévitablement se produire.
- **Propriété d'équité** : énonce que, sous certaines conditions, quelque chose aura lieu (ou n'aura pas lieu) un nombre infini de fois.

### 2.3.2. La logique PCTL

PCTL, introduite par H. Hansson et B. Jonsson [19] est une modification probabiliste de CTL (*Computational Tree Logic*) [8], où le quantificateur ( $\forall$ ) "*Pour tous les chemins*" et ( $\exists$ ) "*Existe un chemin*" a été remplacé par un opérateur probabiliste qui nécessite une probabilité donnée sur un certain ensemble de chemins qui est l'opérateur probabiliste  $P_{\sim p}$  ; où  $P \in [0,1]$  est une liaison de probabilité, et  $\sim \in \{>, <, \leq, \geq\}$ .

#### 2.3.2.1. La syntaxe de PCTL

La logique PCTL, autant qu'une prolongation probabiliste de la logique CTL, distingue les formules d'états, des formules de chemins.

Les formules d'états expriment les propriétés et les comportements attribuent aux états, et les formules de chemins expriment les propriétés et les comportements attribuent aux chemins [3].

Soit  $AP$  un ensemble fini de propositions atomiques. La syntaxe de PCTL est donnée dans la figure 2.2 :

$$\begin{aligned}
\phi &:= true \mid a \mid \neg \phi \mid \phi_1 \wedge \phi_2 \mid P_{\sim p}(\psi) \\
\psi &:= X\phi \mid \phi_1 U \phi_2 \mid \phi_1 U^{\leq k} \phi_2 \\
\text{Où : } a &\in AP, k \in \mathbb{N}, 0 \leq p \leq 1 \text{ et } \sim \in \{>, \geq, <, \leq\}
\end{aligned}$$

Figure 2.2- La syntaxe de la logique PCTL

Cette présentation s'appelle la présentation Backus-Naur [8]. Elle signifie que les formules de la logique PCTL sont construites récursivement à partir de cette syntaxe de la manière suivante :

✓ **Formules d'états**

- a)  $a$  est une formule d'état.
- b) Si  $\phi$  est une formule d'état, alors :  $\neg \phi$  est une formule d'état.
- c) Si  $\phi_1$  et  $\phi_2$  sont des formules d'états, alors  $\phi_1 \wedge \phi_2$  est une formule d'état.
- d) Si  $\psi$  est une formule de chemin,  $P_{\sim p}(\psi)$  est une formule d'état.
- e) Rien d'autre n'est une formule d'état.

✓ **Formules de chemin**

- a) Si  $\phi$  est une formule d'état, alors  $X\phi$  est une formule de chemin.
- b) Si  $\phi_1$  et  $\phi_2$  sont des formules d'états, alors  $\phi_1 U \phi_2$  est une formule de chemin.
- c) Si  $\phi_1$  et  $\phi_2$  sont des formules d'états et  $k \in \mathbb{N}$ , alors  $\phi_1 U^{\leq k} \phi_2$  est une formule de chemin.
- d) Rien d'autre n'est une formule de chemin.

### 2.3.2.2. La sémantique de PCTL

Soit  $AP$  un ensemble fini de propositions atomiques. La sémantique de PCTL est donnée dans la figure 2.3 :

| Sémantique pour les formules d'états    |                                                                                                                       |
|-----------------------------------------|-----------------------------------------------------------------------------------------------------------------------|
| $s \models \text{true}$                 | <i>pour tout</i> $s$                                                                                                  |
| $s \models a$                           | <i>ssi</i> $a \in L(s)$                                                                                               |
| $s \models \neg \phi$                   | <i>ssi</i> $s \not\models \phi$                                                                                       |
| $s \models (\phi_1 \wedge \phi_2)$      | <i>ssi</i> $s \models \phi_1 \wedge s \models \phi_2$                                                                 |
| $s \models P_{\sim p}(\psi)$            | <i>ssi</i> $\text{Prob}(\{ \pi \in \text{Paths}(s) \mid \pi \models \psi \}) \sim p$                                  |
| Sémantique pour les formules de chemins |                                                                                                                       |
| $\pi \models X \phi$                    | <i>ssi</i> $\pi = s_0 s_1 s_2 \dots$ et $s_1 \models \phi$                                                            |
| $\pi \models \phi_1 U \phi_2$           | <i>ssi</i> $\pi = s_0 s_1 s_2 \dots$ et $\exists k$ tq. $s_k \models \phi_2 \wedge \forall j < k, s_j \models \phi_1$ |
| $\pi \models \phi_1 U^{\leq k} \phi_2$  | <i>ssi</i> $\exists i \leq k$ tq. $s_i \models \phi_2 \wedge s_j \models \phi_1, \forall j < i$                       |

Figure 2.3- La sémantique de PCTL

Les opérateurs booléens  $\text{true}$ ,  $\wedge$  et la négation sont interprétés de la façon habituelle. Par exemple, la formule  $\phi_1 \wedge \phi_2$  est valide dans un état s'il satisfait à la fois  $\phi_1$  et  $\phi_2$ .

La formule  $P_{\sim p}(\psi)$  est satisfaite dans un état  $s$ , si et seulement si  $\text{Prob}(\{ \pi \in \text{Paths}(s) \mid \pi \models \psi \})$  est comprise dans l'intervalle défini par  $\sim p$ . Par exemple  $P_{\geq p}(X \phi)$  est satisfaite par le modèle  $M$  si la probabilité de l'ensemble des chemins partant de l'état initial  $s_0$  et dont le prochain état  $s_1$  satisfait  $\phi$ , est supérieure ou égale à  $p$  [3].

### 2.3.3. La logique POCTL

Dans un modèle HMM, les états sont cachés, et les processus cachés produisent un ordre des observations. La logique POCTL présenté dans [18], est une prolongation de la logique PCTL [19]. Avec POCTL nous pouvons énoncer des propriétés sur les HMMs telles que " *Il y a au moins une probabilité de 90 % que le modèle produit une séquence d'observations donnée* ".

POCTL est fondamentalement une prolongation de PCTL où l'opérateur *next* est équipé d'une contrainte d'observation. D'autre part, POCTL peut être considéré également comme une variante de la logique temporelle ACTL (*Action Computation Tree Logic*), présentée par De Nicola et autres [26], dans lequel l'opérateur *next* habituel est prolongé pour contraindre l'étiquette d'action de la transition. POCTL se compose des formules d'état, des formules de chemin et des formules d'état de croyance qui nous permettent de spécifier des propriétés sur les HMMs. De cette façon, nous pouvons énoncer des propriétés sur des observations, par exemple,  $X_o\phi$  signifie que la prochaine observation est  $o$  et le chemin suivant satisfait  $\phi$  [18].

#### 2.3.3.1. La syntaxe de POCTL

Soit  $H = (S, P, L, \theta, B, \pi)$  est un HMM avec  $o \in O$ . La syntaxe de la logique POCTL est définie dans la figure 2.4 :

$$\begin{aligned}
 \phi &:= a \mid \phi_1 \wedge \phi_2 \mid \neg \phi \mid \varepsilon \\
 \psi &:= X_o \phi \mid \phi_1 U^{\leq k} \phi_2 \\
 \varepsilon &:= P_{\sim p}(\psi) \mid \neg \varepsilon \mid \varepsilon_1 \wedge \varepsilon_2 \\
 \text{Où : } &k \in \mathbb{N}, 0 \leq p \leq 1 \text{ et } \sim \in \{>, \geq, <, \leq\}
 \end{aligned}$$

Figure 2.4- La syntaxe de la logique POCTL

La syntaxe de POCTL se compose des formules d'état, des formules de chemin et des formules d'état de croyance. Nous employons  $\phi$  pour la formule d'état et  $\psi$  pour la formule de chemin et  $\varepsilon$  s'appelle la formule d'état de croyance. Pour un HMM, nous sommes incertains au sujet de l'état actuel, mais nous connaissons toujours l'état courant de croyance. Par conséquent, nous voulons savoir si quelques propriétés (probabilistes) sont valides dans des états de croyance. Nous considérons la propriété : " Il y a au moins une probabilité de 90 % que le modèle produit l'ordre des observations  $O = \{o_0, o_1, \dots, o_n\}$  ". Cette propriété peut être exprimée en POCTL comme suit :  $P_{\geq 0.9}(X_{o_0} X_{o_1} \dots X_{o_n})$  comme indiqué par Rabiner dans [11]. Ceci peut être exprimé par une formule d'état de croyance :  $\varepsilon = P_{\geq 0.9}(X_{o_0} X_{o_1} \dots X_{o_n})$  [18].

### 2.3.3.2. La sémantique de POCTL

Soit  $H = (S, P, L, \theta, B, \pi)$  un HMM avec  $s \in S$  et  $\sigma \in Path$ . La sémantique de POCTL (figure 2.5) est définie par une relation de satisfaction (dénotée par  $\models$ ) entre un état  $s$  et une formule d'état  $\phi$ , ou entre un chemin  $\sigma$  et une formule de chemin  $\psi$ , ou entre un état de croyance  $b$  et une formule d'état de croyance  $\varepsilon$ . Nous écrivons  $H, s \models \phi$  ;  $H, \sigma \models \psi$  et  $H, b \models \varepsilon$  si l'état  $s$ , le chemin  $\sigma$  et l'état de croyance  $b$  satisfont la formule d'état  $\phi$ , la formule de chemin  $\psi$  et la formule d'état de croyance  $\varepsilon$ , respectivement [18].

|                                                |                                                                                                               |
|------------------------------------------------|---------------------------------------------------------------------------------------------------------------|
| $s \models a$                                  | $ssi \quad a \in L(s)$                                                                                        |
| $s \models \phi_1 \wedge \phi_2$               | $ssi \quad s \models \phi_1 \wedge s \models \phi_2$                                                          |
| $s \models \neg \phi$                          | $ssi \quad s \models \text{non } \phi$                                                                        |
| $s \models \varepsilon$                        | $ssi \quad b_s \models \varepsilon$                                                                           |
| $\sigma \models X_o \phi$                      | $ssi \quad \sigma_o[0] \models o \wedge \sigma[1] \models \phi$                                               |
| $\sigma \models \phi_1 U^{\leq k} \phi_2$      | $ssi \quad \exists 0 \leq j \leq k (\sigma[j] \models \phi_2 \wedge \forall i < j: \sigma[i] \models \phi_1)$ |
| $b \models P_{\sim p}(\psi)$                   | $ssi \quad p_b\{\sigma / \sigma \models \psi\} \sim p$                                                        |
| $b \models \neg \varepsilon$                   | $ssi \quad b \not\models \varepsilon$                                                                         |
| $b \models \varepsilon_1 \wedge \varepsilon_2$ | $ssi \quad b \models \varepsilon_1 \wedge b \models \varepsilon_2$                                            |

Figure 2.5- La sémantique de POCTL

Un chemin satisfait le nouvel opérateur  $X$  s'il commence par l'observation  $o$  et le suffixe  $\sigma[1]$  satisfait  $\phi$ . Soit  $\Omega$  un ensemble d'observations, c.-à-d.  $\Omega \subseteq \Theta$ .

Nous avons évidemment  $\sigma \models X_\Omega \phi$  ssi  $\sigma_o[0] \in \Omega \wedge \sigma[1] \models \phi$ .

Le prochain opérateur habituel peut être décrit  $X\phi \equiv X_\Theta \phi$ . Ainsi, la logique PCTL peut être considérée en tant que logique secondaire de POCTL [18].

## 2.4. Algorithmes de model-checking probabiliste

Il existe trois algorithmes de model-checking probabiliste : PCTL sur les DTMCs, PCTL sur les MDPs et CSL sur les CTMCs. Un algorithme de model-checking pour PCTL ou CSL prend un modèle du type approprié, d'une formule  $\emptyset$  dans la logique, et renvoie l'ensemble  $Sat(\emptyset)$  contenant les états du modèle qui satisfont  $\emptyset$ . La structure globale de l'algorithme de model-checking probabiliste est la même dans chacun des trois cas, et provient de l'algorithme de model-checking original de CTL présenté dans [27].

De nombreuses méthodes ont été proposées pour vérifier des formules d'une logique temporelle sur des systèmes probabilistes, voire concurrents probabilistes. Vardi [28] a ainsi mis au point une technique à base d'automates pour la vérification de propriétés qualitatives exprimant le fait qu'une formule du temps linéaire est satisfaite avec probabilité 0 ou 1 (propriétés dites *qualitatives*).

Courcoubetis et Yannakakis [29] ont étudié la vérification probabiliste de propriétés *quantitatives* (probabilité de satisfaction arbitraire) également exprimées dans le cadre du temps linéaire. Pour le cas des systèmes totalement probabilistes (DTMCs) [8, 10], la complexité en temps de leur méthode, est polynomiale dans la taille de l'espace des états et exponentielle dans la taille de la formule. Dans le cas concurrent, la complexité en temps est linéaire dans la taille du système, et doublement exponentielle dans la taille de la formule.

Hansson et Jonsson [30] ont introduit la logique PCTL et proposé un algorithme de model-checking pour les formules de cette logique exprimant des propriétés sur des systèmes totalement probabilistes. Leur algorithme combine un calcul d'accessibilité

(comme dans le cas des systèmes non probabilistes) et la résolution de systèmes d'équations linéaires pour calculer la probabilité des formules contenant l'opérateur *Until*.

Pour les systèmes concurrents probabilistes, Bianco et de Alfaro [31] ont montré, dans l'un de leurs articles, que les probabilités minimales et maximales de satisfaction d'une formule construite avec *Until* pouvaient être calculées en résolvant un problème d'optimisation linéaire. La complexité en temps de leur algorithme est polynomiale en la taille du système que l'on considère, et linéaire dans la taille de la formule que l'on vérifie. Depuis quelques temps, des méthodes qui tournent autour de la notion du test statistique d'hypothèses apparaissent dans la littérature.

Ainsi, dans un article récent [32], Younes et Simmons décrivent une méthode pour vérifier des propriétés de systèmes à événements discrets. De manière similaire, la vérification qualitative de propriété a été considérée pour les modèles qui montrent le comportement probabiliste et non déterministe, c.-à-d. les modèles comme les (MDPs) [33, 34, 35, 36, 37]. Une des différences principales ici est qu'une notion de l'équitabilité doit être considérée.

La plus difficile est la vérification des caractéristiques quantitatives. Pour ces derniers, nous devons calculer la probabilité exacte qu'une propriété donnée est satisfaite. Le cas pour DTMCs a été considéré par Courcoubetis et Yannakakis [37, 38], Hansson et Jonsson [30], et Aziz et autres. [39].

La vérification des propriétés quantitatives pour (MDPs) a été considérée par Courcoubetis et Yannakakis [40] et Bianco et de Alfaro [31]. Le model-checking de PCTL est montré pour réduire la solution d'un problème linéaire d'optimisation. Bianco et De Alfaro [31] ont également présenté un algorithme pour le PCTL.

Dans [41], Baier et Kwiatkowska ont montré comment les algorithmes pour le model-checking des propriétés quantitatives peuvent être prolongées pour incorporer une notion d'équitabilité basée sur celle de Vardi [35]. Une autre amélioration de leur algorithme a été donnée par Baier dans [42].

Dans [37], Aziz et autres ont proposé la logique CSL pour fournir des moyens d'indiquer formellement des propriétés de CTMCs. CSL peut être vu comme

prolongation de PCTL. Elle présente l'opérateur Until borné qui peut être employé pour raisonner au sujet du comportement en temps réel de CTMC.

Baier et autres [38] ont alors prolongé CSL en ajoutant un opérateur à la raison sur le comportement équilibré de CTMC et a présenté le premier modèle examinant l'algorithme pour assurer la logique. Des probabilités équilibrées sont calculées de la manière habituelle en résolvant un système d'équation linéaire. Pour l'opérateur Until borné, ils ont proposé un algorithme basé sur une méthode itérative pour la solution approximative d'un système d'équation d'intégrales de Volterra. Dans [43], Baier et autres ont présenté une méthode alternative de model-checking de l'opérateur Until borné de CSL en employant une conversion en analyse passagère. Ceci permet d'employer des techniques de calcul standards et efficaces. L'algorithme a été amélioré plus loin par Katoen et autres dans [44].

## 2.5. Conclusion

Le principal avantage lié à l'utilisation du model-checking est que cette technique de vérification est à la fois simple à mettre en œuvre et entièrement automatisable. Si le modèle du système est de taille exploitable, alors il existe des algorithmes implantant des model-checkers capables de répondre à la question : « *le modèle du système satisfait-il la propriété ?* », sans assistance de la part de l'utilisateur. De plus, lorsque la réponse est non, la plupart des model-checkers exhibent un chemin d'exécution violant la propriété, ce qui facilite la correction de l'erreur. Cependant, outre le fait que cette technique ne permet d'adresser que les modèles dont le nombre d'états est fini, le principal problème pour l'utilisation de model-checking est connu sous le nom d'explosion combinatoire du nombre d'états. La modélisation des systèmes réels (qui peuvent être complexes) conduit généralement à la création d'un nombre d'états tel qu'il devient impossible de les conserver tous dans la mémoire d'un ordinateur, ce qui empêche en pratique l'exploration exhaustive de l'espace d'états. Des techniques permettant de combattre ce phénomène d'explosion combinatoire doivent alors être appliquées pour rendre possible le model-checking des systèmes de grande taille.

Les HMMs connaissent un essor important. Ceci, en raison de nombreux avantages qu'ils offrent dans la possibilité d'intégrer correctement plusieurs sources de connaissance grâce à l'usage d'une théorie, telle que celle des probabilités, l'existence d'algorithmes d'apprentissage des modèles et la facilité de représenter des régularités statistiques de l'application à traiter.

Les logiques temporelles fournissent une manière de raisonnement au sujet du comportement qui change au cours du temps. Leur application dans l'informatique a été introduite par Pnueli qui a argué le fait que les techniques existantes pour la vérification formelle n'étaient pas proportionnées pour les systèmes embarqués. Nous avons adopté dans notre approche la logique temporelle POCTL. D'une part, POCTL est fondamentalement une prolongation de PCTL où l'opérateur next est équipé d'une contrainte d'observation. D'autre part, POCTL peut également être considérée comme variante de la logique temporelle ACTL, dans lequel l'opérateur next habituel est prolongé pour contraindre l'étiquette d'action de la transition.

---

# **Chapitre 3**

## **Outils de model-checking probabiliste**

---

### 3.1. Introduction

Les méthodes de vérification de logiciels et de systèmes embarqués par model-checking ont connu un développement important avec l'évolution rapide des logiciels et leurs applications dans plusieurs domaines comme le commerce électronique, la santé, etc. Ces méthodes ont été utilisées avec succès pour la détection des erreurs pendant le développement de logiciels.

Au cours des dernières années, les outils de model-checking (model-checkers) ont permis de découvrir des erreurs dans des systèmes de taille de plus en plus importante. Ils ont aussi permis de valider d'autres systèmes. Les succès et les limitations de ces outils sont à l'origine d'une importante activité de recherche dans le domaine de la vérification. Cette recherche, tout en étant d'un haut niveau théorique, se confronte systématiquement à la réalité par le biais du développement de nouveaux outils, à chaque fois plus performants.

### 3.2. Model-checkers probabilistes

Il existe un nombre limité d'outils disponibles pour le model-checking probabiliste. Nous explorons cette partie en passant en revue les réalisations qui ont été produites. Dans cette section, nous présentons sept model-checkers probabilistes des plus connus et qui sont :

**PRISM, APMC, ETMCC, MRMC, YMER, VESTA, UPPAAL-PRO.** Ils ont les caractéristiques communes suivantes :

- **Des outils universitaires :** Il s'agit d'outils qui participent au développement de nouvelles méthodes, de nouvelles théories, pour le model-checking. Ils mettent en œuvre des techniques nouvelles et originales, très prometteuses, qui seront probablement présentes sous une forme ou sous une autre, dans les outils industriels de prochaine génération.
- **Des outils disponibles librement :** Les outils retenus sont tous accessibles librement et gratuitement via internet. Le lecteur pourra facilement tester lui-même l'outil de son choix sur tel ou tel problème qui le concerne.
- **Des outils utilisés par de nombreuses équipes :** chaque outil dispose d'une communauté d'utilisateurs dépassant, parfois largement, le groupe à l'origine de

l'outil. C'est un argument fort, permettant d'affirmer que l'outil en question a une indéniable utilité. C'est aussi une source d'évaluation objective des points forts et faibles de l'outil. Notons que la libre disponibilité de l'outil aide grandement à sa diffusion.

### 3.2.1. PRISM

PRISM ( *PRobabilistic Symbolic Model-checker* ) [45] représente le model-checker symbolique probabiliste. L'outil est développé en 2001 à l'université de Birmingham (United Kingdom) pour modéliser et analyser des systèmes probabilistes. C'est un outil libre et de source ouverte, distribué sous le permis de grand public de GNU (GPL *Gnu Public License*). Il est disponible librement via Internet : <http://www.cs.bham.ac.uk/~dyp/prism/>.

### 3.2.2. APMC

APMC ( *Approximate Probabilistic Model Checker* ) est un logiciel écrit en collaboration entre Sylvain Peyronnet, Thomas Héroult et Frédéric Magniette [46]. Ce logiciel est une implémentation de la méthode d'approximation de la vérification de propriétés monotones pour les systèmes totalement probabilistes. Il est un logiciel disponible librement car distribué dans le cadre de la licence GPL qui implémente l'algorithme AGA (*Algorithme Générique d'Approximation*). Il est notamment téléchargeable sur le site: <http://www.lri.fr/~syp/APMC>.

APMC est écrit en langage C à l'aide des outils lex (un générateur d'analyseur lexical) et yacc (un générateur d'analyseur syntaxique), l'interface graphique a été construite à l'aide du *toolkit* graphique (GTK). Il utilise une architecture de calcul client/serveur pour distribuer la génération et la vérification des chemins sur l'ensemble des clients (c'est-à-dire les machines qui travaillent sur la vérification).

Du point de vue utilisateur, APMC prend en entrée une formule de LTL ainsi que la description d'un système probabiliste écrit dans le même langage de description que l'outil PRISM. Le programme prend également en entrée la probabilité de satisfaction de la formule LTL que l'on souhaite, ainsi que la longueur des chemins que l'on va considérer et les paramètres d'approximation et de confiance  $\varepsilon$  et  $\delta$  respectivement.

Ces paramètres peuvent être changés par l'intermédiaire de l'interface graphique. Par cette interface, l'utilisateur peut également choisir différentes stratégies de compilation/génération de code qui permettent de modifier le rapport temps de calcul / espace mémoire utilisé.

### 3.2.3. ETMCC

ETMCC (*Erlangen-Twente Markov Chain Checker*) est développé par le groupe de modélisation stochastique et de vérification de l'université d'Erlangen-Nürnberg, de l'Allemagne, et du groupe de méthodes formelles et d'outils de l'université de Twente, les Pays Bas [47]. C'est une implémentation d'un prototype de model-checker pour les chaînes de Markov à temps continu. L'outil est libre pour des organisations à but non lucratif. L'utilisateur doit compléter une forme d'accord de licence avant de télécharger et utiliser l'outil. Il est disponible librement via Internet : <http://www7.informatik.uni-erlangen.de/etmcc/>.

### 3.2.4. MRMC

MRMC (*Markov Reward Model Checker*) [48] est un model-checker pour les (DMRMs : *Discret Markov Reward Model*) [21] et les (CMRMs : *Continuous Markov Reward Model*) [22, 23]. Il est distribué sous le permis de grand public de GNU (GPL). Il est développé par le groupe de modélisation des logiciels et de vérification (MOVES) à l'université de RWTH Aachen (Allemagne) et le groupe de méthodes formelles et d'outils à l'université de Twente (les Pays Bas). Il est disponible via internet : <http://www.cs.utwente.nl/~zapreevis/mrmc/>.

### 3.2.5. YMER

YMER [49] est un outil pour vérifier les propriétés passagères des systèmes stochastiques. Il supporte des chaînes de Markov à Temps Continu (CTMCs) [10] et des processus généralisés de Semi-Markov (GSMPs) [50]. YMER met en application des techniques statistiques de model-checking, basées sur la simulation d'événements et l'échantillonnage pour le contrôle d'acceptation discret, pour le model-checking de CSL [49]. Il supporte également des techniques numériques pour le model-checking de CTMCs [37]. L'outil est développé par

Håkan Younes à l'université de Carnegie Mellon (Pittsburgh). Il est distribué sous le permis grand public GNU (GPL). L'outil est disponible via internet :

<http://www.cs.cmu.edu/~lorens/ymer.html>.

### 3.2.6. VESTA

VESTA [51] est un outil pour l'analyse statistique des systèmes probabilistes. Il supporte le model-checking statistique et l'évaluation statistique des valeurs prévues des expressions temporelles. Comme YMER, l'outil peut seulement fournir des garanties probabilistes de l'exactitude. Il est développé à l'université de l'Illinois au Urbana-Champaign et peut être obtenue via internet : <http://osl.cs.uiuc.edu/~ksen/vesta2>.

### 3.2.7. UPPAAL-PRO

Uppaal [52] est un environnement d'outils intégrés (une boîte à outils pour la vérification des systèmes en temps réel) pour la modélisation, la simulation et la vérification de systèmes en temps réel tels que des réseaux d'automates chronométrés, étendus avec des types de données (entiers délimités, tableaux, etc.). Il a été appliqué avec succès dans des études de cas allant des protocoles de communication aux applications multimédias.

L'outil est développé en collaboration avec le département de technologie de l'information de l'université d'Uppsala, en Suède et le département d'informatique de l'université d'Aalborg au Danemark.

Dans Uppaal, un système est modélisé en tant que réseau de plusieurs automates chronométrés en parallèle, qui est une machine à états finis étendue avec des variables d'horloge. Un état du système est défini par les emplacements de tous les automates, les contraintes d'horloge et les valeurs des variables discrètes. Chaque automate peut déclencher une transition séparément ou se synchroniser avec un autre automate, ce qui conduit à un nouvel état. Il est disponible pour téléchargement à partir : <http://www.uppaal.com>

### 3.3. Comparaisons

Pour donner une vue comparative des model-checkers probabilistes de la section précédente, Nous présentons les tableaux suivants qui montrent une vue d'ensemble d'outils, les plateformes, les modèles, les logiques supportées, les structures de données, et enfin les algorithmes mis en application, respectivement [53].

| Outil          | Université                                                       | Type                           | Langage de programmation |     |   | Interface utilisateur |                      |
|----------------|------------------------------------------------------------------|--------------------------------|--------------------------|-----|---|-----------------------|----------------------|
|                |                                                                  |                                | Java                     | C++ | C | GUI                   | cmdl. <sup>(1)</sup> |
| APMC           | Paris VII                                                        | numérique                      |                          |     | ✓ | ✓                     |                      |
| PRISM          | Birmingham                                                       | numérique                      | ✓                        | ✓   | ✓ | ✓                     | ✓                    |
| ETMCC          | Erlangen-Nürnberg<br>et Twente                                   | numérique                      | ✓                        |     |   | ✓                     |                      |
| MRMC           | Aachen et Twente                                                 | numérique                      |                          |     | ✓ |                       | ✓                    |
| YMER           | Carnegie Mellon<br>(Pittsburgh)                                  | Statistique<br>ET<br>numérique |                          | ✓   | ✓ |                       | ✓                    |
| VESTA          | Urbana-Champaign, Illinois                                       | statistique                    | ✓                        |     |   | ✓                     | ✓                    |
| UPPAAL<br>_PRO | Uppsala University (Suède)<br>& Aalborg University<br>(Danemark) | Statique                       | ✓                        |     |   | ✓                     | ✓                    |

Tableau 3.1- Vue d'ensemble d'outils

<sup>(1)</sup> - Ligne de commande.

| Outil      | Plateforme |         |         |                  |
|------------|------------|---------|---------|------------------|
|            | Linux      | Windows | Solaris | Mac OS X         |
| APMC       | ✓          | ✓       |         |                  |
| PRISM      | ✓          | ✓       | ✓       | ✓ <sup>(1)</sup> |
| ETMCC      | ✓          | ✓       | ✓       |                  |
| MRMC       | ✓          |         |         |                  |
| YMER       | ✓          |         |         |                  |
| VESTA      | ✓          | ✓       |         |                  |
| UPPAAL_PRO | ✓          | ✓       |         | ✓                |

Tableau 3.2- Plateformes supportées par les outils

| Outil          | Modèles supportés |      |     |    |     |      |      |      |                  |
|----------------|-------------------|------|-----|----|-----|------|------|------|------------------|
|                | DTMC              | CTMC | MDP | PA | PTA | DMRM | CMRM | GSMP | Reactive modules |
| APMC           |                   |      |     |    |     |      |      |      | ✓                |
| PRISM          | ✓                 | ✓    | ✓   | ✓  | ✓   |      |      |      |                  |
| ETMCC          |                   | ✓    |     |    |     |      |      |      |                  |
| MRMC           | ✓                 | ✓    |     | ✓  | ✓   | ✓    | ✓    |      |                  |
| YMER           |                   | ✓    |     |    |     |      |      | ✓    |                  |
| VESTA          | ✓                 | ✓    |     |    |     |      |      |      |                  |
| UPPAA<br>L_PRO |                   |      |     |    | ✓   |      | ✓    |      |                  |

Tableau 3.3- Modèles supportés par les outils

---

<sup>(1)</sup>- En cour de développement.

| Outil          | Logiques supportées |       |     |      |                    |                     |     |
|----------------|---------------------|-------|-----|------|--------------------|---------------------|-----|
|                | PCTL                | PRCTL | CSL | CSRL | QuaTE <sub>x</sub> | aCSL <sup>(1)</sup> | LTL |
| APMC           |                     |       |     |      |                    |                     | ✓   |
| PRISM          | ✓                   |       | ✓   |      |                    |                     |     |
| ETMCC          |                     |       | ✓   |      |                    | ✓                   |     |
| MRMC           | ✓                   | ✓     | ✓   | ✓    |                    |                     |     |
| YMER           |                     |       | ✓   |      |                    | ✓                   |     |
| VESTA          | ✓                   | ✓     |     |      | ✓                  |                     |     |
| UPPAA<br>L_PRO |                     | ✓     |     |      |                    |                     |     |

Tableau 3.4- Logiques supportées par les outils

| Outil      | Structures de données           |                |         |
|------------|---------------------------------|----------------|---------|
|            | Matrice clairsemée [54, 55, 56] | MTBDD [57, 58] | Hybride |
| APMC       |                                 |                |         |
| PRISM      | ✓                               | ✓              | ✓       |
| ETMCC      | ✓                               |                |         |
| MRMC       | ✓                               |                |         |
| YMER       |                                 |                | ✓       |
| VESTA      |                                 |                |         |
| UPPAAL_PRO |                                 |                |         |

Tableau 3.5- Structures de données des outils

---

<sup>(1)</sup> - action-based CSL.

| Outil      | Logique  | Références d'algorithmes |
|------------|----------|--------------------------|
| APMC       | LTL      | [59]                     |
| PRISM      | CSL      | [43, 44]                 |
|            | PCTL     | [30, 31, 41, 42]         |
| ETMCC      | CSL      | [47, 60]                 |
| MRMC       | CSL      | [20]                     |
|            | PCTL     | [30]                     |
|            | CSRL     | [61, 62]                 |
|            | PRCTL    | [21]                     |
| YMER       | CSL      | [32]                     |
| VESTA      | CSL/PCTL | [32, 63, 64]             |
| UPPAAL_PRO | PRCTL    | [21]                     |

Tableau 3.6- Algorithmes utilisés par les outils

### 3.4. Conclusion

Nous avons vu dans ce chapitre les principaux model-checkers probabilistes et présenté chacun d'eux en détail afin de comprendre les travaux qui ont été accomplis dans ce domaine de recherche qui concerne les techniques et les outils de model-checking probabiliste. Ce chapitre termine l'état de l'art de cette thèse et nos contributions seront abordées dans la deuxième partie.

Suite à cette analyse, nous avons fait les suggestions suivantes pour améliorer ces outils. Pour YMER, il serait très utile s'il soutenait plus d'opérateurs de CSL/PCTL, de sorte que ses moteurs de calculs deviennent applicables à une classe plus large des problèmes de model-checking. En outre, il serait intéressant que YMER emploie exactement la même syntaxe que PRISM, améliorant l'interopérabilité d'outil. Pour VESTA, nous suggérons d'améliorer son temps de fonctionnement. En outre, son applicabilité serait agrandie en améliorant l'expressivité du modèle en ajoutant un opérateur de composition parallèle. Pour PRISM, une liaison avec YMER serait pertinente, un utilisateur appellerait le model-checker YMER en appuyant sur un simple bouton. Avec la fonctionnalité courante d'exportation de modèles, il serait commode si des modèles pourraient également être importés. Ceci éliminerait le temps de construction de modèle. Pour MRMC, il est suggéré d'améliorer l'exécution pour de plus grands modèles. Pour ETMCC, il n'est fourni aucune recommandation.

Nous avons constaté qu'aucun de ces model-checkers n'utilise les HMMs comme un formalisme de modélisation des systèmes et le POCTL comme un formalisme de spécification des propriétés. Ainsi, nous préconisons un nouvel outil probabiliste faisant appel aux modèles HMM et POCTL qui serait vraisemblablement le premier model-checker à les utiliser.

---

# **Deuxième partie**

## **Contributions**

---

---

**Chapitre 4**

**Contribution 1:**

**L'outil de vérification**

**HMM\_Model-Checker**

---

## 4.1. Introduction

La mise en œuvre de tout système informatique, nécessite une phase de conception, durant laquelle nous essayerons de définir les différentes étapes à suivre et les différents traitements à effectuer pour satisfaire les besoins spécifiés. Notre conception a comme but la mise en œuvre d'un outil de model-checking probabiliste permettant de déterminer si un système décrit dans un modèle HMM satisfait une formule POCTL.

Au cours du développement de ce chapitre, nous donnerons les détails de notre conception. Nous décrirons aussi l'implémentation de notre vérificateur que nous avons appelé « HMM\_Model-Checker ». Tout d'abord, nous donnerons l'environnement de développement et le matériel utilisé. Puis, nous discuterons les fonctionnalités offertes par notre outil et son schéma de développement pour donner une idée globale sur son implémentation. Ensuite, nous présenterons l'outil, son interface graphique, ainsi que son fonctionnement détaillé. Notons que l'outil est implémenté dans l'environnement Netbeans qui procure des caractéristiques adaptées à nos besoins.

## 4.2. Contexte de notre approche

Pour un HMM [11] donné, nous nous sommes souvent intéressés aux propriétés du processus stochastique fondamental, c.-à-d. du DTMC [8, 10] de base de ce HMM, ce qui a été fait par les model-checkers probabilistes existants. En outre, nous nous sommes préoccupés également pour raisonner sur les propriétés de l'autre ensemble de processus stochastiques qui produisent les observations, ce qui constitue un plus dans notre approche par rapport aux approches précédentes. Dans le chapitre 2, nous avons vu une logique appelée POCTL [18], qui se compose des formules d'état, des formules de chemin et des formules d'état de croyance qui nous permet d'indiquer des propriétés d'intérêt sur les HMMs. Notre approche est agencée en trois étapes :

- **La modélisation du système à inspecter** : Il s'agit de représenter le comportement du système sous forme d'un modèle de Markov caché (HMM) construit à l'aide de la bibliothèque Jahmm [65], où chaque état du HMM représente une instance du système à un moment donné, et les transitions possibles entre ces états indiquent le comportement du système pour effectuer le passage d'un état à un autre.

- **La spécification des propriétés à vérifier :** Afin d'analyser un modèle probabiliste, il est nécessaire d'identifier une ou plusieurs propriétés du modèle (exprimées en fonction d'une formule logique POCTL saisie dans un éditeur de propriétés) qui peuvent être évaluées par l'outil.
- **La vérification de la satisfaction des propriétés :** est assurée par l'algorithme de model-checking pour POCTL sur un HMM présenté auparavant dans [18], adapté de celui pour PCTL sur un DTMC [19], car d'une part le HMM est considéré comme une extension de DTMC en ajoutant l'ensemble des observations, la matrice des probabilités d'observations et le vecteur des probabilités initiales, et d'autre part POCTL est une prolongation de PCTL [19] où l'opérateur Next est rendu porteur d'une contrainte d'observation. Nous décrirons chacune des entités dans les sections suivantes.

### 4.3. Modélisation

La complexité sans cesse croissante des systèmes informatiques impose le recours à des formalismes et outils facilitant tant leur conception que leur réalisation. Le HMM est un formalisme particulièrement intéressant, et très utilisé pour la spécification des systèmes complexes.

Dans un modèle de Markov, l'état courant du système stochastique est représenté par une variable pouvant prendre plusieurs valeurs qui sont chacune un des états possibles du système. L'évolution du système est complètement caractérisée par les probabilités de passage d'une valeur à l'autre, à chaque pas de temps. Ce type de modèle est caractérisé par l'hypothèse de Markov, selon laquelle l'état du système (représenté par la variable d'état) à l'instant  $t$  ne dépend que de son état à l'instant  $t-1$ .

Dans un HMM, l'état courant du système n'est pas directement observable, il est caché. Nous ajoutons alors une autre variable observable qui est l'observation courante du processus, et cette variable n'est fonction que de l'état courant. Ici encore, nous avons une fonction probabiliste qui caractérise les probabilités de telle ou telle observation en fonction de l'état courant caché.

Pour construire un HMM, il est donc nécessaire de définir quels sont les états dans lesquels le système peut être (les différentes valeurs que peut prendre la variable “état”) et quelles sont les observations possibles. Nous ne travaillons pas sur les données brutes du système que l'on veut modéliser, il faut faire une étape de conception qui vise à bien définir le type de modèle que l'on va construire. Ensuite nous construirons le modèle par apprentissage suivant l'algorithme utilisé classiquement pour ce type de modèle.

## 4.4. Spécification des propriétés

Afin d'analyser un modèle probabiliste qui a été indiqué et construit en HMM\_Model-Checker, il est nécessaire d'identifier une ou plusieurs propriétés du modèle qui peut être évalué par l'outil. Comme cela est typiquement le cas pour des techniques formelles de vérification, en HMM\_Model-Checker ceci est fait en utilisant la logique temporelle. Plus spécifiquement, des propriétés sont exprimées en respectant la logique POCTL, prolongement de la logique PCTL [19] qui est elle-même un prolongement probabiliste de la logique temporelle classique CTL [27]. Plus de détails sur les caractéristiques des propriétés autorisées en HMM\_Model-Checker seront fournis dans les sections suivantes. Nous déjà avons vu la syntaxe et la sémantique précises de la logique POCTL [18] dans le chapitre 2.

### 4.4.1. Identification d'un ensemble d'états

Une des tâches fondamentales indiquant des propriétés d'un modèle est d'identifier les ensembles ou les classes particulières des états du modèle. Par exemple, pour vérifier une propriété, il est d'abord nécessaire d'identifier les états du modèle qui correspondent aux situations où la propriété est satisfaite.

En HMM\_Model-Checker, ceci est réalisé simplement en écrivant une expression qui sera évaluée à une valeur booléenne. Cette expression contiendra typiquement des références aux variables (et aux constantes) du modèle auquel elle est reliée. L'ensemble des états correspondants à cette expression inclut les états évalués à « true ».

### 4.4.2. Syntaxe de base

La syntaxe de base de l'expression d'une propriété (prop) dans notre HMM\_Model-Checker est comme suit, l'ajout par rapport à PCTL est mis en gras :

$$(\text{prop}) ::= \text{true} \mid \text{false} \mid (\text{expr}) \mid \textbf{(\text{cryanceprop})} \mid$$

$$\quad \quad \quad ! (\text{prop}) \mid$$

$$(\text{prop}) \ \& \ (\text{prop}) \mid$$

$$(\text{prop}) \mid (\text{prop}) \mid$$

$$(\text{prop}) \Rightarrow (\text{prop}) \mid$$

$$\textbf{(\text{cryanceprop})} ::= \textbf{P}_{(\text{op})(p)} [\textbf{(\text{pathprop})}]$$

$$\quad \quad \quad \textbf{! (\text{cryanceprop})} \mid$$

$$\quad \quad \quad \textbf{(\text{cryanceprop})} \ \& \ \textbf{(\text{cryanceprop})}$$

$$(\text{pathprop}) ::= X_{(\Omega)}(\text{prop}) \mid$$

$$(\text{prop}) \ U \ (\text{prop}) \mid$$

$$(\text{prop}) \ U_{(\text{time})} \ (\text{prop}) \mid$$

$$(\text{time}) ::= \geq(t) \mid \leq(t) \mid [(t), (t)] \mid$$

Où :

- (expr) est une expression qui va être évaluée à un booléen,
- (op) est un opérateur relationnel (<, <=, >= ou >),
- (p) est une expression évaluée à une variable réelle dans l'intervalle [0,1],
- (t) est une expression évaluée à une variable réelle ou entière non négative,
- ( $\Omega$ ) est un ensemble d'observations,  $\Omega \subseteq \Theta$  ( $\Theta$  est l'ensemble de toutes les observations).

Une propriété est évaluée en respectant un état du modèle. Pour la syntaxe donnée précédemment, toutes les propriétés sont évaluées à des valeurs booléennes c.-à-d. pour n'importe quel état  $s$  d'un modèle HMM, une propriété est évaluée soit « true » ou « false » dans cet état. En équivalence, nous disons aussi qu'une propriété est « satisfaite » ou « n'est pas satisfaite » en un état.

Pour les opérateurs de base de la logique (true, false, (expr), !, &, | et =>), les sémantiques sont comme pour la logique propositionnelle classique :

- true est « true » dans tous les états,
- false est n'est pas « true » dans aucun état,
- (expr) est « true » si l'expression (expr) évalue à « true »,
- ! (prop) est « true » si (prop) n'est pas « true »,
- (prop1) & (prop2) est « true » si (prop1) est « true » and (prop2) est « true »,
- (prop1) | (prop2) est « true » si (prop1) est « true » or (prop2) est « true »,
- (prop1) => (prop2) est « true » si (prop1) implique (prop2).

#### 4.4.3. L'opérateur P

L'opérateur principal pour spécifier les propriétés est l'opérateur P. P nous permet de raisonner sur la probabilité qu'un certain type de comportement est observé. Formellement, la propriété :

$$P_{(op)(p)} [(pathprop)]$$

est « true » dans un état  $s$  d'un HMM si « la probabilité qu'une propriété de chemin (pathprop) est satisfaite par les chemins qui commencent par l'état  $s$  atteint la limite (op)(p) ».

Un exemple typique d'une limite sera :

$$P_{>0.5} [(pathprop)]$$

qui signifie : La propriété est « true » dans un état  $s$  d'un HMM si « la probabilité qu'une propriété de chemin (pathprop) est satisfaite par les chemins depuis  $s$  est supérieure à 0.5 ».

#### 4.4.4. Les propriétés de chemins

Comme il est illustré précédemment dans la définition de syntaxe de base, trois types différents de propriétés de chemin peuvent être utilisés à la portée de l'opérateur P :

- « Next » :  $X_{(\Omega)}$ ,
- « Until » : U,
- « bounded Until » :  $U_{(\text{time})}$ .

Une propriété de chemin est évaluée à «true» ou «false» pour un seul chemin dans le modèle. Un chemin dans un HMM est une séquence des états dans le modèle qui sont connectés par des transitions. Considérons un chemin  $\pi = s_0, s_1, s_2, \dots, s_n$ . Dans les sections suivantes, nous décrirons la sémantique de chacun des trois types des propriétés de chemin  $\pi$ .

#### 4.4.5. Les propriétés de chemins « next »

$X_{(\Omega)}$  (prop) :

**Cas 1 :**  $\Omega = \emptyset$  (l'ensemble vide)  $\Rightarrow X_{(\Omega)}(\text{prop}) = X(\text{prop})$

La propriété  $X(\text{prop})$  est « true » pour le chemin  $\pi$  si (prop) est « true » dans l'état  $s_1$ .

Un exemple de ce type de propriété utilisée à la portée de l'opérateur P est :

$$P_{\leq 0.5} [X \ y=1]$$

qui est « true » dans un état  $s$  d'un HMM si « la probabilité que l'expression ( $y=1$ ) soit « true » à l'état prochain est inférieure ou égale à 0.5 ».

**Cas 2:**  $\Omega = O / O \subseteq \Theta \Rightarrow X_{(\Omega)}(\text{prop}) = X_O(\text{prop})$

$X_O(\text{prop})$  signifie que la prochaine observation est O et le chemin suivant satisfait (prop).

Un exemple de ce type de propriété est :

$$P_{\leq 0.5}(X_{O1} \ y=1)$$

qui signifie : « La probabilité que la prochaine observation est  $O_1$  et le chemin suivant satisfait ( $y=1$ ) est inférieure ou égale à 0.5 ».

#### 4.4.6. Les propriétés de chemins « Until »

La propriété  $[(prop1) \text{ U } (prop2)]$  est « true » pour le chemin  $\pi$  si pour un entier  $i$ , ( $prop2$ ) est « true » dans l'état  $s_i$  et ( $prop1$ ) est « true » dans tous les états  $s_j$  pour  $j < i$ , c.-à-d. si ( $prop2$ ) est éventuellement « true » et ( $prop1$ ) est « true » jusqu'à ce point. Une application commune de ce type de propriété est le cas où ( $prop1$ ) est « true ».

Un exemple typique est :

$$P_{\leq 0.5} [\text{true U } y=1]$$

qu'est « true » dans un état si « la probabilité que  $y$  est éventuellement égale à 1 est inférieure ou égale à 0.5 ».

#### 4.4.7. Les propriétés de chemins « Bounded Until »

Les propriétés de chemin « Bounded Until » sont une généralisation des propriétés « Until » où une limite additionnelle est imposée sur le temps du second argument ( $prop2$ ) devant la satisfaire.

Dans une propriété de chemin Bounded Until «  $(prop1) \text{ U}_{(time)} (prop2)$  », la spécification de l'intervalle de temps ( $time$ ) doit être de la forme : «  $\leq (t)$  » où ( $t$ ) est une expression évaluée à un entier non-négatif. La propriété «  $(prop1) \text{ U}_{\leq (t)} (prop2)$  » est « true » pour le chemin  $\pi$  si pour un entier  $i \leq (t)$ , ( $prop2$ ) est « true » dans l'état  $s_i$  et ( $prop1$ ) est « true » dans tous les états  $s_j$  pour  $j < i$ , c.-à-d. si ( $prop2$ ) est « true » avant ( $t$ ) unités de temps et ( $prop1$ ) est (true) jusqu'à ce point ».

Un exemple typique est :

$$P_{\leq 0.5} [\text{true U}_{\leq 5} y=1]$$

qui signifie : « La probabilité que  $y$  soit égale à 1 dans au plus 5 unités de temps est inférieure ou égale à 0.5 ».

#### 4.4.8. Spécification des propriétés

Toutes les propriétés décrites précédemment sont évaluées à une valeur booléenne. Bien que la satisfaction d'une propriété est définie en termes d'un seul état, au moment de l'analyse d'une propriété d'un modèle. Notre HMM\_Model-Checker considère que la propriété soit « true » si elle est satisfaite dans tous les états du modèle, et « false » sinon.

### 4.5. Algorithme pour POCTL sur un HMM

Les algorithmes de model-checking dépendent énormément des classes de modèles choisies. Une fois les formalismes pour les systèmes et propriétés fixés, il faut développer des algorithmes en vue de leur implémentation. Les critères de développement de ces algorithmes sont principalement l'efficacité et la facilité d'implémentation. Bien sûr, l'efficacité est fortement liée à la complexité théorique du problème du model-checking, mais elle ne lui est pas équivalente. De plus, le développement des algorithmes va de pair avec la proposition de structures de données en vue de leur implémentation. Ces structures de données doivent être relativement compactes (pour limiter les problèmes de mémoire) et doivent permettre de réaliser efficacement toutes les opérations apparaissant dans l'algorithme.

La vérification d'une formule  $\phi$  sur un modèle  $H$  consiste en une exploration judicieuse de l'espace d'états basée sur la sémantique des opérateurs logiques formant  $\phi$  [18].

En effet, il ne sert à rien de parcourir tous les états d'un modèle pour retenir ceux qui vérifient une formule. Par exemple, supposons qu'un système  $H$  possède deux étiquettes sur ses états :  $p$  et  $q$ , et que l'on veut vérifier une formule comme  $p$ ,  $\neg q$  et  $p \wedge q$ , alors, nous pouvons parcourir tous les états pour vérifier leurs étiquettes et ainsi déduire s'ils satisfont la formule. Cette méthode est très coûteuse en temps, surtout si l'ensemble des états est infini, Par contre, il est plus judicieux de calculer  $sat(p)$  et  $sat(q)$ , donnés par la fonction d'étiquetage  $L$ , et déduire ensuite quels états satisfont la formule donnée. Alors, les calculs se résument à deux appels à la fonction  $L$  et quelques opérations sur les ensembles (intersection et différence). En somme, cette approche se veut une implémentation directe du calcul de  $sat(\phi)$ , tel qu'il est donné par la sémantique de POCTL.

Dans ce qui suit, nous allons présenter les algorithmes nécessaires pour la vérification [18].

L'algorithme de vérification (figure 4.1) prend en entrée un modèle HMM  $H$  et une formule  $\phi$ , et renvoie l'ensemble des états de  $H$  qui vérifient la formule  $\phi$ . Il procède de manière récursive pour déterminer l'ensemble. Voici en détails l'algorithme  $Sat(H, \phi)$  [18].

**Algorithme  $Sat(HMM H, Formule \phi)$**

**Dépendant de  $\phi$ , Choisir :**

**Cas  $a$**

**Retourner  $Sat(a) : \{ s \in S / a \in L(s) \}$  ;**

**Cas  $\neg \phi_1$**

**Retourner  $S/Sat(\phi_1)$  ;**

**Cas  $\phi_1 \wedge \phi_2$**

**Retourner  $Sat(\phi_1) \wedge Sat(\phi_2)$  ;**

**Cas  $P_{\sim p}(\psi)$**

**Retourner  $\{ s \in S / P_s(\psi) \sim p \}$  ;**

**Fin Choisir ;**

**Fin Algorithme.**

Figure 4.1- Algorithme de Model checking pour POCTL sur HMM

Nous verrons clairement que l'algorithme est une implantation directe de ce qui est décrit dans la sémantique de POCTL vue en chapitre 2.

Les calculs à l'intérieur de l'algorithme utilisent les opérations de base de l'arithmétique des ensembles (intersection et différence). Les trois premiers cas sont directement construits à partir de la sémantique.

Le dernier cas, celui où  $P_{\sim p}(\psi)$ , est plus compliqué puisqu'il faut calculer l'ensemble :

$\{s \in S \setminus P_s(\psi) \sim p\}$ . Pour ce faire, nous utilisons deux sous-procédures :

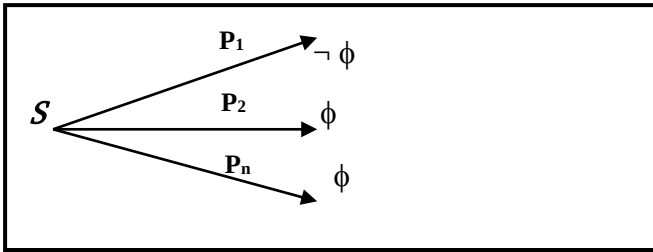
La première, *Calculer Probabilité*  $(s, s')$   $s' \in sat(\phi)$ , retourne une fonction de  $S$  dans  $[0, 1]$  qui donne la probabilité  $P_s(\psi)$ , pour tout  $s \in S$ , la deuxième sous-procédure prend une

fonction et une valeur et retourne l'ensemble des états pour lesquels  $Ps(\psi) \sim p$ . Étant donné que la fonction ici est la fonction de probabilité  $P_{\sim p}(\psi)$ , la sous-procédure retourne l'ensemble des états telle que la probabilité d'aller vers  $\text{sat}(s')$  avec la probabilité  $p$  est indiquée par l'opérateur borné  $\sim p$ .

#### 4.5.1. Le calcul de $P_s(\psi)$

Dépendant de  $\psi$ , choisir :

- **Cas  $X_\Omega \phi$**



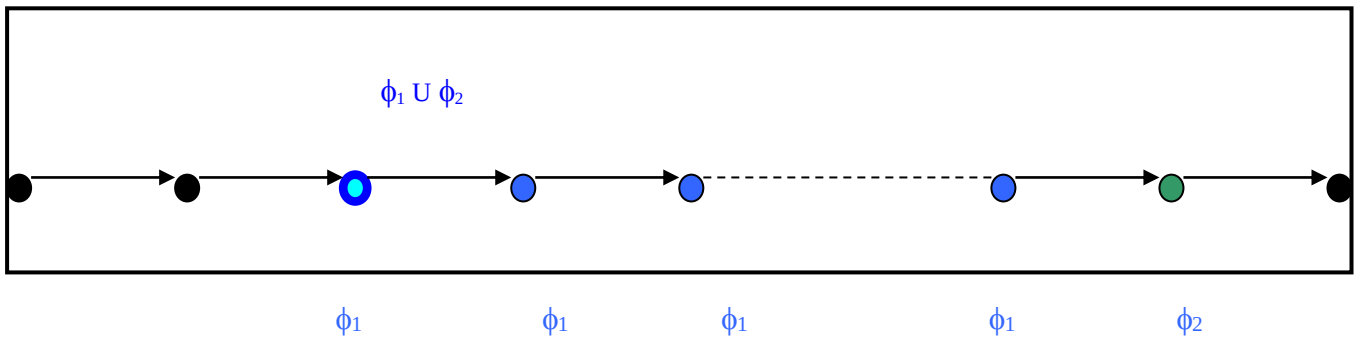
Pour tout  $s \in S$ , il est facile de montrer que :

$$p_s(X_\Omega \phi) = \mu_s(\Omega) \cdot \sum_{s' \in \text{sat}(\phi)} p(s, s').$$

Tq  $\Omega \subseteq \Theta$ , un sous ensemble d'observations :

$$\mu_s(\Omega) = \sum_{o \in \Omega} \mu_s(o) \text{ et } \text{sat}(\phi) = \{s \in S \mid s \models \phi\}.$$

- **Cas  $\phi_1 U \phi_2$**



Ici il faut regrouper l'ensemble de tous les états de l'HMM en 3 sous-ensembles disjoints définis comme suit :

$$S^{\text{yes}} = \text{sat}(\phi_2).$$

$$S^{\text{no}} = S / (\text{sat}(\phi_1) \wedge \text{sat}(\phi_2).)$$

$$S^? = S / (S^{\text{yes}} \cup S^{\text{no}})$$

Pour tout  $s \in S$  faire :

$$x_s = \begin{cases} 0 & \text{si } s \in S^{\text{no}} \\ 1 & \text{si } s \in S^{\text{yes}} \\ \sum_{s' \in S} p(s, s') \cdot x_{s'} & \text{si } s \in S^? \end{cases}$$

```

Prob0(sat( $\phi_1$ ),sat( $\phi_2$ ));
  R := sat( $\phi_2$ );
  done := false;
  while (done = false);
    R' := R  $\cup$  {s  $\in$  sat( $\phi_1$ ) /  $\exists s' \in$  R. p(s, s') > 0 };
    If (R' = R) then done := true;
    R := R';
  end while;
return S / R;

```

Figure 4.2- Algorithmme Prob0

```

Prob1 ( $\text{sat}(\phi_1), \text{sat}(\phi_2), S^{\text{no}}$ );
     $R := S^{\text{no}}$ ;
    Done := false;
    While (done = false);
         $R' := RU\{s \in (\text{sat}(\phi_1) \setminus (\text{sat}(\phi_2))) / \exists s' \in R. P(s, s') > 0\}$ ;
        If ( $R' = R$ ) then done := true;
         $R := R'$ ;
    End while;
    Return  $S/R$ ;

```

Figure 4.3- Algorithme Prob1

- **Prob0** (figure 4.2) calcule tous les états dans lesquels il est possible (avec une probabilité  $\neq 0$ ) d'atteindre des états satisfaisants  $\phi_2$  sans laisser les états satisfaisant  $\phi_1$ , il retranche alors ceux-ci de  $S$  pour déterminer les états qui ont une probabilité = 0.
- **Prob1** (figure 4.3) calcule l'ensemble  $S^{\text{yes}}$  de la même façon que prob0 en déterminant d'abord l'ensemble des états pour lesquels la probabilité est inférieure à 1, ce sont les états ayant une probabilité ( $\neq 0$ ) d'atteindre un état de  $S^{\text{no}}$  qui contient les états satisfaisant  $\phi_1$  mais pas  $\phi_2$ .

Ces deux algorithmes forment la première partie de calcul  $p_s(\phi_1 \cup \phi_2)$ . Pour cette raison, nous leur faisons référence comme algorithmes de pré-calcul.

Il devrait être noté que pour les propriétés ayant  $p$  dans l'opérateur  $P_{\sim p}(\psi)$  à 0 ou 1, ou pour le cas où  $p_s(\phi_1 \cup \phi_2)$  arrive à être égal à 0 ou 1 pour tous les états (c.-à-d.  $S^{\text{no}} \cup S^{\text{yes}} = S$ ), il suffit d'utiliser ces algorithmes.

Pour les propriétés avec un lien arbitraire  $\sim p$ , le calcul numérique est exigé habituellement, les algorithmes de pré-calcul sont encore précieux, en particulier dès qu'ils déterminent la

probabilité exacte pour les états  $S^{no}$ ,  $S^{yes}$ . Alors que le calcul numérique typiquement calcule seulement une approximation et peut être soumis aussi à des erreurs d'arrondi.

Nous avons besoin encore de calculer  $p_s(\phi_1 \cup \phi_2)$  pour les états restants

$$S^? = S \setminus (S^{no} \cup S^{yes}).$$

Cela peut être fait en résolvant le système d'équations linéaires dont les variables  $\{x_s \mid s \in S\}$

$$\text{et } x_s = p_s(\phi_1 \cup \phi_2).$$

$$x_s = \begin{cases} 0 & \text{si } s \in S^{no} \\ 1 & \text{si } s \in S^{yes} \\ \sum_{s' \in S} p(s, s') \cdot x_{s'} & \text{si } s \in S^? \end{cases}$$

Nous pouvons le réécrire de façon traditionnelle :  $A \cdot x = b$  tel que :

- $A = I - P'$  où  $I$  est la matrice d'identité, et  $P'$  est donné par :

$$p'(s, s') = \begin{cases} p(s, s') & \text{si } s \in S \\ 0 & \text{sinon} \end{cases}$$

- $b$  : est un vecteur colonne sur les états avec :

$$b(s) = \begin{cases} 1 & \text{si } s \in S^{yes}. \\ 0 & \text{sinon.} \end{cases}$$

Le système  $A \cdot x = b$  peut être résolu par toutes les approches standards [53], celles-ci incluent des méthodes directes, telles que l'élimination Gaussienne ou les méthodes itératives, telle que : Jacobi et Gauss Seidel [53].

### 4.5.2. Exemples explicatifs

Pour mieux comprendre les algorithmes précédents, nous proposons les deux exemples suivants : dans chaque exemple, nous présentons un modèle HMM ensuite nous présentons des exemples simples de formules POCTL correctement construites, puis nous allons les vérifier sur ce modèle en s'appuyant sur les algorithmes vus précédemment.

#### - Exemple 1

Soit un système probabiliste représenté par le modèle HMM H1 illustré par la figure 4.4:

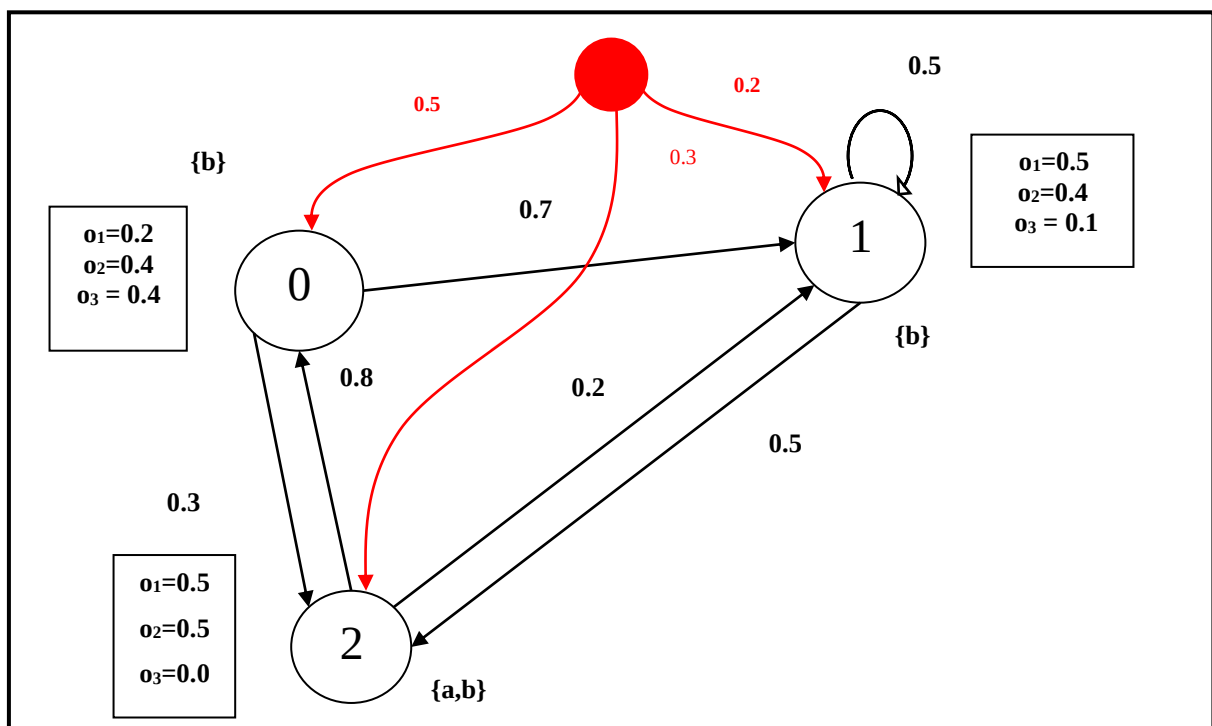


Figure 4.4 – Le modèle HMM de l'exemple explicatif 1

- HMM H1 ( $S, P, L, \theta, \mu, \pi$ ).
- $S = \{S_0, S_1, S_2\}$  est l'ensemble des états.
- La matrice de probabilité  $P$  représente les transitions entre les états, elle est définie comme suit (figure 4.5) :

$$\mathbf{P} = \begin{matrix} & \begin{matrix} \mathbf{S}_0 & \mathbf{S}_1 & \mathbf{S}_2 \end{matrix} \\ \begin{matrix} \mathbf{S}_0 \\ \mathbf{S}_1 \\ \mathbf{S}_2 \end{matrix} & \begin{pmatrix} 0 & 0.7 & 0.3 \\ 0 & 0.5 & 0.5 \\ 0.8 & 0.2 & 0 \end{pmatrix} \end{matrix}$$

Figure 4.5- La matrice P de l'exemple explicatif 1

- L : est une fonction associée à chaque état s, L(s) l'ensemble des propositions atomiques vérifiées par s : L(0) = {b}, L(1) = {b}, L(2) = {a, b}. Elle est représentée par la matrice L définie comme suit (figure 4.6) :

$$\mathbf{L} = \begin{matrix} & \begin{matrix} \mathbf{a} & \mathbf{b} \end{matrix} \\ \begin{matrix} \mathbf{S}_0 \\ \mathbf{S}_1 \\ \mathbf{S}_2 \end{matrix} & \begin{pmatrix} 0 & 1 \\ 0 & 1 \\ 1 & 1 \end{pmatrix} \end{matrix}$$

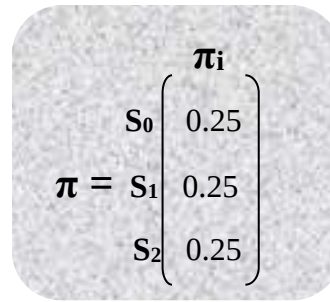
Figure 4.6- La matrice L de l'exemple explicatif 1

- $\theta = \{o_1, o_2, o_3\}$  : l'ensemble d'observations représenté par la matrice  $\mu$ , dite matrice de distribution des probabilités d'observations, définie comme suite (figure 4.7) :

$$\mathbf{B} = \begin{matrix} & \begin{matrix} \mathbf{o}_1 & \mathbf{o}_2 & \mathbf{o}_3 \end{matrix} \\ \begin{matrix} \mathbf{S}_0 \\ \mathbf{S}_1 \\ \mathbf{S}_2 \end{matrix} & \begin{pmatrix} 0.2 & 0.4 & 0.4 \\ 0.5 & 0.4 & 0.1 \\ 0.5 & 0.5 & 0 \end{pmatrix} \end{matrix}$$

Figure 4.7- La matrice  $\mu$  de l'exemple explicatif 1

- $\pi$  est le vecteur de distribution des probabilités de transitions initiales (figure 4.8) :



$$\pi = \begin{matrix} s_0 \\ s_1 \\ s_2 \end{matrix} \begin{pmatrix} 0.25 \\ 0.25 \\ 0.25 \end{pmatrix}$$

Figure 4.8- Le vecteur  $\pi$  de l'exemple explicatif 1

Nous présentons des exemples simples de formules POCTL correctement construites, puis nous allons les vérifier sur ce modèle en s'appuyant sur les algorithmes vus précédemment.

Soit les formules POCTL :  $(b)$ ,  $(\neg b)$ ,  $(a \wedge b)$ ,  $P_{<0.5}(Xo_2 b)$ .

#### - La formule $(b)$

Cette formule est le cas le plus simple qui représente une proposition atomique.

D'après l'algorithme de la figure 4.4, nous avons :  $\text{Sat}(b) = \{s \in S / b \in L(s)\}$ .

Donc nous allons calculer l'ensemble  $\text{Sat}(b)$ , si le résultat de calcul nous donne l'ensemble total des états du modèle  $S$  alors nous déduisons que le modèle vérifie la formule.

Nous avons:  $L(s_0) = \{b\}$ ,  $L(s_1) = \{b\}$ ,  $L(s_2) = \{a, b\}$ .

$\text{Sat}(b) = \{s_0, s_1, s_2\} = S$ . Donc le modèle vérifie la formule  $(b)$ .

#### - La formule $(\neg b)$

Cette formule représente le cas de négation d'une formule (Cas :  $\neg \phi_1$  de l'algorithme). Ici nous avons  $\phi_1$  est la formule  $(b)$ .

D'après l'algorithme, nous avons: l'ensemble des états qui satisfont la formule de négation de la formule  $(b)$  est l'ensemble  $S/\text{Sat}(b)$  ;

Nous avons :  $\text{Sat}(b) = \{s_0, s_1, s_2\} = S$ . Donc l'ensemble  $S/\text{Sat}(b) = S/S = \{\}$ , donc le modèle n'a pas vérifié la formule  $(\neg b)$  car tous les états du modèle ne satisfont pas la formule  $(\neg b)$ .

### - La formule $(a \wedge b)$

Cette formule représente le cas de conjonction de deux formules (Cas :  $\phi_1 \wedge \phi_2$  de l'algorithme). Ici nous avons  $\phi_1$  est la formule (a) et  $\phi_2$  est la formule (b).

$$\begin{aligned} \text{Nous avons : } \text{Sat}(\phi_1 \wedge \phi_2) &= \text{Sat}(\phi_1) \wedge \text{Sat}(\phi_2) \Rightarrow \text{Sat}(a \wedge b) = \text{Sat}(a) \wedge \text{Sat}(b). \\ &= \{s_2\} \wedge \{s_0, s_1, s_2\} = \{s_2\}. \end{aligned}$$

Donc nous concluons que la formule  $(a \wedge b)$  est satisfaite dans l'état  $s_2$ .

### - La formule $P_{<0.5}(X_{O_2} b)$

Nous savons d'après ce que nous avons dit précédemment que :

$$p_s(X_{\Omega} \phi) = \mu_s(\Omega) \cdot \sum_{s' \in \text{sat}(\phi)} p(s, s').$$

Donc nous allons calculer la satisfaction de cette formule dans chaque état. Si le résultat de vérification donne l'ensemble total des états du modèle alors nous concluons que le modèle vérifie la formule.

- La vérification de la formule  $P_{<0.5}(X_{O_2} b)$  dans l'état  $s_0$ :

$$\begin{aligned} P_{s_0}(X_{O_2}(b)) &= \mu_{s_0}(O_2) \cdot \sum_{s' \in \text{sat}(b)} p(s_0, s'). \\ &= \mu_{s_0}(O_2) (p(s_0, s_1) + p(s_0, s_2)) \\ &= 0.4 (0.7 + 0.3) = \mathbf{0.4} \end{aligned}$$

- Nous remarquons que  $\mathbf{0.4} < 0.5$  donc  $s_0 \models (X_{O_2} b)$  (1)

- La vérification de la formule  $P_{<0.5}(X_{O_2} b)$  dans l'état  $s_1$ :

$$\begin{aligned} P_{s_1}(X_{O_2} b) &= \mu_{s_1}(O_2) \cdot \sum_{s' \in \text{sat}(b)} p(s_1, s'). \\ &= \mu_{s_1}(O_2) (p(s_1, s_1) + p(s_1, s_2)) \\ &= 0.3 * (0.5 + 0.5) = \mathbf{0.3} \end{aligned}$$

- Ici, nous remarquons aussi que  $\mathbf{0.3} < 0.5$  donc  $s_1 \models (X_{O_2} b)$  (2)

- La vérification de la formule  $P_{<0.5} (X o_2 \ b)$  dans l'état  $s_2$  :

$$\begin{aligned}
 P_{s_2} (X o_2 \ b) &= \mu_{s_2} (o_2) \cdot \sum_{s' \in \text{sat}(b)} p(s_2, s') \\
 &= \mu_{s_2} (o_2) (p(s_2, s_0) + p(s_2, s_1)) \\
 &= 0 \cdot (0.8 + 0.2) \\
 &= \mathbf{0}
 \end{aligned}$$

- Comme dans les deux résultats précédentes  $\mathbf{0} < 0.5$  donc  $s_2 \models (X o_2 \ b)$  (3)

De (1), (2) et (3) nous concluons que  $\text{Sat}(X o_2 \ b) = \{s_0, s_1, s_2\} = S$  alors la formule " $X o_2 \ b$ " est vérifiée dans tout le modèle.

## - Exemple 2

Soit un système représenté par le modèle HMM illustré par la figure 4.9 :

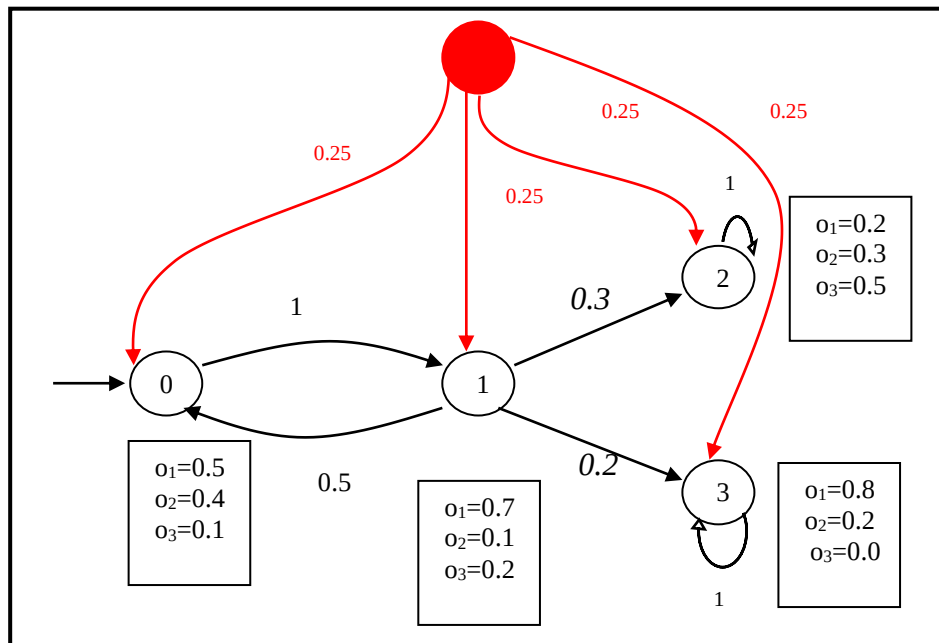


Figure 4.9– Le modèle HMM de l'exemple explicatif 2

- HMM  $H_2 (S, P, L, \theta, \mu, \pi)$ .
- $S = \{s_0, s_1, s_2, s_3\}$  est l'ensemble des états.

- La matrice de probabilités représente les transitions entre les états, elle est définie comme suit (figure 4.10) :

$$\mathbf{P} = \begin{matrix} & \begin{matrix} S_0 & S_1 & S_2 & S_3 \end{matrix} \\ \begin{matrix} S_0 \\ S_1 \\ S_2 \\ S_3 \end{matrix} & \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0.5 & 0 & 0.3 & 0.2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{matrix}$$

Figure 4.10- La matrice P de l'exemple explicatif 2

- L : est une fonction associée à chaque état s, L(s) l'ensemble des propositions atomiques vérifiées par s : L(0) = { }, L(1) = { }, L(2) = {a} , L(3) = {b}. Elle est représentée par la matrice L définie comme suit (figure 4.11) :

$$\mathbf{L} = \begin{matrix} & \begin{matrix} a & b \end{matrix} \\ \begin{matrix} S_0 \\ S_1 \\ S_2 \\ S_3 \end{matrix} & \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} \end{matrix}$$

Figure 4.11- La matrice L de l'exemple explicatif 2

- $\theta = \{o_1, o_2, o_3\}$  : l'ensemble des observations représenté par la matrice  $\mu$ , dite de distribution des probabilités d'observations, définie comme suit (figure 4.12) :

$$\mathbf{B} = \begin{matrix} & \begin{matrix} \mathbf{o}_1 & \mathbf{o}_2 & \mathbf{o}_3 \end{matrix} \\ \begin{matrix} \mathbf{s}_0 \\ \mathbf{s}_1 \\ \mathbf{s}_2 \\ \mathbf{s}_3 \end{matrix} & \begin{pmatrix} 0.5 & 0.4 & 0.1 \\ 0.7 & 0.1 & 0.2 \\ 0.8 & 0.2 & 0 \\ 0.2 & 0.3 & 0.5 \end{pmatrix} \end{matrix}$$

Figure 4.12- La matrice  $\mu$  de l'exemple explicatif 2

- $\pi$  est le vecteur de distribution des probabilités de transitions initiales (figure 4.13) :

$$\pi = \begin{matrix} & \pi_i \\ \begin{matrix} \mathbf{s}_0 \\ \mathbf{s}_1 \\ \mathbf{s}_2 \\ \mathbf{s}_3 \end{matrix} & \begin{pmatrix} 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \end{pmatrix} \end{matrix}$$

Figure 4.13- Le vecteur  $\pi$  de l'exemple explicatif 2

Nous présentons des exemples simples de formules POCTL correctement construites, puis nous allons les vérifier sur ce modèle en s'appuyant sur les algorithmes vus précédemment.

Soit les formules POCTL :  $P_{\geq 0.5} (\neg b \text{ U } a)$ ,  $P_{< 0.7} (X o_1 \neg b)$ .

#### - La formule $P_{\geq 0.5} (\neg b \text{ U } a)$

La vérification de cette formule nécessite d'abord le calcul de  $p_s (\neg b \text{ U } a)$  pour les états  $s \in \{s_0, s_1, s_2, s_3\}$ . Nous avons :  $a$  est vraie dans l'état  $s_2$  et ni  $a$ , ni  $\neg b$  sont vrais dans l'état  $s_3$  donc  $S^{\text{no}} = \{3\}$  et  $S^{\text{yes}} = \{2\}$  et les probabilités pour ces 2 états sont 0 et 1 respectivement .

Sur cet exemple, les algorithmes de pré-calcul Prob0 et Prob1 n'identifient pas les états supplémentaires.

Les probabilités des états restants peuvent être calculées en résolvant le système d'équations linéaires suivant :

$$x_0 = x_1.$$

$$x_1 = 0.5x_0 + 0.3x_2 + 0.2x_3.$$

$$x_2 = x_1.$$

$$x_3 = x_0.$$

avec :  $p_s(\neg b \cup a) = x$

Cela produit la solution :  $(x_0, x_1, x_2, x_3) = (0.6, 0.6, 1, 0)$

Nous verrons que la formule  $P_{\geq 0.5}(\neg b \cup a)$  est satisfaite dans les états :  $\{0, 1, 2\}$ . Nous concluons que  $\text{Sat}(\neg b \cup a) = \{s_0, s_1, s_2\} = S$  alors la formule " $x_{01} \neg b$ " est vérifiée dans tout le modèle.

#### - La formule $P_{<0.9}(x_{01} \neg b)$

s satisfait  $P_{<0.7}(x_{01} \neg b)$  ssi  $p_s(x_{01} \neg b) < 0.7$ .

Nous avons :  $p_s(x_{\Omega} \phi) = \mu_s(\Omega) \cdot \sum_{s' \in \text{sat}(\phi)} p(s, s')$ .

$$\begin{aligned} p_{s0}(x_{01} \neg b) &= \mu_{s0}(o_1) \cdot \sum_{s' \in \text{sat}(\neg b)} p(s, s'). \text{ tq: } s' = \{s_1\}. \\ &= 0.5 (1) = \mathbf{0.5} \end{aligned}$$

$$\begin{aligned} p_{s1}(x_{01} \neg b) &= \mu_{s1}(o_1) \cdot \sum_{s' \in \text{sat}(\neg b)} p(s, s'). \text{ tq: } s' = \{s_0, s_2\}. \\ &= 0.7 (0.5 + 0.3) = \mathbf{0.56} \end{aligned}$$

$$\begin{aligned} p_{s2}(x_{01} \neg b) &= \mu_{s2}(o_1) \cdot \sum_{s' \in \text{sat}(\neg b)} p(s, s'). \text{ tq: } s' = \{s_2\}. \\ &= 0.8 (1) = \mathbf{0.8} \end{aligned}$$

$$\begin{aligned} p_{s3}(x_{01} \neg b) &= \mu_{s3}(o_1) \cdot \sum_{s' \in \text{sat}(\neg b)} p(s, s'). \text{ tq: } s' = \{ \}. \\ &= \mathbf{0} \end{aligned}$$

Nous verrons que la formule  $P_{<0.7}(x_{01} \neg b)$  est satisfaite dans les états :  $\{0, 1, 2, 3\}$ . Nous concluons que  $\text{Sat}(x_{01} \neg b) = \{s_0, s_1, s_2, s_3\} = S$  alors la formule " $x_{01} \neg b$ " est vérifiée dans tout le modèle.

## 4.6. Vue d'ensemble sur HMM\_Model-Checker

Notre objectif est d'élaborer un outil de model-checking probabiliste correspondant aux HMMs. Rappelons que pour atteindre cet objectif - et comme pour tout model-checker -, il faut disposer d'un modèle formel du système à réaliser sur lequel nous pourrions appliquer des méthodes de vérification. Donc, ce type d'approches nécessite trois ingrédients (figure 4.14) :

- une description formelle du comportement du système (*Modèle HMM*).
- un langage de spécification permettant d'exprimer les propriétés du système que l'on souhaite vérifier (*Logique POCTL*).
- une procédure de décision qui permet de contrôler la conformité entre la description opérationnelle du système et sa spécification, c'est-à-dire une procédure qui permet de vérifier que le système satisfait effectivement les propriétés que l'on attend de lui (*Algorithme de model ckecking pour POCTL*).

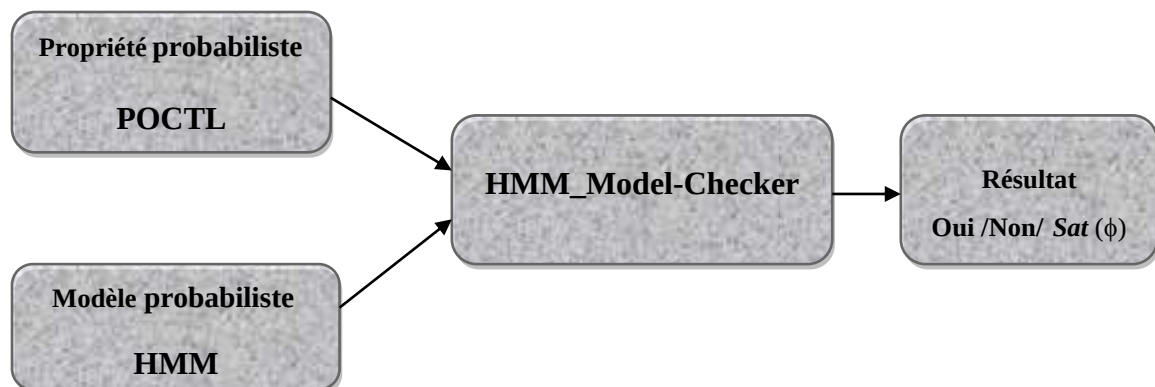


Figure 4.14- Vue d'ensemble de HMM\_Model-Checker

## 4.7. Architecture du HMM\_Model-Checker

Notre HMM\_Model-Checker [66], illustré par la figure 4.15, est un ensemble de techniques algorithmiques qui permettent de vérifier des propriétés décrivant le comportement des systèmes, nous sommes basés sur une modélisation des systèmes à vérifier en HMM, une formalisation des propriétés en formules logiques POCTL et sur des algorithmes de parcours de ces HMMs.

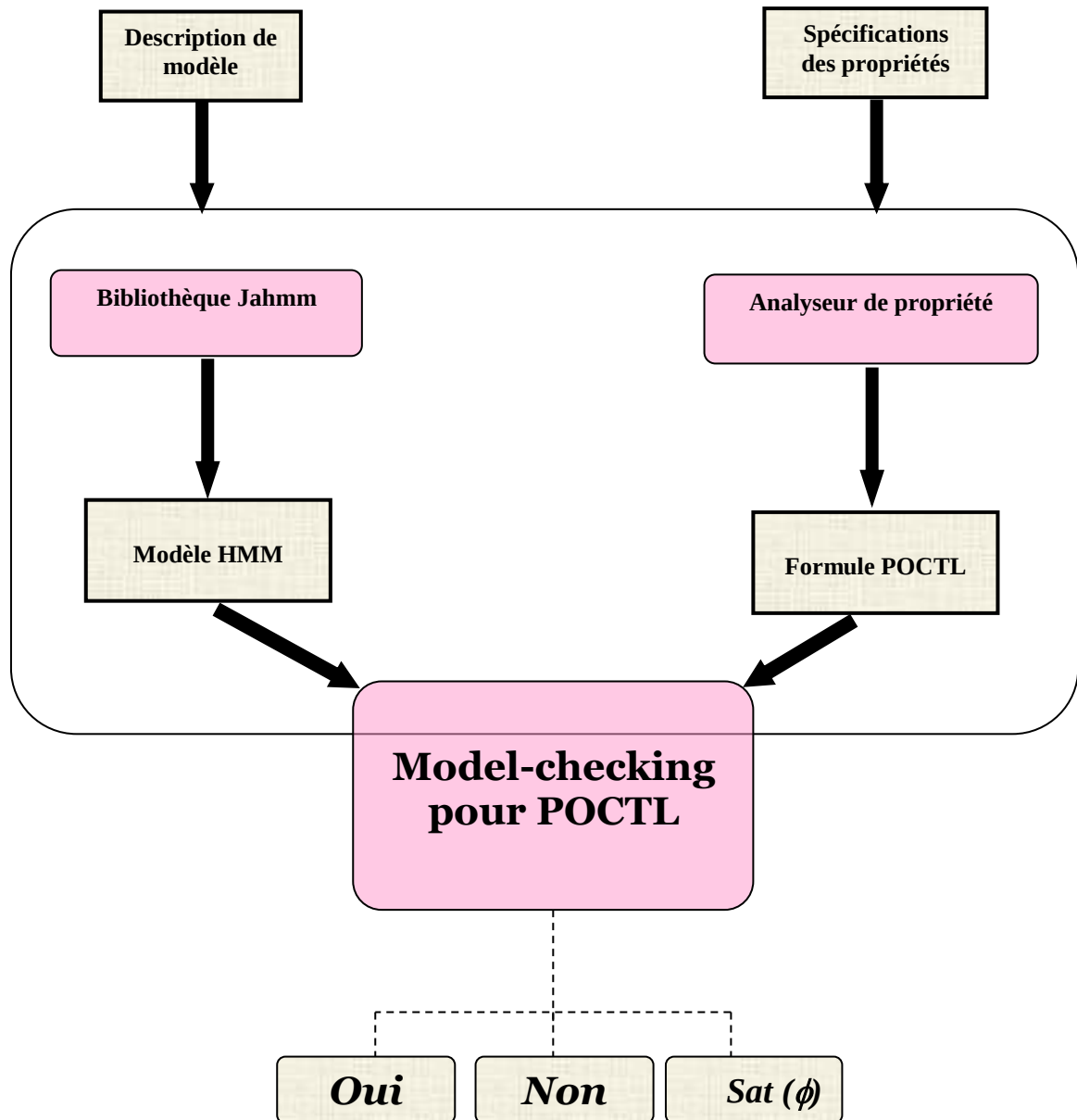


Figure 4.15- Architecture du HMM\_Model-Checker

## 4.8. Environnement de développement

L'implémentation de notre vérificateur a été réalisée en utilisant les outils suivants :

- La plate forme système Microsoft Windows 7 Professionnel (64bits), car sa structure nous offre plusieurs avantages notamment la facilité de manipulation des images et des fichiers, la puissance, la fiabilité, etc. ce qui rend les utilisateurs aisés par son interface.

- L'environnement Netbeans: l'implémentation est faite dans l'environnement Netbeans. Nous avons opté pour cet environnement la construction de notre système car il offre une certaine souplesse de programmation du fait de ses avantages : auto-complétion, léger, rapide, prise en main facile, facile à configurer, les librairies intégrés, la gestion des projets.

## 4.9. Matériel utilisé

Pour la réalisation de notre vérificateur, nous avons utilisé un PC ayant la configuration donnée dans le tableau 4.1.

|                   |                                      |
|-------------------|--------------------------------------|
| <b>Marque</b>     | HP                                   |
| <b>Processeur</b> | Intel® Core™ i5-6200U CPU @ 2,30 GHz |
| <b>RAM</b>        | 8GO                                  |
| <b>Disque Dur</b> | 1TO                                  |

Tableau 4.1- Caractéristiques du matériel utilisé

## 4.10. Description générale du HMM\_Model-Checker

Un vérificateur formel est un programme complexe pouvant rassembler plusieurs fonctionnalités. Dans cette section, nous allons présenter celles que nous avons choisies comme objectifs pour arriver à bâtir notre implémentation. Voici les fonctionnalités retenues pour l'implémentation :

- 1) Charger un HMM  $H$  à partir d'un fichier.
- 2) Créer un HMM  $H$ .
- 3) Faire l'apprentissage des paramètres d'un modèle HMM  $H$ . À partir d'un modèle donné *a priori* et d'observations supposées émises par ce modèle, nous cherchons les

probabilités de transition et d'émission maximisant la vraisemblance des observations.

- 4) Sauvegarder un HMM  $H$  dans un fichier.
- 5) Charger une séquence d'observations à partir d'un fichier.
- 6) Générer une séquence d'observations.
- 7) L'évaluation de la probabilité que la suite des observations ait été émise par un modèle. Lorsque plusieurs modèles existent, cette évaluation permet le choix du modèle le plus probable.
- 8) Sauvegarder une séquence d'observations dans un fichier.
- 9) La recherche de la séquence d'états d'un modèle ayant produit la séquence des observations, c'est-à-dire la recherche de la séquence cachée de plus forte probabilité et qui explique mieux ces observations.
- 10) La saisie à l'écran d'une formule de la logique POCTL.
- 11) Aider l'utilisateur à saisir la formule POCTL correctement et rapidement, en lui affichant une information concernant la syntaxe de la formule et lui donnant des boutons permettant d'afficher automatiquement certains opérateurs de la logique constituant la formule comme : Until, Next, And, ...
- 12) Afficher des messages d'erreurs si la formule saisie ne respecte pas la syntaxe de base de la propriété POCTL que nous avons donné dans le chapitre précédent.
- 13) Saisir à l'écran un commentaire sur la propriété à vérifier.
- 14) Le calcul de  $sat(\phi)$ , l'ensemble des états qui satisfont la formule.
- 15) La vérification de  $H \models \phi$  :
  - Si  $sat(\phi) = S$  Alors retourner OUI ;
  - sinon si  $sat(\phi) = \emptyset$  Alors retourner NON ;
  - Sinon retourner  $sat(\phi)$ .

Avec les fonctionnalités énumérées plus haut et les algorithmes nécessaires à la vérification, nous avons choisi de développer le programme selon le schéma suivant (figure 4.16) :

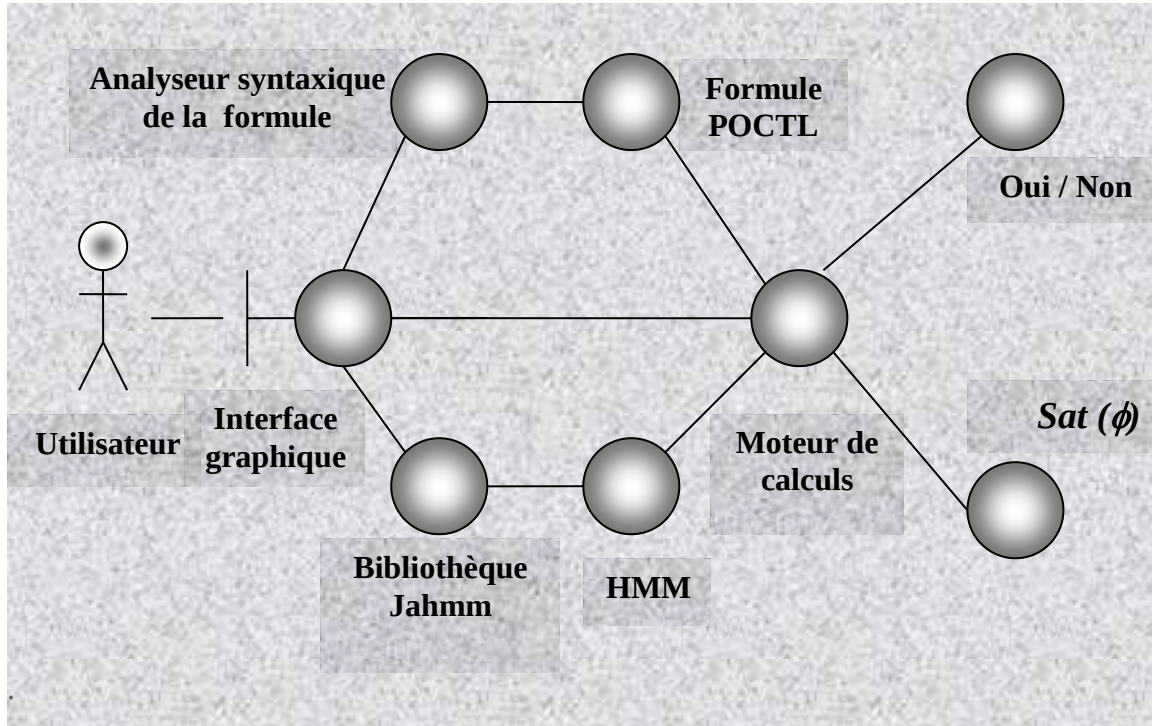


Figure 4.16- Schéma de développement du HMM\_Model-Checker

Donc, l'utilisateur doit utiliser l'interface graphique pour entrer :

- Son système par le chargement d'un fichier contenant le modèle de ce système.
- Sa formule par la saisie de celle-ci dans une boîte de discussion à l'écran sous forme d'une chaîne de caractères.

S'il n'y a aucune erreur de syntaxe dans la formule, deux entrées seront alors données au moteur de calculs afin de commencer la vérification. Une fois cette dernière terminée, le moteur de calculs envoie un message à l'interface graphique déclarant si le système probabiliste vérifie la formule ou non.

## 4.11. Présentation du HMM\_Model-Checker

Notre HMM\_Model-Checker est un ensemble de techniques algorithmiques qui permettent de vérifier des propriétés décrivant le comportement des systèmes, nous avons fondé pour réaliser ceci sur une modélisation des systèmes à vérifier en HMM, une formalisation des propriétés en formules logiques POCTL et sur des algorithmes de parcours de ces HMMs.

L'utilisation de notre outil permet à partir d'un HMM et d'une propriété POCTL désirée, d'une part de détecter systématiquement, s'il existe des exécutions du modèle qui violent cette propriété. Si au moins l'une de ces exécutions existe (c.à.d. si la propriété n'est pas respectée dans tous les états du système), nous déduisons que le modèle ne satisfait pas la propriété donnée. D'autre part, de renvoyer le cas échéant, l'ensemble des états de S qui vérifient la formule  $\phi$ .

### 4.11.1. L'interface graphique

En exécutant l'application, l'interface illustrée dans la figure 4.17 s'apparue à l'écran [66].

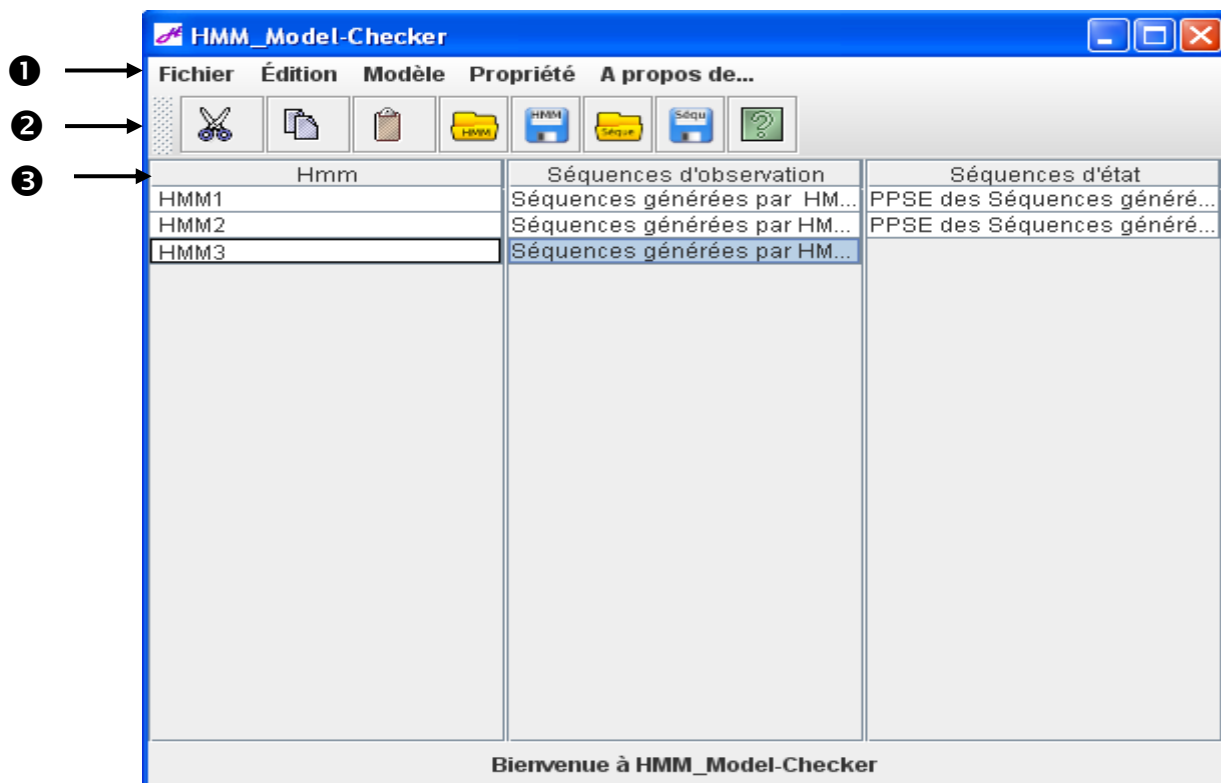


Figure 4.17- L'interface graphique

- ❶ La barre des menus (figure 4.18).



Figure 4.18- Barre des menus

- ❷ La barre d'outils (figure 4.19).



Figure 4.19- Barre d'outils

- ❸ Les listes des fenêtres ouvertes (qui représentent un HMM, un ensemble de séquences des observations ou une séquence d'états). Les listes peuvent être utilisées pour modifier un nom de fenêtre.

### 4.11.2. Fonctionnement du système

1)- La barre de menus illustrée par ❶ comporte cinq menus : Fichier, Édition, Modèle, Propriété, À propos de, en voici une description :

- **Fichier** : Ce menu possède l'action Quitter qui permet de fermer l'outil (figure 4.20).



Figure 4.20- Le menu Fichier

- **Édition** : Ce menu possède les actions d'édition habituelles : Couper, Copier, Coller, Supprimer, Sélectionner tout (figure 4.21).



Figure 4.21- Le menu Édition

- **Modèle** : Ce menu possède trois sous menus : HMM, Séquences-observations, Séquence-états (figure 4.22).

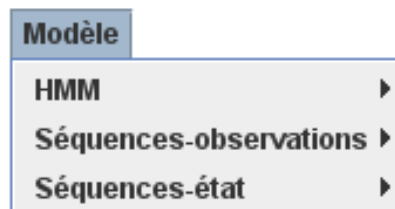


Figure 4.22- Le menu Modèle

Voici une description de ces trois sous menus:

- ❖ **HMM** : possède les actions : Ouvrir HMM, Créer HMM, Apprentissage, Sauvegarder HMM (figure 4.23).



Figure 4.23- Le sous menu HMM

- **Ouvrir HMM** : Lit un HMM à partir d'un fichier (.hmm) (figure 4.24).

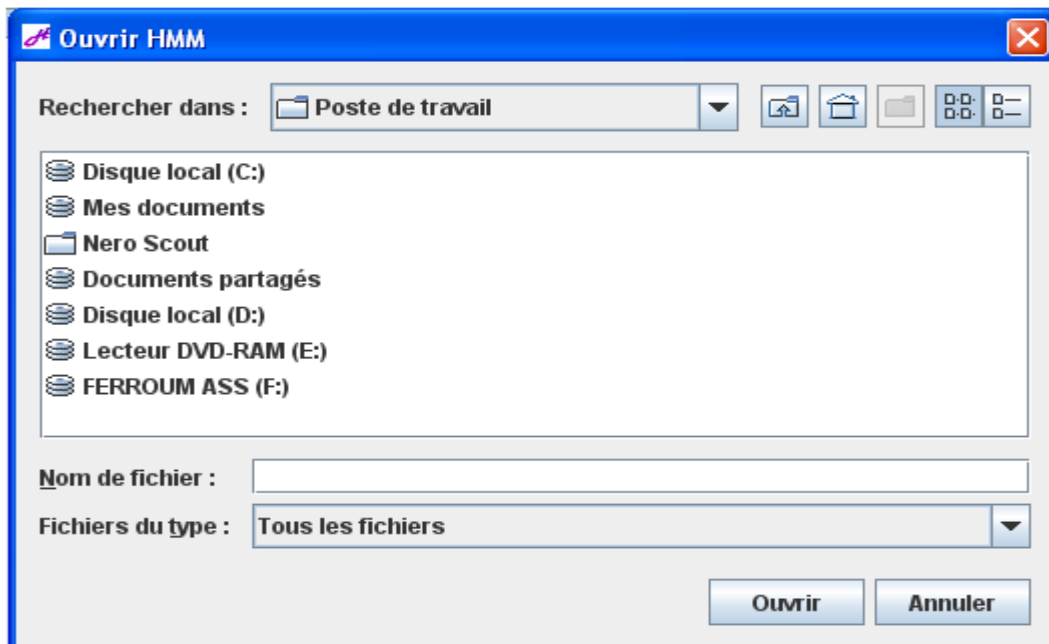


Figure 4.24- La fenêtre Ouvrir HMM

- **Créer HMM** : permet de construire un nouveau HMM. Une boîte de dialogue demande son nombre des états et le type d'observations sur lesquelles il opère (figure 4.25).

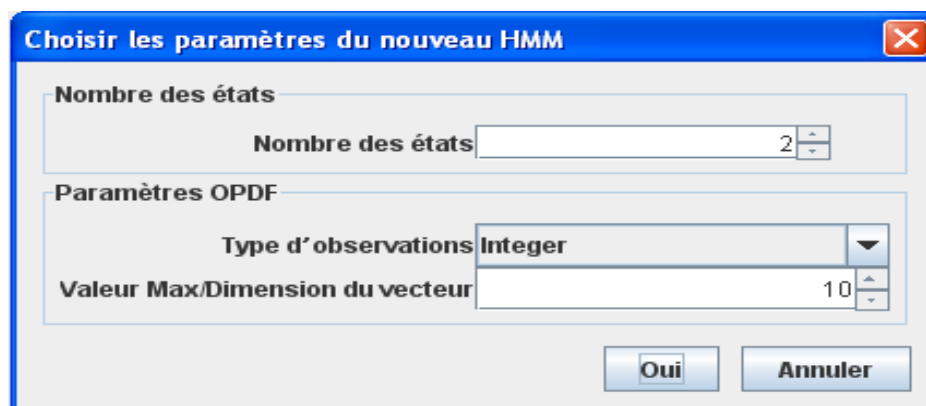


Figure 4.25- La boîte de dialogue pour choisir les paramètres du nouveau HMM

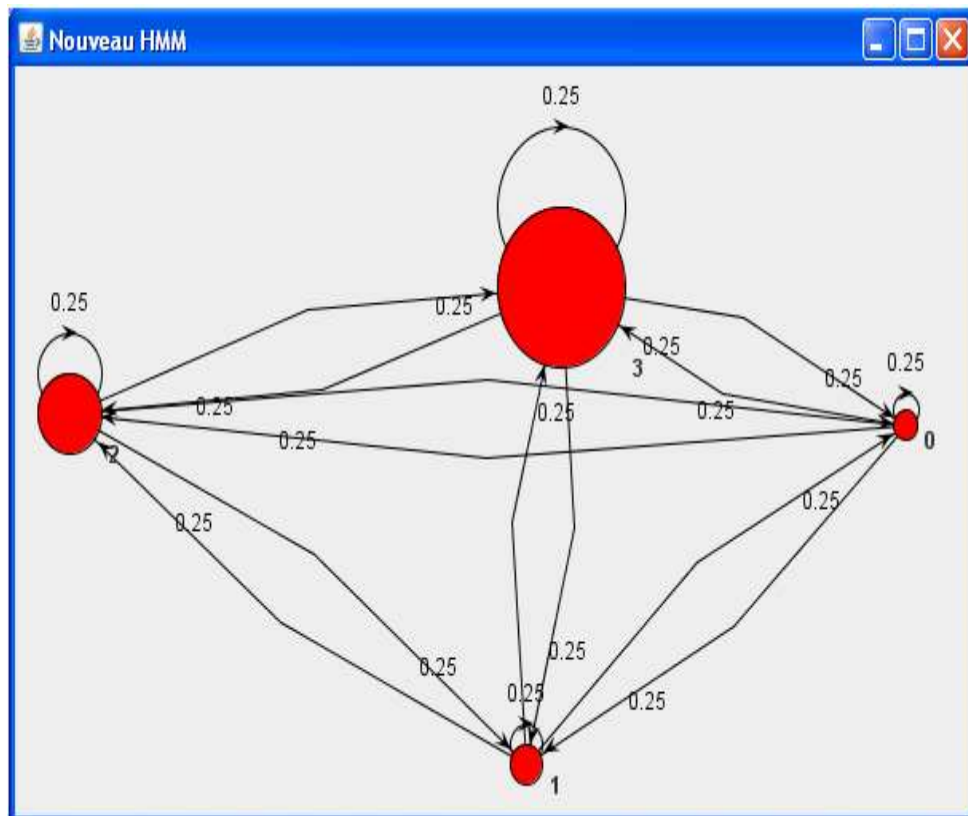


Figure 4.26- Exemple d'un nouveau HMM avec quatre états

La figure 4.26 montre un HMM composé de 4 états. La taille des états (cercles rouges) donne la valeur de son probabilité initiale ( $\pi_i$ ); l'état initial est le plus probable, donc l'état 3.

Pour voir les informations associées à chaque état (valeur de  $\pi_i$  et fonction de distribution des probabilités des observations), nous mettons le curseur quelques secondes sur un état, une info-bulle apparaît.

La syntaxe FDPO pour un FDPO discrète pourrait ressembler: FDPO [0.1 0.5 0.6 0.7], dans cet exemple, la probabilité de l'observation "0" est "0.1", "1" est "0.5", "2" est "0.6", et "3" est "0.7". Un exemple de distribution gaussienne multi-variée est: FDPO [moyenne: [1. 2.]], seule la valeur moyenne de la distribution gaussienne est donnée, dans cet exemple, il est un vecteur 2-dimensionnel, sa valeur première dimension est «1» et le second est "2".

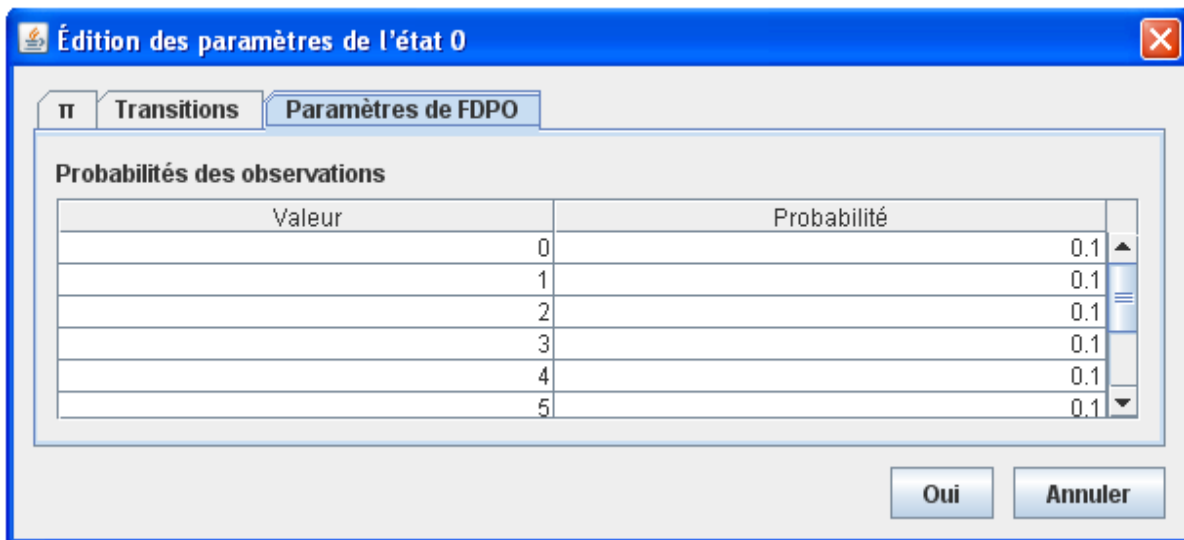


Figure 4.27- La boîte de dialogue donnant les informations associées à un état

Cette boîte de dialogue (figure 4.27) apparaît quand on clique sur un état HMM. Il est permis de modifier la probabilité initiale ( $\pi_i$ ) de cet état, la probabilité de transition pour les autres états et la fonction de distribution des probabilités des observations.

- **Apprentissage :** Crée un HMM qui modélise une séquence des observations. Si l'apprentissage ne parvient pas à lancer une exception, il serait peut-être parce que nous avons essayé d'estimer une distribution gaussienne avec très peu d'observations (au moins  $n + 1$  observations sont nécessaires pour estimer les paramètres d'une distribution gaussienne multivariée de dimension  $n$ ). Cela signifie que le HMM est composé de trop nombreux états (ou que nous avons utilisé une séquence des observations courte).

- **Sauvegarder HMM** : écrit un HMM vers un fichier (.hmm) (figure 4.28).

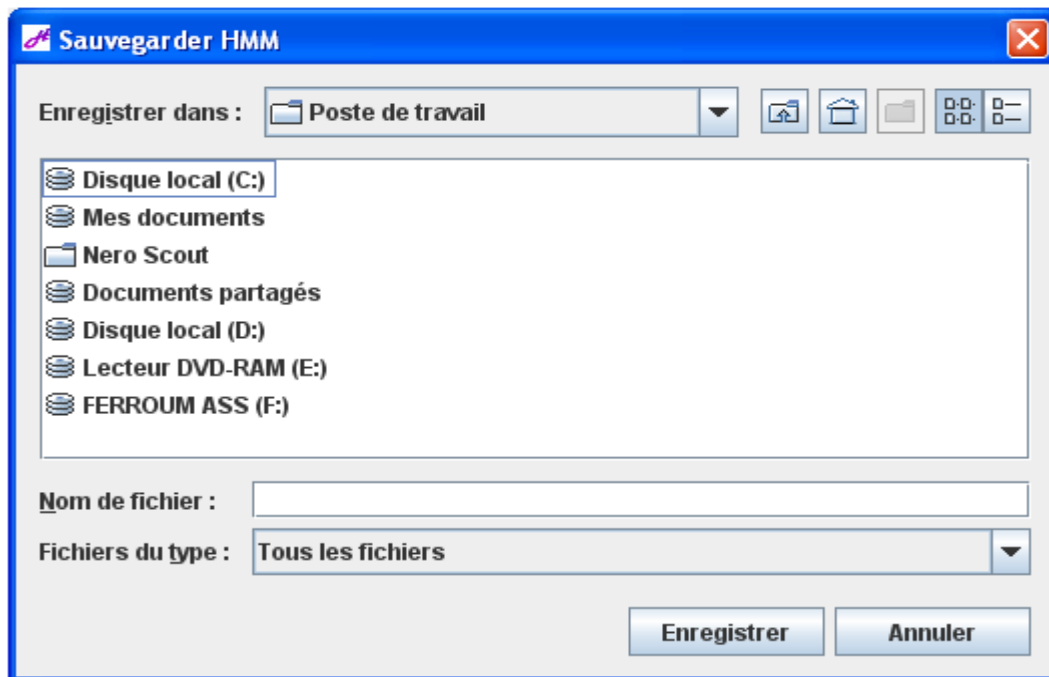


Figure 4.28- La fenêtre Sauvegarder HMM

- ❖ **Séquences-observations** : possède les actions : Ouvrir séquences, Générer séquences, Probabilité de séquence, Sauvegarder séquences (figure 4.29).

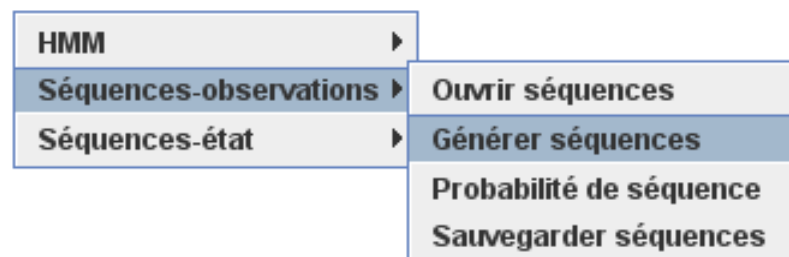


Figure 4.29- Le sous menu Séquences-observations

- **Ouvrir séquences** : Lit un ensemble de séquences d'observations à partir d'un fichier (.seq) (figure 4.31). Lors de la lecture :

- Si les observations sont des entiers, il est demandé de fournir une limite supérieure des observations à lire. Cette limite supérieure doit être strictement supérieure à toutes les observations. La mémoire utilisée augmente avec cette valeur (figure 4.30).
- Si les observations sont des vecteurs des réels, il sera demandé de donner la dimension du vecteur (figure 4.30).

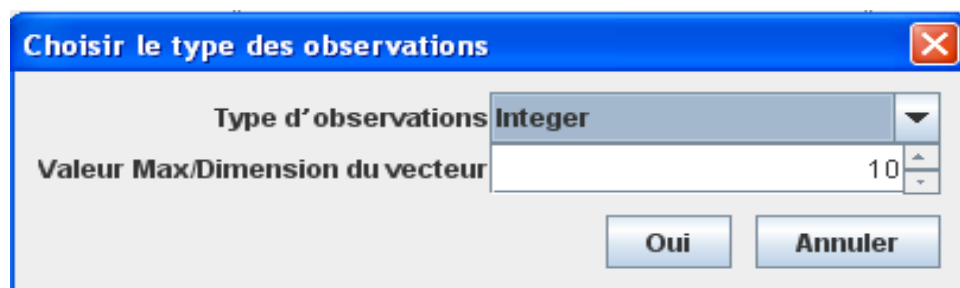


Figure 4.30- La boîte de dialogue pour choisir le type des observations

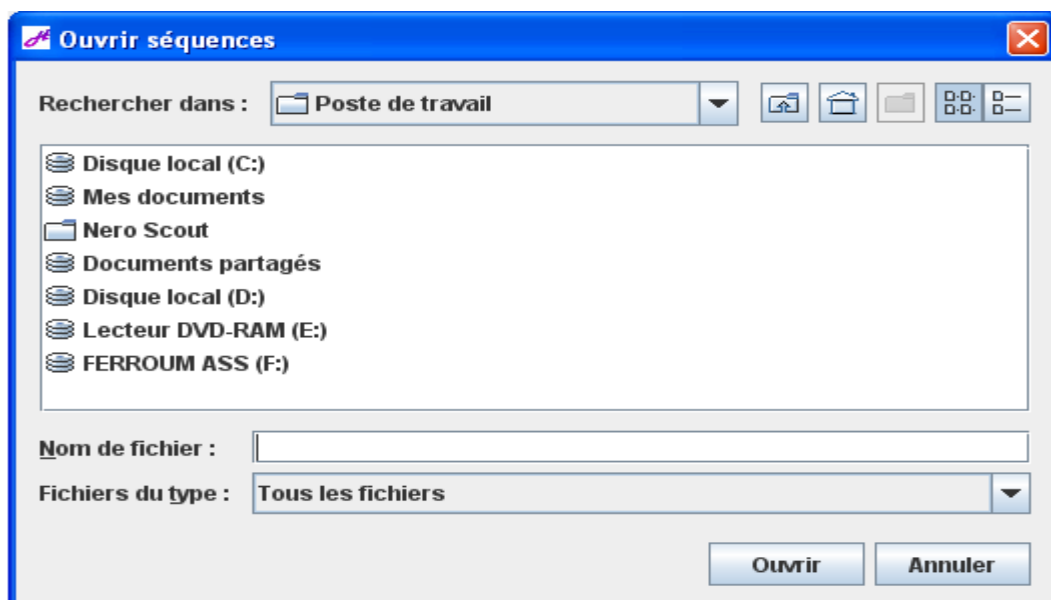


Figure 4.31- La fenêtre Ouvrir Séquences

- **Générer séquences:** génère quelques séquences des observations en utilisant un HMM (figure 4.32).

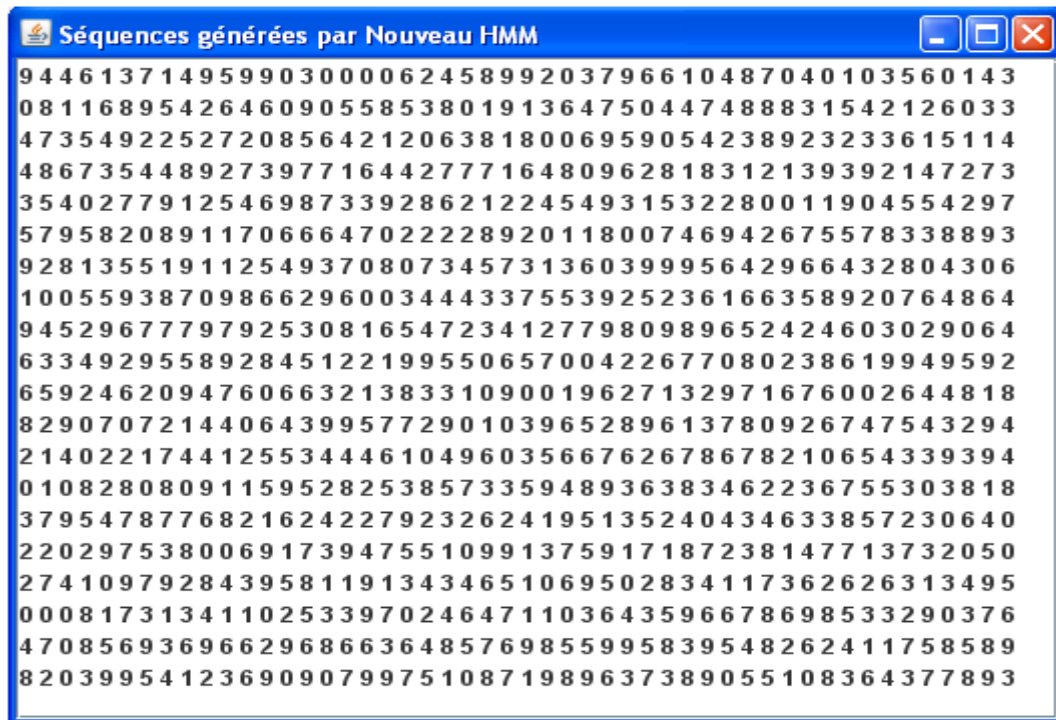


Figure 4.32- La fenêtre Séquences générées par nouveau HMM

- **Probabilité de séquence :** calcule la probabilité d'une séquence d'observations pour un HMM donné (figure 4.33).

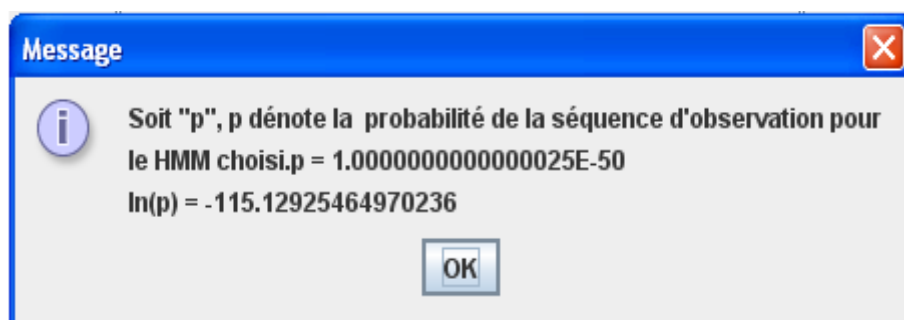


Figure 4.33- Message donnant la probabilité d'une séquence d'observations du nouveau HMM

- **Sauvegarder séquences** : écrit un ensemble de séquences des observations vers un fichier (.seq) (figure 4.34).

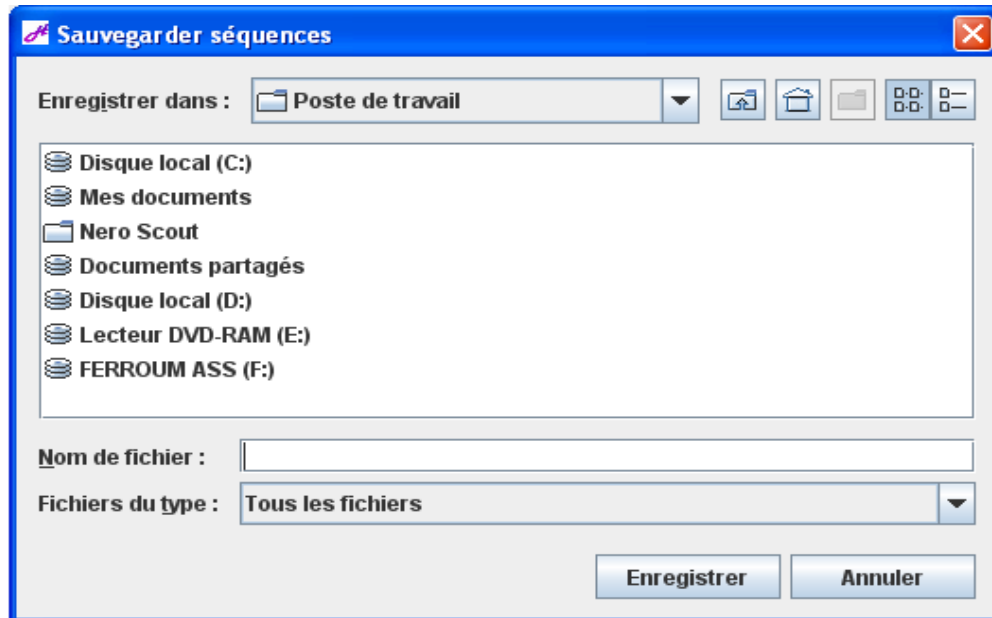


Figure 4.34- La fenêtre Sauvegarder Séquences

- ❖ **Séquences-états** : possède l'action : La plus probable séquence-états (figure 4.35).



Figure 4.35- Le sous menu Séquences-état

- **La plus probable séquence-états**: trouve une suite d'états HMM qui correspondent "mieux" à une des séquences des observations (figure 4.36).

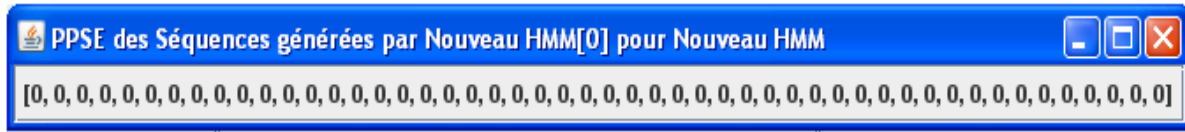


Figure 4.36- La PPSE des séquences générées par nouveau HMM[0]

- **Propriété** : Ce menu possède l'action : Vérifier propriété (figure 4.37).



Figure 4.37- Le menu Propriété

- **Vérifier propriété** : permet de faire apparaître la fenêtre de vérification d'une formule POCTL qui donne à l'utilisateur la possibilité de saisir son propriété spécifiée sous forme d'une formule POCTL et ensuite de la vérifier (figure 4.38).

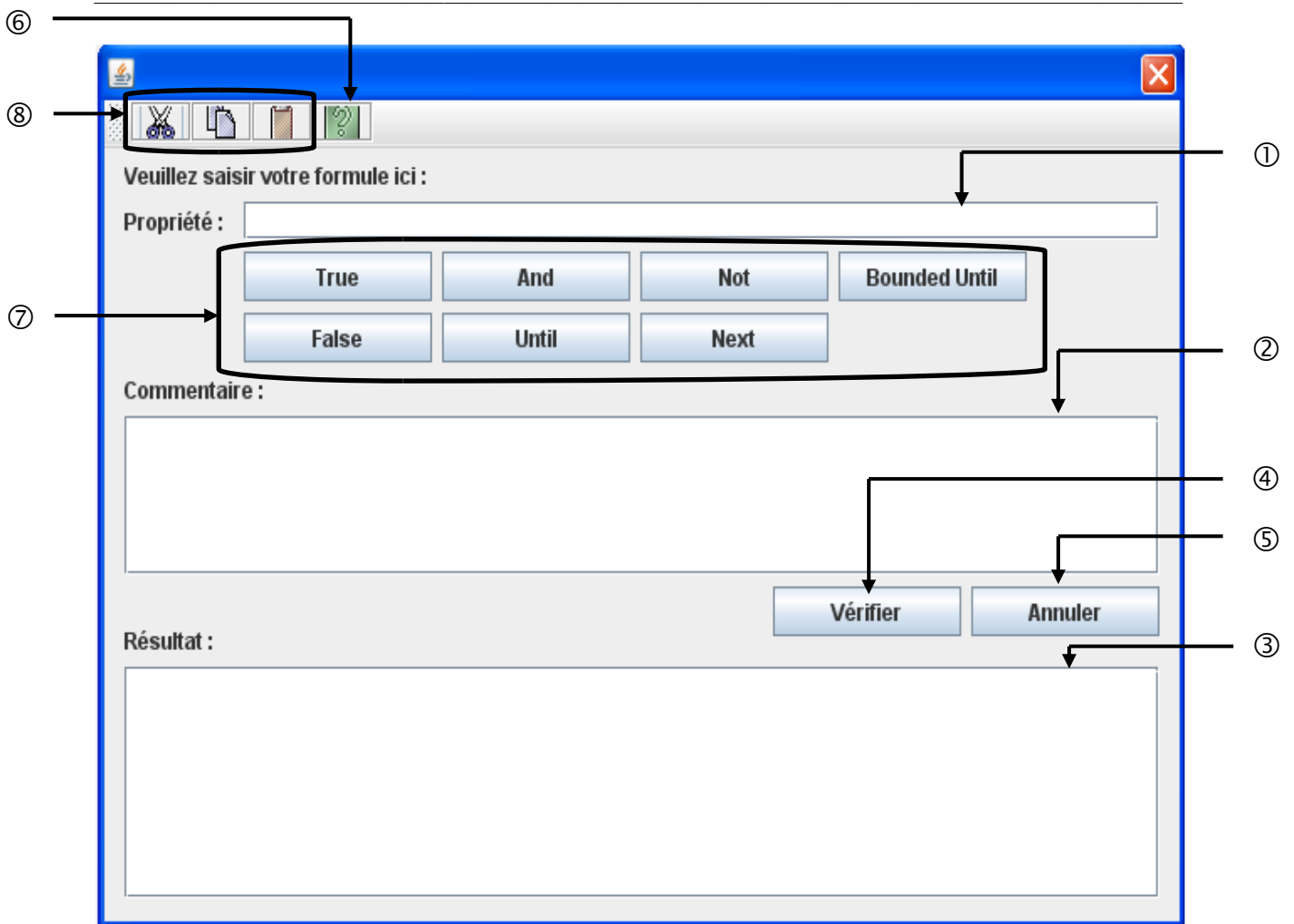


Figure 4.38- La fenêtre de vérification d'une formule POCTL

Voici une description de cette fenêtre :

①: Un espace pour saisir la formule désirée.

②: Un espace pour saisir la signification de la formule.

③ : Un espace pour afficher les résultats de vérification.

④: Le bouton Vérifier qui permet de déclencher la vérification. Une fois les calculs terminés, le programme affiche oui ou non, tout dépendant si le système vérifie la formule ou non, sinon dans le cas où le système ne satisfait pas la formule et il existe des états satisfont la formule, affiche l'ensemble de ces états dans la zone illustrée par ③.

⑤ : Le bouton Annuler qui permet d'annuler la vérification.

⑥: Appuyer sur ce bouton pour faire apparaître la fenêtre « Aide » donnant la syntaxe que doit respecter la formule POCTL saisie par l'utilisateur (figure 4.39).

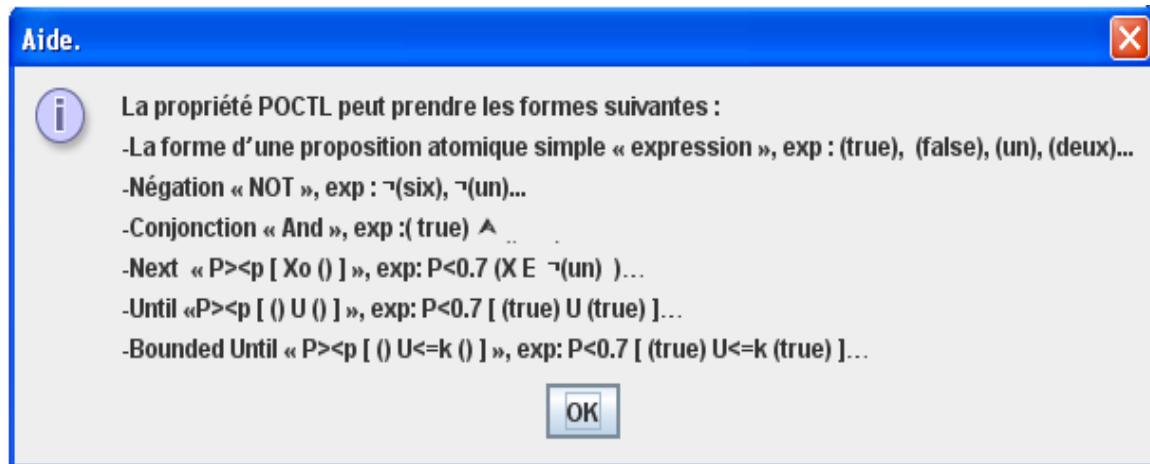


Figure 4.39- La fenêtre « Aide »

⑦ : Boutons aidant l'utilisateur à saisir son formule POCTL correctement et rapidement comme suit :

-  : Afficher au champ de saisie : (true).
-  : Afficher au champ de saisie : (false).
-  : Afficher au champ de saisie : () & ().
-  : Afficher au champ de saisie : !().
-  : Afficher au champ de saisie : P><p [ Xo () ].
-  : Afficher au champ de saisie : P><p [ () U () ].

-  : Afficher au champ de saisie :  $P > < p [ () U \leq k () ]$ .

⑧ : Les boutons images d'édition habituelles pour aider l'utilisateur à saisir son propriété à vérifier dans le champ de saisie illustré par ① et sont respectivement: Couper, Copier, Coller.

## 4.12. Conclusion

Dans ce chapitre, nous avons détaillé la conception de notre model-checker que nous avons appelé HMM\_Model-Checker. Premièrement, nous avons vu le contexte de notre approche qui nous permet de démontrer la nouveauté de notre travail par rapport aux travaux antérieurs vus dans le chapitre précédent puis une vue d'ensemble de notre vérificateur ainsi que son architecture générale. Ensuite, nous avons détaillé l'étape de modélisation qui représente la première étape dans le processus de model-checking, les HMMs sont un formalisme particulièrement intéressant, et très utilisé pour la spécification des systèmes complexes. Enfin, nous avons décrit aussi la deuxième étape de model-checking qui permet la spécification des propriétés ainsi que l'algorithme de model-checking pour POCTL sur les HMMs.

A travers ce chapitre aussi, nous avons présenté un guide utilisateur de notre HMM\_Model-Checker qui est implémenté dans l'environnement Netbeans. A l'aide de l'interface graphique, l'utilisateur peut accéder aux différentes opérations du processus de vérification depuis le chargement de modèle jusqu'à l'affichage du résultat final. Nous avons détaillé l'environnement de développement et le matériel utilisé, les fonctionnalités que nous avons retenues pour l'implémentation qui nous donnent le schéma de développement de notre HMM\_Model-Checker. Nous avons présenté HMM\_Model-Checker et nous avons détaillé son fonctionnement depuis l'interface graphique jusqu'à l'obtention du résultat de vérification.

Notre HMM\_Model-Checker permet comme les autres model-checkers probabilistes de vérifier des formules logiques de type proposition atomique, conjonction de deux propositions atomiques, négation d'une proposition atomique, Next, Until, bounded until, c.à.d. les propriétés du processus stochastique fondamental d'un HMM mais il est le premier et le seul model-checker qui permet de raisonner sur des propriétés de l'autre ensemble de processus stochastique qui produisent les observations.

Notre décision d'implémentation du model-checker HMM\_Model-Checker dans l'environnement Netbeans s'est avérée judicieuse. En effet, il a permis de développer une interface utilisateur facile à utiliser avec le moteur de calculs du model-checker. En plus, les calculs numériques sont améliorés et présentent une performance appréciable.

---

**Chapitre 5**

**Contributions 2 et 3:**

**L'environnement de vérification**

**avec Étude de cas**

---

## 5.1. Introduction

Notre objectif est de concevoir et d'implémenter un environnement de vérification qui comporte plusieurs outils de model-checking probabiliste. De cette façon, l'utilisateur peut faire la recherche avec des paramètres appropriés pour lancer le vérificateur souhaité.

Au cours du développement de ce chapitre, nous allons détailler la conception de notre environnement de vérification, en décrivant l'architecture du système puis en donnant quelques diagrammes UML : cas d'utilisation, diagramme d'activité, diagramme de séquence. Ensuite, nous décrirons son implémentation en donnant les détails concernant les langages et les outils de développement utilisés, puis nous discuterons des fonctionnalités offertes par notre environnement et son schéma de développement pour donner une idée globale sur son implémentation. Enfin, nous présenterons l'environnement, son interface graphique, ainsi que son fonctionnement détaillé. Notons que l'environnement est implémenté en langage Java en utilisant l'environnement Netbeans.

.

## 5.2. Cadre conceptuel du système

L'objectif de notre approche est d'intégrer plusieurs outils dans une plateforme, de cette façon l'utilisateur peut faire la recherche d'un outil avec des paramètres appropriés pour lancer la vérification.

## 5.3. Architecture du système

Notre environnement de vérification, illustré sur la figure 5.1, est un ensemble d'outils algorithmiques qui permettent de vérifier des propriétés décrivant le comportement des systèmes. Le système est composé d'une bibliothèque de 4 outils de vérification des systèmes embarqués. En général, chaque outil possède des modèles et des logiques qui diffèrent des autres outils. A partir de ces caractéristiques (modèles et logiques), on a conçu une interface qui permet à « l'utilisateur » d'obtenir l'outil qui correspond le plus à ses critères de recherche ou bien « l'utilisateur » choisit directement l'outil qui lui est familier, et dans les deux cas, le système affiche l'interface principale de l'outil et ensuite l'utilisateur entre les paramètres de la vérification (modèle et propriété) et la vérification sera lancée. Un autre usager de la

plateforme est « l'administrateur », ce dernier s'occupe de la maintenance des vérificateurs : ajouter ; supprimer ; modifier les outils dans la base de données.

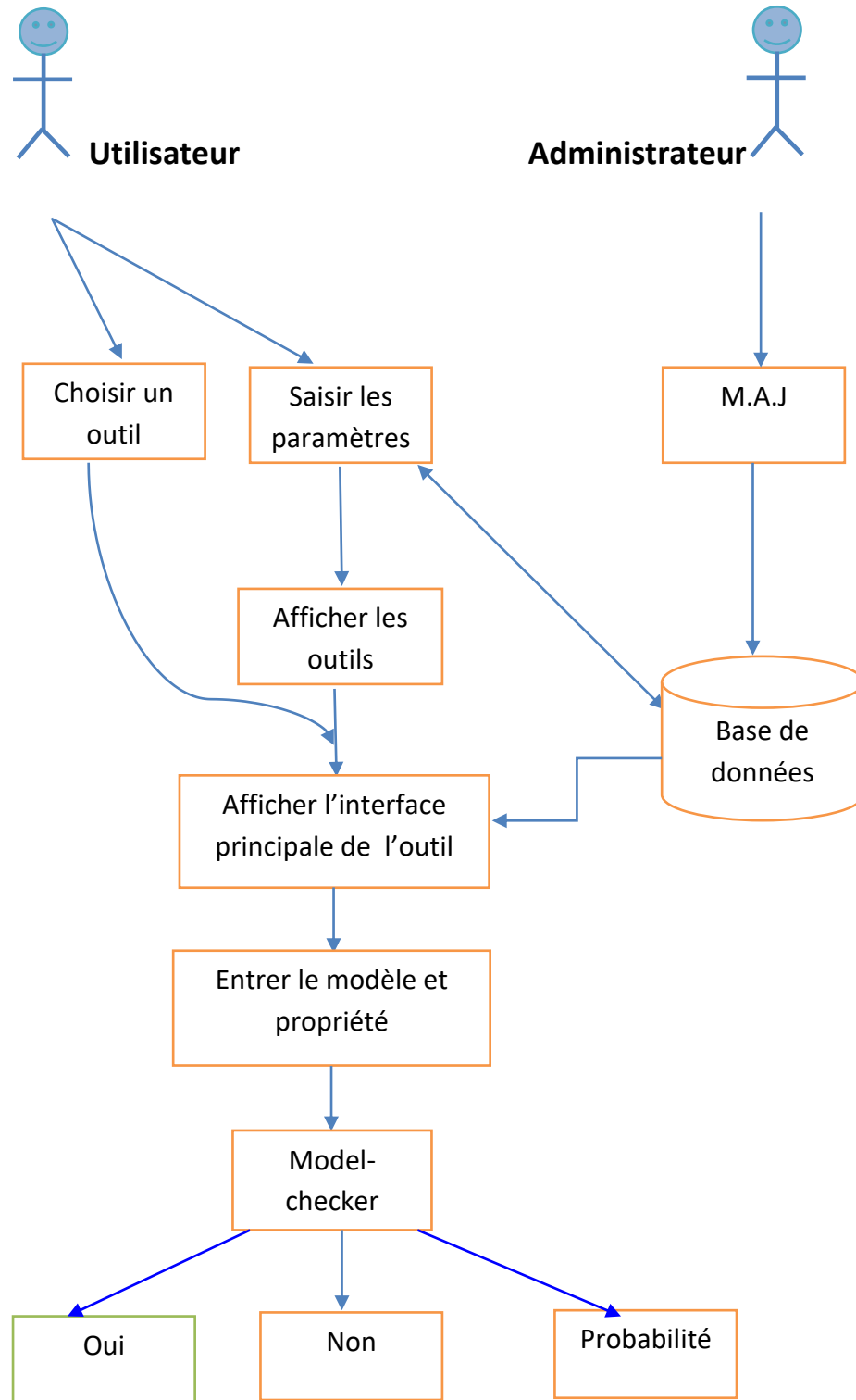


Figure 5.1- Architecture de notre environnement de vérification

## 5.4. Le langage de modélisation UML

En informatique, UML (de l'anglais *Unified Modeling Language*), ou Langage de Modélisation unifié, est un langage de modélisation graphique à base de pictogrammes. Il est utilisé en développement logiciel, et en conception orientée objet. UML est couramment utilisé dans les projets logiciels. Il peut être appliqué à toutes sortes de systèmes. Ainsi, il ne se limite pas au domaine informatique. UML est l'accomplissement de la fusion de précédents langages de modélisation objet : BOOCH, OMT, OOSE [67]. UML 2.3 propose 14 types de diagrammes, nous présentons ci-dessous les plus courants (figure 5.2, figure 5.3, figure 5.4, figure 5.5) :

### 5.4.1. Diagramme des cas d'utilisation

Les diagrammes des cas d'utilisations sont destinés à représenter les besoins des utilisateurs par rapport au système. Il constitue un des diagrammes les plus structurants dans l'analyse d'un système, il montre les interactions fonctionnelles entre les acteurs et le système étudié [67]. Un cas d'utilisation (*use case*) modélise une interaction entre le système informatique à développer et un utilisateur ou acteur interagissant avec le système. Plus précisément, un cas d'utilisation décrit une séquence d'actions réalisées par le système qui produit un résultat observable pour un acteur [68].

### 5.4.2. Diagramme d'activité

La fiche descriptive d'un cas d'utilisation peut contenir plusieurs scénarios alternatifs et/ou d'exception. Il est alors difficile d'avoir une vision de l'ensemble des actions. Le diagramme d'activité et diagramme de séquence sont des moyens graphiques pour donner cette vision d'ensemble. Le diagramme d'activité représente le déroulement des actions, sans utiliser les objets. En phase d'analyse, il est utilisé pour consolider les spécifications d'un cas d'utilisation [68].

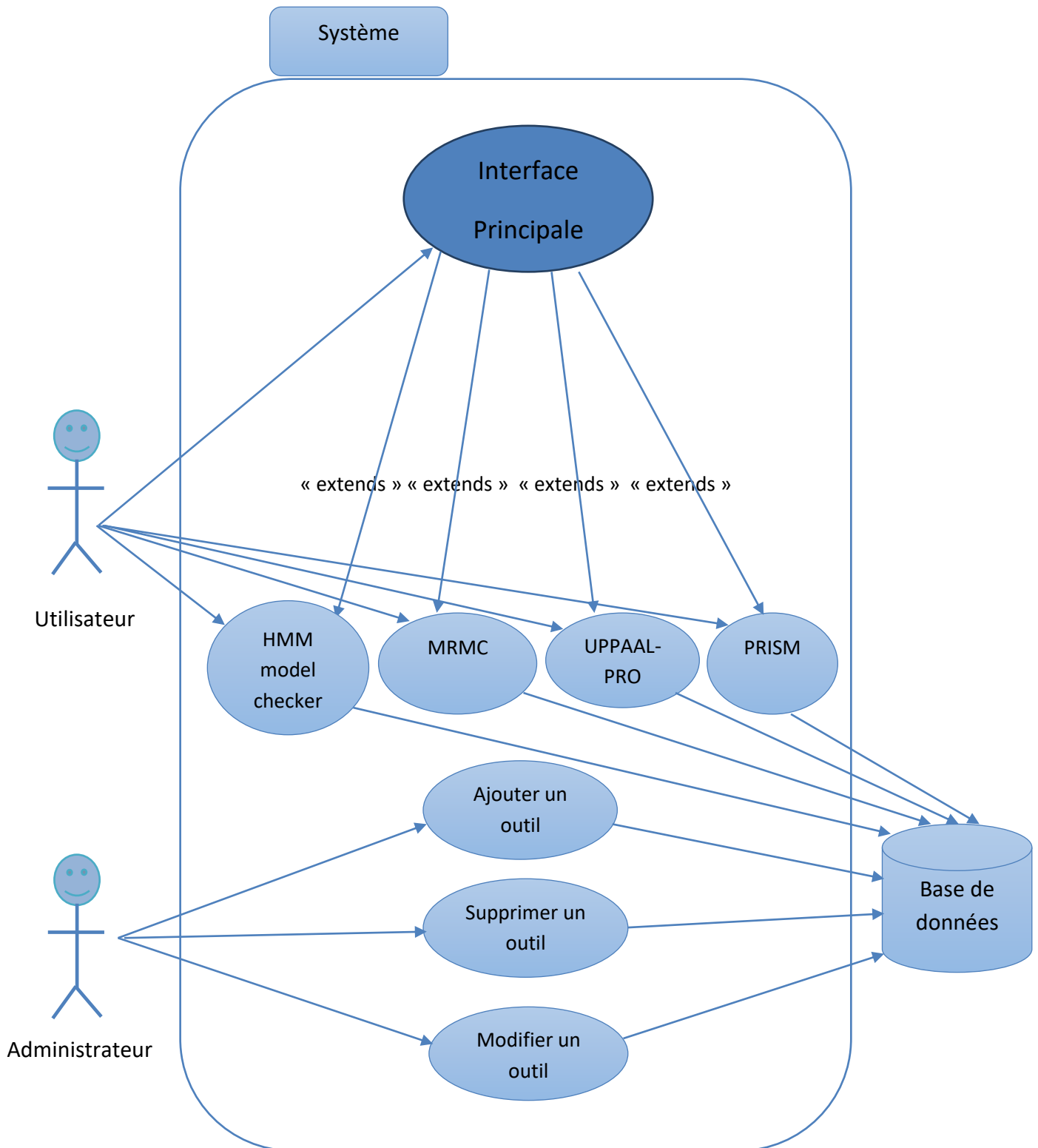


Figure 5.2- Diagramme des cas d'utilisation

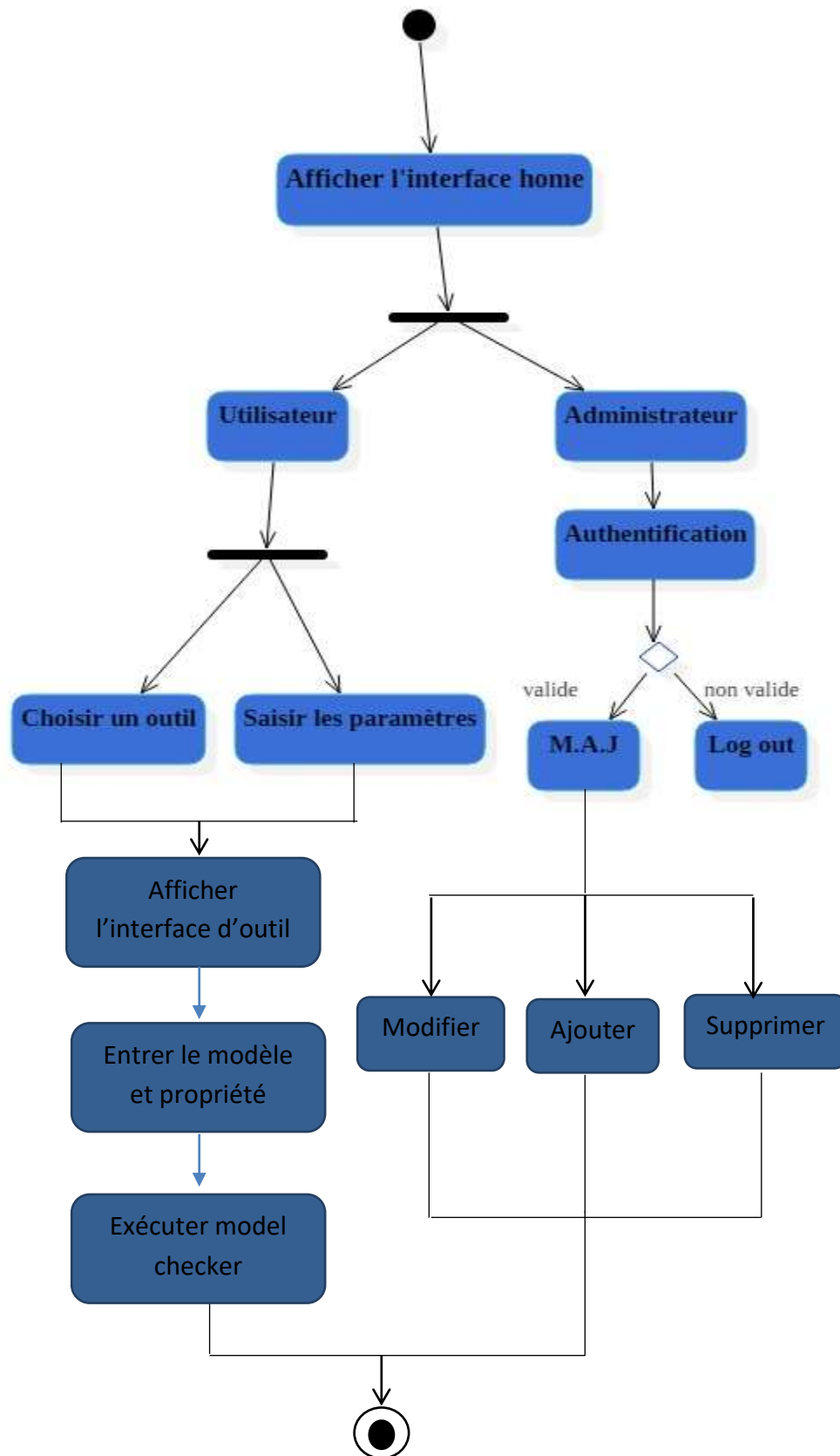


Figure 5.3- Diagramme d'activité.

### 5.4.3. Diagramme de séquence

Le diagramme de séquence est la représentation séquentielle du déroulement des traitements et des interactions entre les éléments du système et/ou de ses acteurs. Un diagramme de séquence représente en général un seul 'scénario' de cas d'utilisation ou flux d'évènements [68].

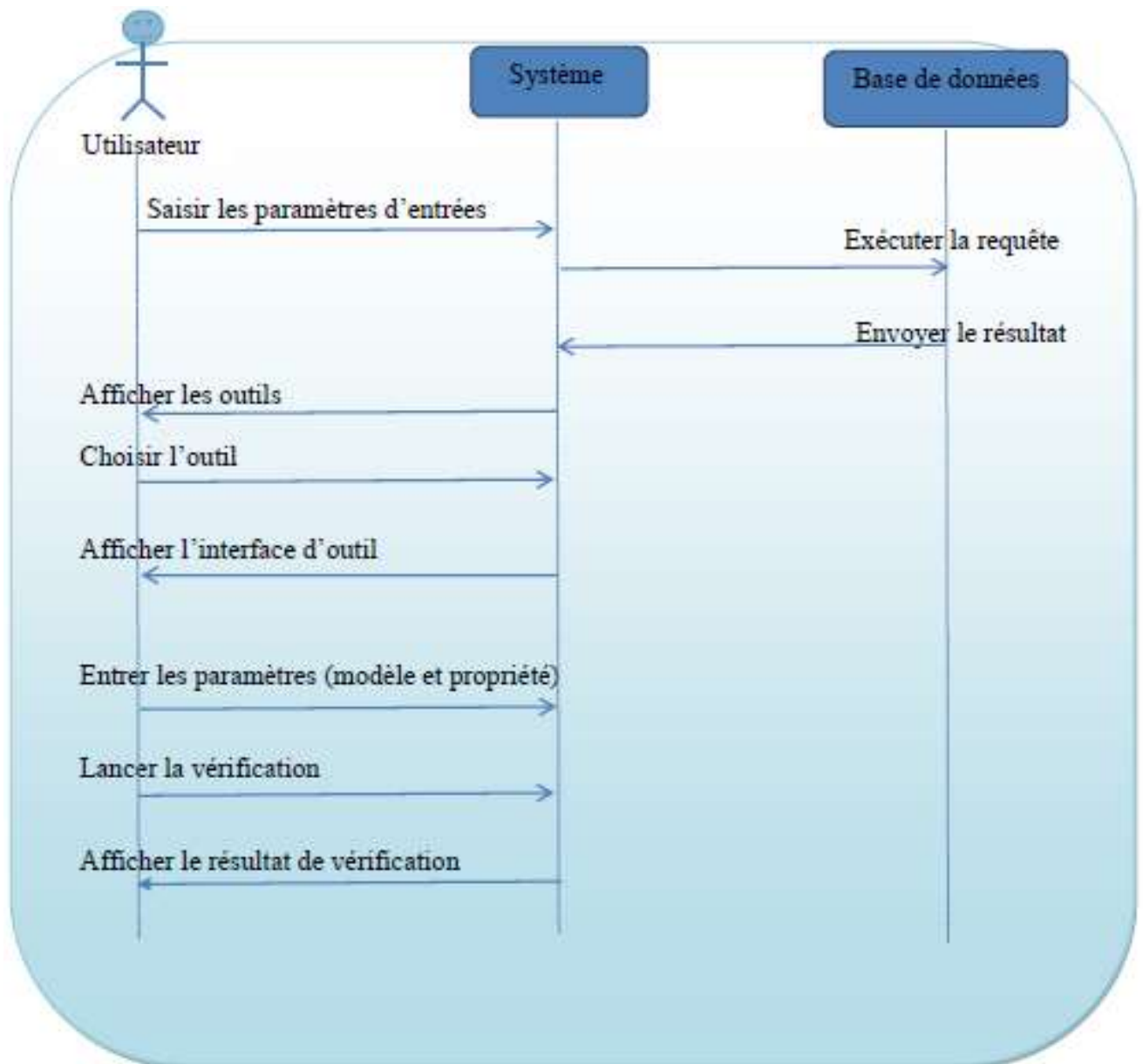


Figure 5.4- Diagramme de séquence (utilisateur)

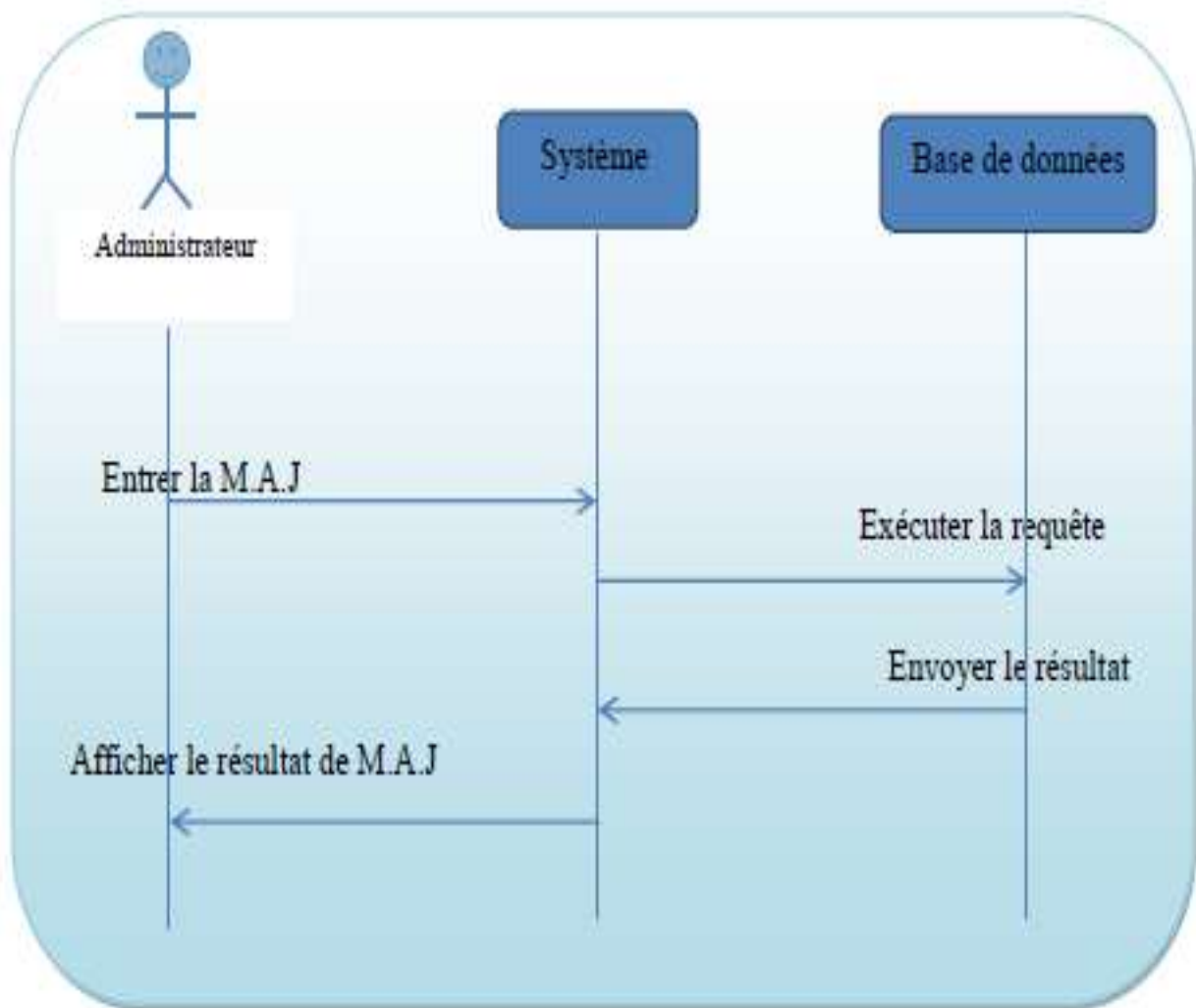


Figure 5.5- Diagramme de séquence (Administrateur)

## 5.5. Environnement du développement

Pour réaliser notre système, nous avons recouru à des langages et des outils de développement :

- **Les langages de développement**

Les langages utilisés pour élaborer cette plateforme sont Java [69] et MySQL [70].

- **Les outils de développement**

De même, il a été fait appel à des outils comme : *JDK SE 8* [69], *Netbeans* [71] et *EasyPHP* [72].

## 5.6. Description générale de l'environnement

Notre environnement de vérification est un programme complexe pouvant rassembler plusieurs fonctionnalités. Dans cette section, nous allons présenter quelques-unes des plus importantes.

- Aider l'utilisateur à saisir le modèle de son système.
- Possibilité de choisir un outil de model-checking probabiliste qui est familier à l'utilisateur.
- Possibilité de choisir et lancer le meilleur outil, s'il y a plusieurs outils utilisant le même modèle et la même logique.
- Possibilité d'ajouter, supprimer ou bien modifier un outil.
- Aider l'utilisateur à saisir le modèle du système, la propriété qui va être vérifiée et ensuite lancer la vérification.

## 5.7. Présentation de l'environnement

Notre environnement de vérification est un ensemble de techniques algorithmiques qui permettent de vérifier des propriétés décrivant le comportement des systèmes, nous avons intégré plusieurs outils probabilistes les plus courants qui sont : PRISM, MRMC, UPPAAL-PRO, ainsi que notre propre outil HMM\_Model-Checker.

Le lancement de notre environnement permet de donner à l'utilisateur la possibilité de choisir un modèle dans une liste, et lui associer automatiquement la logique correspondante, ensuite l'utilisateur peut cliquer sur le bouton *recherche* pour afficher la liste des outils adéquats.

Notre environnement de vérification permet aussi de déterminer le meilleur outil à être utilisé selon des paramètres calculés parmi les outils affichés.

## 5.8. Les interfaces graphiques

En exécutant l'application, l'interface illustrée dans la figure 5.6 apparaît à l'écran. C'est une interface Home, qui permet de choisir le type d'utilisateur.



Figure 5.6- L'interface Home

Si le choix est l'utilisateur, l'interface illustrée dans la figure 5.7 s'affiche à l'écran



Figure 5.7- L'interface de recherche d'un vérificateur

Cette interface permet à l'utilisateur de faire la recherche d'un outil avec des paramètres appropriés (modèle, logique) pour lancer la vérification. Par exemple, si on sélectionne le modèle DTMC dans la case *Modèle* et après on clique sur le bouton *OK* le système affiche la logique correspondante dans la case *Logique*. Comme illustré en figure 5.8.

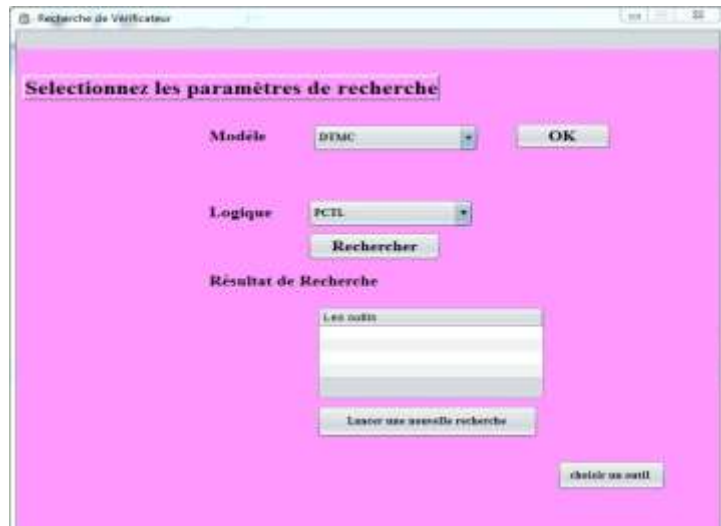


Figure 5.8- L'interface de recherche d'un vérificateur (afficher modèle et logique)

Si on clique sur le bouton Recherche, le système affiche les outils correspondants. Comme illustré en figure 5.9 et après on lance un outil pour la vérification.



Figure 5.9- L'interface de recherche d'un vérificateur (résultat de recherche)

Si le souhait est de choisir un outil directement, on clique sur le bouton « *choisir un outil* » l'interface illustrée dans la figure 5.10 apparaît à l'écran.



Figure 5.10- L'interface des outils

Il y a un autre type d'acteur qui est l'administrateur. Ce dernier s'occupe de la maintenance des outils (Ajouter, Supprimer...) comme montré en figure 5.11.

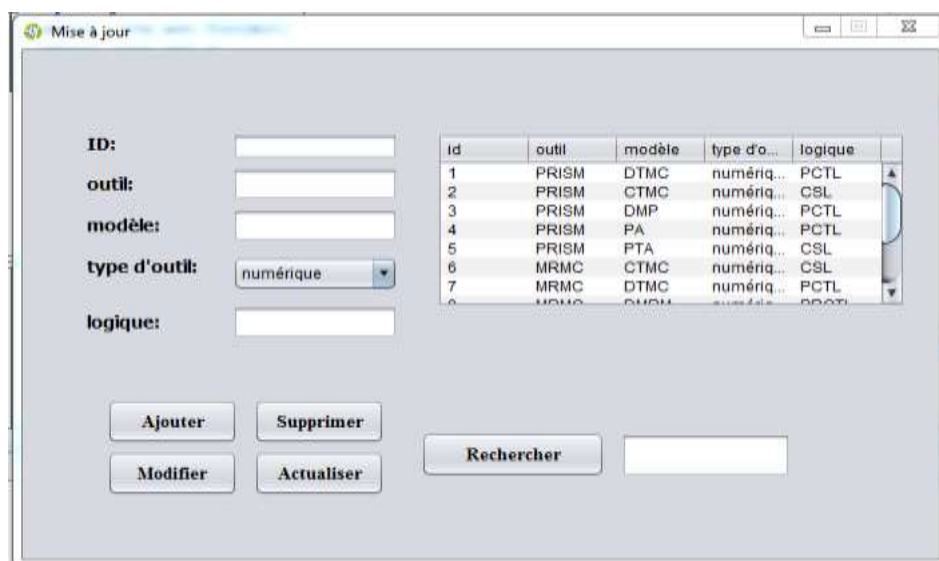


Figure 5.11- L'interface de M.A.J

## 5.9. Étude de cas

Comme l'objectif final est de vérifier de vrais systèmes, tout ce travail de recherche doit être validé. Depuis quelques années, de nombreux outils ont vu le jour et ont permis de réaliser de nombreuses études de cas.

Lors d'une étude de cas, la partie « modélisation » est difficile : il n'y a pas vraiment de règles générales, mais elle se fait au cas par cas. Un cahier des charges décrivant le système réel étant fourni, la première étape consiste à réaliser une première modélisation assez grossière du système : cela sert à se rendre compte quelle classe de modèles paraît bien adaptée à la représentation du système et cela permet aussi éventuellement de choisir déjà un outil qui permettra d'implémenter la modélisation. La deuxième phase consiste à faire une modélisation complète du système, en l'adaptant éventuellement aux choix qui viennent d'être faits.

Dans cette section, nous présentons une étude de cas [73] d'un système embarqué réel intéressant qui est le télescope spatial Hubble [74] (en anglais, *Hubble Space Telescope* ou HST). Cet exemple [53] (une adaptation de [75]) modélise le comportement d'échec du télescope spatial *Hubble*. Nous commençons par une présentation du télescope HST, puis nous décrivons le système réel, constitué de composants qui peuvent probablement échouer. Une fois que le système est compris, nous faisons la modélisation de son comportement d'échec, de sorte que nous puissions prévoir son comportement et poser la question comme : « Quelle est la probabilité que la prochaine observation est une observation d'un échec de gyroscope ? ».

### 5.9.1. Le télescope spatial Hubble

Le télescope spatial *Hubble* (En anglais, *Hubble Space Telescope* ou *HST*) est un télescope en orbite à environ 600 kilomètres d'altitude, il effectue un tour complet de la Terre toutes les 100 minutes. Il est nommé en l'honneur de l'astronome Edwin Hubble. Son lancement, effectué le 24 avril 1990 par une navette spatiale, est le fruit d'un long travail de recherche de la NASA et de l'ESA.

### - Description technique

Le télescope Hubble pèse environ 11 tonnes, fait 13,2 mètres de long, a un diamètre maximum de 4,2 mètres et a coûté 2 milliards de dollars US. C'est un télescope réflecteur à deux miroirs ; le miroir primaire a un diamètre d'environ 2,4 mètres. Il est couplé à divers spectromètres et trois caméras : la première, champ étroit pour les objets faiblement lumineux, la seconde, à large champ pour les images planétaires et la troisième pour l'infrarouge. Le télescope a été lancé le 24 avril 1990 par la mission STS-31 de la navette spatiale Discovery. Ce lancement avait déjà été retardé en 1986 à cause de la catastrophe de la navette spatiale Challenger en janvier de la même année.

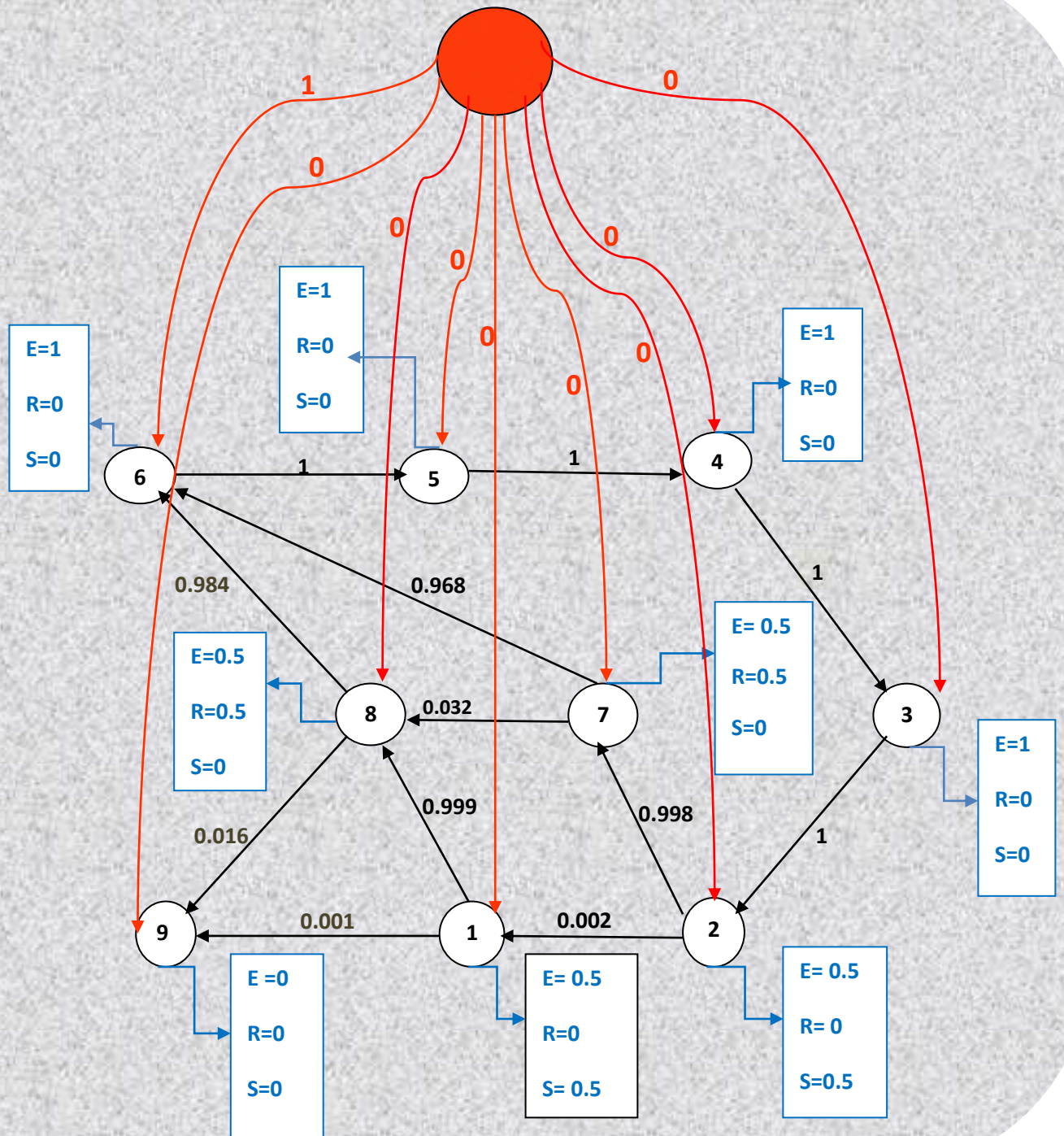
### - Missions d'entretien

Le télescope a été visité quatre fois par des astronautes opérant à partir de navettes spatiales afin de corriger les défauts de fonctionnement et d'installer de nouveaux équipements. De plus, en raison du freinage atmosphérique, le télescope perd lentement de l'altitude (et gagne de la vitesse). Nous profitons donc de chacune de ces visites d'entretien pour replacer le télescope à une orbite plus haute à l'aide de la navette.

#### 5.9.2. Le Système réel

Le système a une unité de direction avec six gyroscopes, qui sont utilisés pour viser le télescope. La redondance est une question importante à la conception des systèmes embarqués tels que les télescopes spatiaux, puisqu'il est évident que l'exécution des réparations ne soit pas insignifiante. C'est pourquoi le télescope est conçu de telle manière qu'il reste en fonction avec la pleine exactitude quand seulement trois de ses six gyroscopes sont opérationnels. Au dessous de trois gyroscopes opérationnels, le télescope se transforme en mode *sommeil*, signifiant que les réparations sont nécessaires. Si aucun des gyroscopes n'est opérationnel, le télescope se brise. Aucune mission de réparation ne sera entreprise, tant qu'il y a plus de deux gyroscopes opérationnels [53].

## 5.9.3 Modélisation avec le modèle HMM

Figure 5.12- Le modèle HMM du télescope spatial *Hubble* (HST)

A partir du DTMC du HST présenté dans [53], nous avons construit son modèle HMM illustré en figure 5.12. Le modèle a un total de neuf états.

- L'ensemble de tous les états est :  $S = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ .
- Les propositions atomiques qui sont « true » dans chaque état :

Les états 1 à 6 : les propositions atomiques égalent le nombre de gyroscopes opérationnels.

- $S_1$  : un
- $S_2$  : deux
- $S_3$  : trois
- $S_4$  : quatre
- $S_5$  : cinq
- $S_6$  : six.
- Les états 7, 8, et 9 sont marqués *sommeil2*, *sommeil1* et *accident*.
  - $S_7$  : *sommeil2* signifie que le télescope est en mode *sommeil* de niveau 2.
  - $S_8$  : *sommeil1* signifie que le télescope est en mode *sommeil* de niveau 1.
  - $S_9$  : *accident* signifie que le télescope est en *accident* (destruction).
- $L$ : est une fonction associant à chaque état  $s$ ,  $L(s)$  l'ensemble des propositions atomiques vérifiées par  $s$  :
 
$$L(S_1) = \{\text{un}\}, L(S_2) = \{\text{deux}\}, L(S_3) = \{\text{trois}\}, L(S_4) = \{\text{quatre}\}, L(S_5) = \{\text{cinq}\}, L(S_6) = \{\text{six}\}, L(S_7) = \{\text{sommeil2}\}, L(S_8) = \{\text{sommeil1}\}, L(S_9) = \{\text{accident}\}.$$

Elle est représentée par la matrice  $L$  définie comme suite (figure 5.13) :

$$\mathbf{L} = \begin{matrix} & \begin{matrix} \text{un} & \text{deux} & \text{trois} & \text{quatre} & \text{cinq} & \text{six} & \text{sommel2} & \text{sommel1} & \text{accident} \end{matrix} \\ \begin{matrix} S_1 \\ S_2 \\ S_3 \\ S_4 \\ S_5 \\ S_6 \\ S_7 \\ S_8 \\ S_9 \end{matrix} & \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{matrix}$$
Figure 5.13- La matrice L du HMM du télescope *HST*

- L'ensemble des observations  $\theta = \{E, R, S\}$ .

Les observations sont :

E (Echec) : représente une observation d'un échec d'un gyroscope.

R (Réparation) : représente une observation d'une mission de réparation entreprise avec succès.

S (Sommeil) : représente une observation qui signifie que le télescope va entrer dans le mode de *sommeil* indiquant que les réparations sont nécessaires.

- La matrice P de probabilités de transition entre les états est comme suite (figure 5.14) :

|                          | <b>S<sub>1</sub></b> | <b>S<sub>2</sub></b> | <b>S<sub>3</sub></b> | <b>S<sub>4</sub></b> | <b>S<sub>5</sub></b> | <b>S<sub>6</sub></b> | <b>S<sub>7</sub></b> | <b>S<sub>8</sub></b> | <b>S<sub>9</sub></b> |
|--------------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|
| <b>S<sub>1</sub></b>     | 0                    | 0                    | 0                    | 0                    | 0                    | 0                    | 0                    | 0.999                | 0.001                |
| <b>S<sub>2</sub></b>     | 0.002                | 0                    | 0                    | 0                    | 0                    | 0                    | 0.998                | 0                    | 0                    |
| <b>S<sub>3</sub></b>     | 0                    | 1                    | 0                    | 0                    | 0                    | 0                    | 0                    | 0                    | 0                    |
| <b>P = S<sub>4</sub></b> | 0                    | 0                    | 1                    | 0                    | 0                    | 0                    | 0                    | 0                    | 0                    |
| <b>S<sub>5</sub></b>     | 0                    | 0                    | 0                    | 1                    | 0                    | 0                    | 0                    | 0                    | 0                    |
| <b>S<sub>6</sub></b>     | 0                    | 0                    | 0                    | 0                    | 1                    | 0                    | 0                    | 0                    | 0                    |
| <b>S<sub>7</sub></b>     | 0                    | 0                    | 0                    | 0                    | 0                    | 0.968                | 0                    | 0.032                | 0                    |
| <b>S<sub>8</sub></b>     | 0                    | 0                    | 0                    | 0                    | 0                    | 0.984                | 0                    | 0                    | 0.016                |
| <b>S<sub>9</sub></b>     | 0                    | 0                    | 0                    | 0                    | 0                    | 0                    | 0                    | 0                    | 0                    |

Figure 5.14- La matrice P du HMM du *HST*

La matrice  $\mu$  de probabilités des observations est définie comme suite (figure 5.15) :

$$\mu = \begin{pmatrix} 0.5 & 0 & 0.5 \\ 0.5 & 0 & 0.5 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0.5 & 0.5 & 0 \end{pmatrix}$$

Figure 5.15- La matrice  $\mu$  du HMM correspondante

- $\pi$  est le vecteur de distribution des probabilités de transitions initiales. L'état 6 est le seul état initial où tous les gyroscopes sont opérationnels ; c'est également l'état du télescope après une mission réussie de réparation. Le vecteur  $\pi$  s'écrit (figure 5.16) :

$$\pi = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Figure 5.16- Le vecteur  $\pi$  du HMM du télescope *HST*

#### 5.9.4 Choix des propriétés à vérifier

Parmi les propriétés vérifiées par notre outil, nous avons choisi de présenter les trois propriétés suivantes :

- **La propriété 1 :**

« La probabilité que la prochaine observation est une observation d'un échec de gyroscope et le chemin suivant satisfait que le nombre des gyroscopes opérationnels est différent de six, est inférieure ou égale à 1. ». C'est une propriété d'*atteignabilité*.

- **La propriété 2 :**

« La probabilité que la prochaine observation est une observation d'un échec de gyroscope et le chemin suivant satisfait que le télescope se brisera est supérieure ou égale à 0 ». C'est une propriété de *sûreté*.

- **La propriété 3 :**

« La probabilité que la prochaine observation est une observation d'un échec de gyroscope et le chemin suivant satisfait que le télescope se brisera est inférieure ou égale à 0.008. c.à.d. la probabilité que le télescope se brisera est inférieure ou égale à 0.008 ». C'est une propriété de *sûreté*.

- **Spécification des propriétés choisies en POCTL**

Nous avons spécifié chaque propriété en suivant la syntaxe de logique POCTL comme suit :

- **La propriété 1 :**  $P_{\leq 1} [X_E \neg(\text{six})]$
- **La propriété 2 :**  $P_{>=0} [X_E (\text{accident})]$
- **La propriété 3 :**  $P_{\leq 0.008} [X_E (\text{accident})]$

### 5.9.5. Vérification des propriétés

Nous vérifions la satisfiabilité ou non de ces trois propriétés spécifiées en POCTL dans la section précédente, sur le modèle HMM du *HST* - vu dans la sous-section 5.9.3 - manuellement et automatiquement, puis une comparaison entre les deux sera faite pour valider le fonctionnement de notre HMM\_Model-Checker.

## - Application manuelle de l'algorithme

### 1. $P_{\leq 1} [X_E \neg(\text{six})]$

Nous savons d'après ce que nous avons vu dans le chapitre 4 que :

$$p_s(X_\Omega \phi) = \mu_s(\Omega) \cdot \sum_{s' \in \text{sat}(\psi)} p(s, s').$$

Donc, nous allons calculer la satisfaction de cette formule dans chaque état, si le résultat de vérification nous donne l'ensemble total des états du modèle, alors nous allons conclure que le modèle a vérifié la formule.

- La vérification de la formule  $P_{\leq 1} [X_E \neg(\text{six})]$  dans l'état  $s_1$ :

$$P_{s_1} [X_E \neg(\text{six})] = \mu_{s_1}(E) \cdot \sum_{s' \in \text{sat}(\neg \text{six})} p(s_1, s').$$

$$= \mu_{s_1}(E) (p(s_1, s_8) + p(s_1, s_9))$$

$$= 0.5 (0.999 + 0.001)$$

$$= \mathbf{0.5}$$

- Nous remarquons que  $\mathbf{0.5} \leq 1$  donc  $s_1 \models [X_E \neg(\text{six})]$  (1)

- La vérification de la formule  $P_{\leq 1} [X_E \neg(\text{six})]$  dans l'état  $s_2$ :

$$P_{s_2} [X_E \neg(\text{six})] = \mu_{s_2}(E) \cdot \sum_{s' \in \text{sat}(\neg \text{six})} p(s_2, s').$$

$$= \mu_{s_2}(E) (p(s_2, s_1) + p(s_2, s_7))$$

$$= 0.5 (0.002 + 0.998)$$

$$= \mathbf{0.5}$$

- Nous remarquons que  $\mathbf{0.5} \leq 1$  donc  $s_2 \models [X_E \neg(\text{six})]$  (2)

- La vérification de la formule  $P_{\leq 1} [X_E !(six)]$  dans l'état  $s_3$  :

$$\begin{aligned}
 Ps_3 [X_E !(six)] &= \mu_{s_3}(E) \cdot \sum_{s' \in \text{sat}(\neg six)} p(s_3, s'). \\
 &= \mu_{s_3}(E) (p(s_3, s_2)) \\
 &= 0 \text{ (1)} \\
 &= \mathbf{0}
 \end{aligned}$$

- Nous remarquons que  $\mathbf{0} \leq 1$  donc  $s_3 \models [X_E !(six)]$  (3)

- La vérification de la formule  $P_{\leq 1} [X_E !(six)]$  dans l'état  $s_4$  :

$$\begin{aligned}
 Ps_4 [X_E !(six)] &= \mu_{s_4}(E) \cdot \sum_{s' \in \text{sat}(\neg six)} p(s_4, s'). \\
 &= \mu_{s_4}(E) (p(s_4, s_3)) \\
 &= 1 \text{ (1)} \\
 &= \mathbf{1}
 \end{aligned}$$

- Nous remarquons que  $\mathbf{1} \leq 1$  donc  $s_4 \models [X_E !(six)]$  (4)

- La vérification de la formule  $P_{\leq 1} [X_E !(six)]$  dans l'état  $s_5$  :

$$\begin{aligned}
 Ps_5 [X_E !(six)] &= \mu_{s_5}(E) \cdot \sum_{s' \in \text{sat}(\neg six)} p(s_5, s'). \\
 &= \mu_{s_5}(E) \\
 &= 1 \text{ (1)} \\
 &= \mathbf{1}
 \end{aligned}$$

- Nous remarquons que  $\mathbf{1} \leq 1$  donc  $s_5 \models [X_E !(six)]$  (5)

- La vérification de la formule  $P_{\leq 1} [X_E !(six)]$  dans l'état  $s_6$  :

$$Ps_6 [X_E !(six)] = \mu_{s_6}(E) \cdot \sum_{s' \in \text{sat}(\neg six)} p(s_6, s').$$

$$= \mu_{s_6}(E) (p(s_6, s_5))$$

$$= 1 (1)$$

$$= \mathbf{1}$$

- Nous remarquons que  $\mathbf{1} \leq 1$  donc  $s_6 \models [X_E !(six)]$  (6)

- La vérification de la formule  $P_{\leq 1} [X_E !(six)]$  dans l'état  $s_7$  :

$$Ps_7 [X_E !(six)] = \mu_{s_7}(E) \cdot \sum_{s' \in \text{sat}(\neg six)} p(s_7, s').$$

$$= \mu_{s_7}(E) (p(s_7, s_8))$$

$$= 0.5 (0.032)$$

$$= \mathbf{0.016}$$

- Nous remarquons que  $\mathbf{0.016} \leq 1$  donc  $s_7 \models [X_E !(six)]$  (7)

- La vérification de la formule  $P_{\leq 1} [X_E !(six)]$  dans l'état  $s_8$  :

$$Ps_8 [X_E !(six)] = \mu_{s_8}(E) \cdot \sum_{s' \in \text{sat}(\neg six)} p(s_8, s').$$

$$= \mu_{s_8}(E) (p(s_8, s_9))$$

$$= 0.5 (0.016)$$

$$= \mathbf{0.008}$$

- Nous remarquons que  $\mathbf{0.008} \leq 1$  donc  $s_8 \models [X_E !(six)]$  (8)

- La vérification de la formule  $P_{\leq 1} [X_E !(six)]$  dans l'état  $s_9$  :

$$\begin{aligned} P_{s_9} [X_E !(six)] &= \mu_{s_9}(E) \cdot \sum_{s' \in sat(\neg six)} p(s_9, s') \\ &= \mu_{s_9}(E) \\ &= \mathbf{0} \end{aligned}$$

- Nous remarquons que  $\mathbf{0} \leq 1$  donc  $s_9 \models [X_E !(six)]$  (9)

De (1), (2), (3), (4), (5), (6), (7), (8), (9) nous concluons que :

$Sat [X_E !(six)] = \{ s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9 \} = S$ , alors la propriété 1 est vérifiée dans tout le modèle.

## 2. $P_{\geq 0} [X_E (\text{accident})]$

$s$  satisfait  $P_{\geq 0} [X_E (\text{accident})]$  ssi  $p_s(X_E \text{ accident}) \geq 0$ .

Nous avons :  $p_s(X_\Omega \phi) = \mu_s(\Omega) \cdot \sum_{s' \in sat(\phi)} p(s, s')$

$$\begin{aligned} p_{s_1} [X_E (\text{accident})] &= \mu_{s_1}(E) \cdot \sum_{s' \in sat(\text{accident})} p(s_1, s'). \text{ tq: } s' = \{s_9\} \\ &= 0.5 (0.001) = \mathbf{0.0005} \geq \mathbf{0} \end{aligned}$$

$$\begin{aligned} p_{s_2} (X_E \text{ accident}) &= \mu_{s_2}(E) \cdot \sum_{s' \in sat(\text{accident})} p(s_2, s'). \text{ tq: } s' = \{s_9\} \\ &= 0.5 (0) \geq \mathbf{0} \end{aligned}$$

$$\begin{aligned} p_{s_3} [X_E (\text{accident})] &= \mu_{s_3}(E) \cdot \sum_{s' \in sat(\text{accident})} p(s_3, s'). \text{ tq: } s' = \{s_9\} \\ &= 1 (0) \geq \mathbf{0} \end{aligned}$$

$$p_{s4}[X_E \text{ (accident)}] = \mu_{s4}(E) \cdot \sum_{s' \in \text{sat}(\text{accident})} p(s4, s') \cdot tq: s' = \{ \} \\ = 1 (0) \geq 0$$

$$p_{s5}[X_E \text{ (accident)}] = \mu_{s5}(E) \cdot \sum_{s' \in \text{sat}(\text{accident})} p(s5, s') \cdot tq: s' = \{ \} \\ = 1 (0) \geq 0$$

$$p_{s6}[X_E \text{ (accident)}] = \mu_{s6}(E) \cdot \sum_{s' \in \text{sat}(\text{accident})} p(s6, s') \cdot tq: s' = \{ \} \\ = 1 (0) \geq 0$$

$$p_{s7}[X_E \text{ (accident)}] = \mu_{s7}(E) \cdot \sum_{s' \in \text{sat}(\text{accident})} p(s7, s') \cdot tq: s' = \{ \} \\ = 0.5 (0) \geq 0$$

$$p_{s8}[X_E \text{ (accident)}] = \mu_{s8}(E) \cdot \sum_{s' \in \text{sat}(\text{accident})} p(s8, s') \cdot tq: s' = \{s9\} \\ = 0.5 (0.016) = \mathbf{0.008} \geq 0$$

$$p_{s9}[X_E \text{ (accident)}] = \mu_{s9}(E) \cdot \sum_{s' \in \text{sat}(\text{accident})} p(s9, s') \cdot tq: s' = \{ \} \\ = 0 (0) \geq 0$$

Nous verrons que la formule  $P_{\geq 0} [X_E \text{ (accident)}]$  est satisfaite dans tous les états du modèle. Alors La probabilité que le télescope se brisera est supérieure ou égale à 0.

### 3. $P_{\leq 0.008} [X_E \text{ (accident)}]$

s satisfait  $P_{\leq 0.008} (X_E \text{ accident})$  ssi  $p_s(X_E \text{ accident}) \leq 0.008$ .

On a:  $p_s(X_{\Omega}\phi) = \mu_s(\Omega) \cdot \sum_{s' \in \text{sat}(\phi)} p(s, s')$ ,  $\phi = \text{accident}$  donc  $s' = \{s9\}$

Dans ce qui suit, chaque ligne correspond à la satisfaction de la propriété au sein d'un état ( $s_i$ ,  $i=1-9$ )

$$\begin{aligned} p_{s1} [X_E \text{ (accident)}] &= \mu_{s1}(E) \cdot \sum_{s' \in \text{sat (accident)}} p(s_1, s') = 0.5 (0.001) \\ &= 0.0005 \leq 0.008 \end{aligned}$$

$$\begin{aligned} p_{s2} [X_E \text{ (accident)}] &= \mu_{s2}(E) \cdot \sum_{s' \in \text{sat (accident)}} p(s_2, s') = 0.5 (0) \\ &= 0 \leq 0.008 \end{aligned}$$

$$\begin{aligned} p_{s3} [X_E \text{ (accident)}] &= \mu_{s3}(E) \cdot \sum_{s' \in \text{sat (accident)}} p(s_3, s') = 1 (0) \\ &= 0 \leq 0.008 \end{aligned}$$

$$\begin{aligned} p_{s4} [X_E \text{ (accident)}] &= \mu_{s4}(E) \cdot \sum_{s' \in \text{sat (accident)}} p(s_4, s') = 1 (0) \\ &= 0 \leq 0.008 \end{aligned}$$

$$\begin{aligned} p_{s5} [X_E \text{ (accident)}] &= \mu_{s5}(E) \cdot \sum_{s' \in \text{sat (accident)}} p(s_5, s') = 1 (0) \\ &= 0 \leq 0.008 \end{aligned}$$

$$\begin{aligned} p_{s6} [X_E \text{ (accident)}] &= \mu_{s6}(E) \cdot \sum_{s' \in \text{sat (accident)}} p(s_6, s') = 1 (0) \\ &= 0 \leq 0.008 \end{aligned}$$

$$\begin{aligned} p_{s7} [X_E \text{ (accident)}] &= \mu_{s7}(E) \cdot \sum_{s' \in \text{sat (accident)}} p(s_7, s') = 0.5 (0) \\ &= 0 \leq 0.008 \end{aligned}$$

$$\begin{aligned} p_{s8} [X_E \text{ (accident)}] &= \mu_{s8}(E) \cdot \sum_{s' \in \text{sat (accident)}} p(s_8, s') = 0.5 (0.016) \\ &= 0.008 \leq 0.008 \end{aligned}$$

$$\begin{aligned} p_{s9} [X_E \text{ (accident)}] &= \mu_{s9}(E) \cdot \sum_{s' \in \text{sat (accident)}} p(s_9, s') = 0 (0) \\ &= 0 \leq 0.008 \end{aligned}$$

Nous venons de voir d'après la trace de l'algorithme que la formule  $P_{\leq 0.008} [X_E(\text{accident})]$  est satisfaite dans tous les états du modèle ( $s_i$ ,  $i=1 \dots 9$ ). Ainsi la propriété (sûreté) est vérifiée pour tout le modèle, et par conséquent la probabilité que le télescope se brisera est inférieure ou égale à 0.008.

D'après les deux propriétés 2 et 3, nous déduisons que la probabilité que le télescope se brisera est dans l'intervalle  $[0, 0.008]$ .

## - Vérification automatique

### 1. $P_{\leq 1} [X_E \neg(\text{six})]$

Veuillez saisir votre formule ici :

Propriété :

True And Not Bounded Until

False Until Next

Commentaire :

La probabilité que la prochaine observation est une observation d'un échec d'un gyroscope et le chemin suivant satisfait que le nombre des gyroscopes opérationnels est différent à six est inférieur ou égal à 1.

Vérifier Annuler

Résultat :

OUI

Figure 5.17- Résultat de la vérification de la propriété 1 en utilisant notre outil

## 2. $P_{\geq 0} [X_E (\text{accident})]$

The screenshot shows a software window with a blue title bar and a standard Windows-style toolbar. The main area is divided into several sections:

- Veillez saisir votre formule ici :** A label followed by a text input field containing the formula  $P_{\geq 0} [X_E (\text{accident})]$ .
- Buttons:** A grid of buttons for logical operations: True, And, Not, Bounded Until, False, Until, and Next.
- Commentaire :** A large text area containing the comment: "La probabilité que le télescope se brisera est supérieure ou égale à 0."
- Buttons:** Two buttons, "Vérifier" and "Annuler", positioned to the right of the comment area.
- Résultat :** A label followed by a text area displaying the result "OUI".

Figure 5.18- Résultat de la vérification de la propriété 2 en utilisant notre outil

### 3. $P_{\leq 0.008} [X_E(\text{accident})]$

The screenshot shows a software window with a blue title bar and a standard toolbar. The main area contains the following elements:

- Veillez saisir votre formule ici :** A label above a text input field.
- Propriété :** A label above the text input field, which contains the formula  $P_{\leq 0.008} [X_E(\text{accident})]$ .
- Buttons:** A grid of buttons including "True", "And", "Not", "Bounded Until", "False", "Until", and "Next".
- Commentaire :** A label above a large text area containing the comment: "La probabilité que le télescope se brisera est inférieure ou égale à 0.008."
- Résultat :** A label above a text area.
- Buttons:** "Vérifier" and "Annuler" buttons.
- Result:** The text "OUI" is displayed in the result area.

Figure 5.19- Résultat de la vérification de la propriété 3 en utilisant notre outil

#### 5.9.6. Résultats et discussions

Nous avons confronté les résultats obtenus par HMM\_Model-Checker à une double comparaison sur le même système HST : la première aux résultats obtenus manuellement et la seconde à ceux obtenus par l'outil pState [76]. Comme il est présenté ci-dessus, les résultats de la vérification manuelle coïncident avec ceux de la vérification automatique, ce qui nous a permis d'affirmer que l'implémentation de notre outil est correcte. Pour nous permettre la comparaison avec les résultats obtenus par l'outil pState présentés dans le travail de Nokovic [77], nous avons considéré la même propriété à vérifier (Propriété 3), les résultats obtenus sont meilleurs que ceux fournis par l'outil pState [77] : Les auteurs de pState, ont déterminé la probabilité pour que le télescope se brisera égale à 0.0122, alors qu'avec notre outil, la

probabilité est inférieure ou égale à 0.008.

## 5.10. Conclusion

Dans ce chapitre, nous avons détaillé la conception et l'implémentation de notre environnement de vérification. Nous avons décrit l'architecture du système puis nous avons donné quelques diagrammes UML : cas d'utilisation, diagramme d'activité et diagramme de séquence. Ensuite, nous avons présenté un guide utilisateur de notre environnement qui est implémenté en langage Java en utilisant l'environnement Netbeans. A l'aide de l'interface graphique, l'utilisateur peut accéder à l'outil de vérification adéquat ou souhaité très facilement.

Dans ce chapitre, nous avons présenté aussi, une étude de cas d'un système embarqué réel important qui est le *HST* et les résultats obtenus sont encourageants. Nous avons construit le modèle HMM du HST en l'adaptant de son modèle DTMC présenté dans [53] : ajout de l'ensemble des observations  $\theta$ , la matrice  $\mu$  de distribution des probabilités d'observations, le vecteur  $\pi$  de distribution des probabilités de transitions initiales. L'étape de modélisation est la partie la plus difficile lors d'une étude de cas, car les classes de modèles implémentées dans l'outil ne permettent pas toujours d'exprimer le système en entier, mais uniquement une simplification de ce système (ce que l'on appelle une *abstraction*). Après avoir implémenté le modèle et la propriété dans l'outil, théoriquement, il ne reste plus qu'à appuyer sur le bouton de l'outil pour obtenir le résultat voulu, à savoir la réponse à la question « est-ce que le modèle vérifie la spécification ? ». Cependant, les limitations dues aux machines utilisées peuvent faire échouer ce calcul (par exemple à cause d'un manque de mémoire). Il faut alors encore simplifier la modélisation (tout en préservant les caractéristiques importantes du système) et recommencer.

Dans l'étape de spécification des propriétés, nous avons choisi de présenter trois propriétés qui nous ont permis de vérifier le bon fonctionnement du système, puis nous avons spécifié ces propriétés en logique POCTL. Toutes les propriétés choisies sont du type Next suivies d'une observation qui est le cas qui démontre notre approche par rapport aux autres model-checkers, car pour un HMM donné, nous sommes souvent intéressés par les propriétés du processus stochastique fondamental que nous pouvons considérer comme un modèle DTMC. En plus, nous nous sommes intéressés également au raisonnement sur des propriétés de l'autre ensemble de processus stochastiques qui produisent des observations.

Dans la dernière étape de model-checking qui est la validation des propriétés, nous avons vérifié la satisfiabilité ou non des trois propriétés spécifiées en POCTL dans la troisième section sur le modèle HMM, manuellement en utilisant l'algorithme de model-checking pour POCTL sur le modèle HMM vu dans le chapitre 4, et automatiquement en utilisant le HMM\_Model-Checker pour valider le bon fonctionnement de notre outil.

---

# **Conclusion Générale & Perspectives**

---

## 6.1. Conclusion

Dans ce travail, nous nous sommes penchés sur un aspect de vérification automatique à base de modèle probabiliste, une technique plus générale et réaliste, qui continue à attirer davantage d'adeptes dans l'univers de la vérification probabiliste.

Ce travail peut être apprécié sous plusieurs facettes : une introduction pour la première fois du formalisme mathématique « HMM » dans le domaine de la vérification probabiliste ; un enrichissement de model checkers probabilistes existants par des HMMs ; une intégration de plusieurs outils ensemble dans un environnement de vérification; et enfin, une étude de cas.

Ce travail méthodiquement mené, a permis :

- D'introduire pour la première fois le formalisme mathématique « HMM » dans le domaine de la vérification probabiliste car d'une part, les HMMs sont des outils statistiques d'apprentissage simple, riches en propriétés, fondés sur des bases mathématiques solides et ils permettent de modéliser divers systèmes probabilistes à partir d'un ensemble d'observations et dont le nombre d'états du système peut ne pas être connu. Et d'autre part, leur utilisation dans le domaine informatique est très répandue.
- D'enrichir les model-checkers probabilistes existant à notre jour par la conception et la réalisation d'un outil baptisé *HMM\_Model-Checker* dédié au modèle HMM comme un nouveau formalisme de modélisation et de la logique POCTL prolongée de PCTL, comme un nouveau formalisme de spécification. L'outil est implémenté dans l'environnement Netbeans. Il a une interface assez conviviale. A notre connaissance, il n'existe aucun model-checker probabiliste utilisant le HMM et le POCTL.

- D'élèver le niveau d'abstraction de modèle et de la propriété, en intégrant dans notre outil la bibliothèque JaHMM qui offre à l'utilisateur la possibilité de créer un HMM en entrant les paramètres associés au modèle sans le recours à une description du modèle dans un langage de modélisation spécifique comme est le cas dans l'outil PRISM requérant un utilisateur expert dans le domaine.
- De concevoir et d'implémenter dans Netbeans un environnement de vérification des systèmes probabilistes en intégrant plusieurs model-checkers dans une même plateforme pour offrir à l'utilisateur la possibilité de choisir un modèle parmi les différents modèles probabilistes existants afin d'obtenir l'outil souhaité dans la liste.
- De fournir à l'utilisateur une plateforme s'appuyant sur une technologie à « bouton poussoir » qui n'exige pas un degré élevé d'expertise. A l'aide de l'interface graphique, l'utilisateur peut accéder aux différentes opérations du processus de vérification depuis la création du modèle HMM, la saisie de la propriété jusqu'à l'affichage du résultat final de vérification.
- D'expérimenter notre travail sur une étude de cas d'un système embarqué réel qui est le HST et les résultats obtenus sont prometteurs.

## **6.2. Perspectives**

Plusieurs pistes de recherche peuvent être explorées afin d’approfondir le travail présenté dans cette thèse. Nous avons particulièrement identifié quelques points qui nous paraissent intéressants :

- Ajout de nouvelles fonctionnalités à notre outil HMM\_Model-Checker.
- Extension de la vérification à d’autres propriétés.
- Réalisation d’autres études de cas sur d’autres systèmes embarqués réels.
- Le développement de langages de description de systèmes, car la véritable difficulté dans le processus de model-checking réside dans la tâche de modélisation.
- Pour élever le niveau d’abstraction de modèle et de propriété, nous comptons intégrer à notre environnement l’outil pState [76] utilisant les pCharts [78], une version étendue de machines à états hiérarchiques pour la description de modèle et la spécification de propriété en même temps.
- Ajout de nouvelles fonctionnalités à notre environnement telles que : la génération d’un modèle à partir d’un autre, comparaison d’outils afin de choisir le meilleur, etc.

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <b>ACTL</b>              | Action Computation Tree Logic                           |
| <b>BDD</b>               | Binary Decision Diagram                                 |
| <b>CAO</b>               | Conception Assisté par Ordinateur                       |
| <b>CMRM</b>              | Continuous Markov Reward Model                          |
| <b>CSL</b>               | Continuous Stochastic Logic                             |
| <b>CSRL</b>              | Continuous Stochastic Reward Logic                      |
| <b>CTMC</b>              | Continuous-Time Markov Chain                            |
| <b>CMU</b>               | Carnegie-Mellon University                              |
| <b>DMRM</b>              | Discrete Markov Reward Model                            |
| <b>DTMC</b>              | Discrete-Time Markov Chain                              |
| <b>ETMCC</b>             | Erlangen-Twente Markov Chain Checker                    |
| <b>ESA</b>               | European Space Agency                                   |
| <b>FDPO</b>              | Fonction de Distribution de Probabilité d'Observation   |
| <b>GPL</b>               | General Public License                                  |
| <b>GSMP</b>              | Generalized Semi-Markov Process                         |
| <b>GUI</b>               | Graphical User Interface                                |
| <b>HMM</b>               | Hidden Markov model                                     |
| <b>HST</b>               | Hubble Space Telescope                                  |
| <b>IA</b>                | Intelligence Artificielle                               |
| <b>LTL</b>               | Linear temporal logic                                   |
| <b>MDP</b>               | Markov Decision Process                                 |
| <b>MDD</b>               | Model Driven Development                                |
| <b>MOVES</b>             | Software Modelling and Verification                     |
| <b>MRMC</b>              | Markov Reward Model Checker                             |
| <b>MTBDD</b>             | Multi-Terminal Binary Decision Diagram                  |
| <b>NASA</b>              | National Aeronautics and Space Administration           |
| <b>OS</b>                | Operating System                                        |
| <b>PCTL</b>              | Probabilistic Computation Tree Logic                    |
| <b>PC</b>                | Personal Computer                                       |
| <b>PPSE</b>              | Plus Probable Séquence-États                            |
| <b>PLTL</b>              | Probabilistic linear temporal logic                     |
| <b>POCTL</b>             | Branching-time temporal logic Probabilistic Observation |
| <b>PRCTL</b>             | Probabilistic Reward Computation Tree Logic             |
| <b>PRISM</b>             | Probabilistic Symbolic Model Checker                    |
| <b>PSL</b>               | Process Specification Language                          |
| <b>QuaTE<sub>x</sub></b> | Quantative Temporal Expressions                         |
| <b>QoS</b>               | Quality of Service                                      |
| <b>RAM</b>               | Random Access Memory                                    |
| <b>RWTH</b>              | Rheinisch-Westfälische Technische Hochschule            |

---

# **Références Bibliographiques**

---

- [1] Nokovic B., 2016. Verification and implementation of embedded systems from high-level models, Ph.D. thesis, McMaster University, Canada. 164p.
- [2] Rebaia M.L., 2011. Spécification et Vérification des Systèmes Critiques : Extension de l'Environnement VALID pour la Prise en Charge du Temps Réel. Thèse de doctorat en informatique, Université Batna, Algérie, 179p.
- [3] Muhimpundu J., 2014. Analyse de l'erreur dans la vérification probabiliste, Maîtrise en informatique, Université Laval, Québec, Canada. 106p.
- [4] Murat V., 2014. Extensions d'automates d'arbres pour la vérification de systèmes à états infinis. Thèse de doctorat en informatique, Université de Rennes 1, France, 220p.
- [5] Calinescu R., Ghezzi C., Johnson K., Pezzé M., Rafiq Y. & Tamburrelli .G, 2016. Formal verification with confidence intervals to establish quality of service properties of software systems, IEEE transactions on reliability, vol. 65, no. 1, 107–125.
- [6] Nleng C. H., 2014. Modélisation et vérification du flux d'information pour les systèmes orientés objets, Maîtrise ès sciences appliquées, département de génie informatique et génie logiciel, Ecole polytechnique de Montréal, Canada, 136p.
- [7] Moussa M., 2014. Vérification et configuration automatiques de pare-feux par model checking et synthèse de contrôleur, Maîtrise ès sciences appliquées, département de génie informatique et génie logiciel, Ecole polytechnique de Montréal, Canada, 111p.
- [8] Christel B., Joost-Pieter K., & al., 2008. Principles of model checking, volume 26202649. MIT press Cambridge.
- [9] Encrenaz E., 2015. Formal Methods for Functional Verification of Cache-Coherent Systems-on-Chip, Thèse de doctorat, université de Grenoble, France, 153p.
- [10] Parker D., 2002. Implementation of symbolic model checking for probabilistic systems, these de doctorat, University of Birmingham.

- [11] Rabiner L.R., 1989. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition, *Proceedings of the IEEE*, Vol. 77(2), 257-286.
- [12] Jurafsky D. & Martin J.H., 2000. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall.
- [13] Vlontzos J. A. & Kung S. Y., October 1992. Hidden Markov models for character recognition. *IEEE Transactions on Image Processing*, Vol. (1), 539–543.
- [14] Birney E., 2001. Hidden Markov models in biological sequence analysis. *IBM Journal of Research and Development*, Vol. 45(3), 449–454.
- [15] Pevzner P.A., 2000. *Computational Molecular Biology: An Algorithmic Approach*. The MIT Press.
- [16] Salamatian K. & Vaton S., 2001. Hidden markov modeling for network communication channels. In *SIGMETRICS*, ACM Press, 92–101.
- [17] Francois J.M. & Leduc G., 2005. Mobility prediction's influence on QoS in wireless networks: A study on a call admission algorithm. In *3rd International Symposium on Modeling and Optimization in Mobile, Ad-Hoc and Wireless Networks*, IEEE Computer Society, 238–247.
- [18] Zhang L., Hermanns H. & Jansen D. N., 2005. Logic and Model Checking for Hidden Markov Models. *Proceeding of FORTE*, LNCS 3731, 98 -112.
- [19] Hansson H. & Jonsson B., 1994. A logic for reasoning about time and probability, *Formal Aspects of Computing*, Vol. 6(5), 512-535.
- [20] Christel B., Boudewijn H., Holger H. & Joost-Pieter K., 2003. Model-checking algorithms for continuous-time Markov chains. *IEEE Transactions on Software Engineering*, Vol. 29(6), 524–541.

- [21] Andova S., Holger H. & Joost-Pieter K., 2003. Discrete-time rewards model-checked. In Kim G. Larsen and Peter Niebert, editors, *Formal Modeling and Analysis of Timed Systems: First International Workshop (FORMATS'03)*, Vol. 2791 of LNCS, Berlin, Springer, 88–104.
- [22] Baier C., Boudewijn R. H., Holger H. & Joost-Pieter K., 2000. On the logical characterisation of performability properties. In Ugo Montanari, José D. P. Rolim, and Emo Welzl, editors, *Proc. 27th International Colloquium on Automata, Languages and Programming (ICALP '00)*, Vol. 1853 of LNCS, Berlin, Springer, 780–792.
- [23] Boudewijn H., Lucia C., Holger H., Joost-Pieter K. & Christel B., 2002. Model-checking performability properties. In *International Conference on Dependable Systems and Networks (DSN'02)*, Washington D. C., IEEE CS Press, 103–112.
- [24] Gul A. A., José M. & Koushik S., 2005. Pmaude: Rewrite-based specification language for probabilistic object systems. *Electronic Notes in Theoretical Computer Science*, Vol. 153(2), 213–239.
- [25] Holger H., Joost-Pieter K., Joachim M. K. & Markus S., 2000. Towards model checking stochastic process algebra. In Wolfgang Grieskamp, Thomas Santen, and Bill Stoddart, editors, *2nd International Conference on Integrated Formal Methods (IFM'00)*, Vol. 1945 of LNCS, Berlin, Springer, 420–439.
- [26] Nicola R.D. & Vaandrager F.W., 1990. Action versus state based logics for transition systems. *Semantics of Systems of Concurrent Processes*, /LNCS 469, 407-419.
- [27] Clarke E., E. Emerson, & Sistla A., 1986. Automatic verification of finite-state concurrent systems using temporal logics. *ACM Transactions on Programming Languages and Systems*, Vol. 8(2), 244–263.
- [28] Vardi M. Y., 2006. From Church and Prior to PSL. In *25MC*.

- [29] Courcoubetis C. & Yannakakis M., 1995. The Complexity of Probabilistic Verification. *Journal of the ACM*, VOL. 42(4), 857–907.
- [30] Hansson H. & Jonsson B., 1994. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, Vol. 6(5), 512–535.
- [31] Bianco A. & De Alfaro L., 1995. Model checking of probabilistic and nondeterministic systems. In P. Thiagarajan, editor, *Proc. 15th Conference on Foundations of Software Technology and Theoretical Computer Science*, vol. 1026 of LNCS, Springer, 499–513.
- [32] Younes H. L. S. & Simmons R. G., 2002. Probabilistic verification of discrete event systems using acceptance sampling. In Ed Brinksma and Kim Guldstrand Larsen, editors, *Proc. 14th International Conference on Computer Aided Verification (CAV'02)*, Vol. 2404 of LNCS, Berlin, Springer, 223–235.
- [33] Hart S., Sharir M. & Pnueli A., 1983. Termination of probabilistic concurrent programs. *ACM Transactions on Programming Languages and Systems*, Vol. 5(3), 356–380.
- [34] Pnueli A., 1983. On the extremely fair treatment of probabilistic algorithms. In *Proc. 15th Annual ACM Symposium on Theory of Computing*, 278–290.
- [35] Vardi M. Y., 1985. Automatic verification of probabilistic concurrent finite state programs. In *Proc. 26th Annual Symposium on Foundations of Computer Science (FOCS'85)*, IEEE Computer Society Press, 327–338.
- [36] Pnueli A. & Zuck L., 1986. Verification of multiprocess probabilistic protocols. *Distributed Computing*, Vol. 1(1), 53–72.
- [37] Aziz A., Sanwal K., Singhal V. & Brayton R., 1996. Verifying continuous time Markov chains. In R. Alur and T. Henzinger, editors, *Proc. 8th International Conference on Computer Aided Verification (CAV'96)*, vol. 1102 of LNCS, Springer, 269–276.

- [38] Baier C., Katoen J.-P. & Hermanns H., 1999. Approximate symbolic model checking of continuous-time Markov chains. In J. Baeten and S. Mauw, editors, Proc. 10th International Conference on Concurrency Theory (CONCUR'99), vol. 1664 of LNCS, Springer, 146–161.
- [39] Aziz A., Singhal V., Balarin F., Brayton R. & Sangiovanni-Vincentelli A., 1995. It usually works: The temporal logic of stochastic systems. In P. Wolper, editor, Proc. 7th International Conference on Computer Aided Verification (CAV'95), vol. 939 of LNCS, Springer, 155–165.
- [40] Courcoubetis C. & Yannakakis M., 1990. Markov decision processes and regular events. In M. Paterson, editor, Proc. 17th International Colloquium on Automata, Languages and Programming (ICALP'90), Vol. 443 of LNCS, Springer, 336–349.
- [41] Baier C. & Kwiatkowska M. Z., 1998. Model checking for a probabilistic branching time logic with fairness. *Distributed Computing*, Vol.11(3), 125–155.
- [42] Baier C., 1998. On algorithmic verification methods for probabilistic systems. habilitation thesis. University of Mannheim.
- [43] Baier C., Boudewijn R. H., Holger H. & Joost-Pieter K., 2000. Model checking continuous-time Markov chains by transient analysis. Proc. 12th International Conference on Computer Aided Verification (CAV'00), In E. Allen Emerson and A. Prasad Sistla, editors, Vol. 1855 of LNCS, Berlin, Springer, 358–372.
- [44] Joost-Pieter K., Kwiatkowska M. Z., Gethin N. & David P., 2001. Faster and symbolic CTMC model checking. In Luca de Alfaro and Stephen Gilmore, editors, Proc. 1st Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification (PAPM/PROBMIV'01), Vol. 2165 of LNCS, Berlin, Springer, 23–38.
- [45] Marta Z. K., Gethin N., & David P., 2002. PRISM: Probabilistic symbolic model checker. In Tony Field, Peter G. Harrison, Jeremy T. Bradley, & Uli Harder, editors, Proc.

12th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation (TOOLS'02), Vol. 2324 of LNCS, Berlin, Springer, 200–204.

[46] Herault T., Lassaigne R., Magniette F. & Peyronnet S., 2004. Approximate Probabilistic Model Checking. In Proceedings of Fifth International VMCAI'04, Vol. 2937 of LNCS, 73–84.

[47] Hermanns H., Katoen J.-P., Meyer-Kayser J. & Siegle M., 2000. A Markov chain model checker. In Proc. 6th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Vol. 1785 of LNCS, Springer, 347–362.

[48] <http://www.cs.utwente.nl/~zapreevis/mrmc/>.

[49] Håkan L. S. Y., 2005. Ymer. A statistical model checker. In Kousha Etessami and Sriram K. Rajamani, editors, Proc. 17th International Conference on Computer Aided Verification (CAV'05), Vol. 3576 of LNCS, Berlin, Springer, 429–433.

[50] Ward W., 1980. Continuity of generalized semi-markov processes. Mathematics of Operations Research, Vol. 5, 494–501.

[51] <http://osl.cs.uiuc.edu/~ksen/vesta2>.

[52] <http://www.uppaal.com>

[53] Oldenkamp H.A., 2007. Probabilistic model checking: a comparison of tools. Master's thesis, University of Twente, Netherlands. 108p.

[54] Kurshan R. P., 1997. Formal Verification in a Commercial Setting. In ACM IEEE DA'97, ACM.

[55] Sergio P., 1984. Sparse Matrix Technology. Academic Press, London, UK. ISBN 0-12-557580-7.

- [56] Youcef S., Bernard P. & William S., 1992. Numerical methods in Markov chain modeling. *Operations Research*, Vol. 40(6), 1156–1179.
- [57] Fujita M., McGeer P.C. & Yang J. C.-Y., 1997. Multi-terminal binary decision diagrams: An efficient data structure for matrix representation. *Form. Methods Syst. Des.*, Vol. 10(2-3), 149–169.
- [58] De Alfaro L., Kwiatkowska M. Z., Gethin N., David P., & Roberto S., 2000. Symbolic model checking of probabilistic processes using MTBDDs and the Kronecker representation. In Susanne Graf and Michael I. Schwartzbach, editors, *Proc. 6th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'00)*, Vol.1785 of LNCS, Berlin, Springer, 395–410.
- [59] Hart S. & Sharir M., 1986. Probabilistic temporal logics for finite and bounded models. In *Proc. 16th Annual ACM Symposium on Theory of Computing*, pages 1–13, 1984. The full version “Probabilistic Propositional Temporal Logic” is available in *Information and Control*, Vol. 70:(2/3), 97–155.
- [60] Holger H., Joost-Pieter K., Joachim M. K. & Markus S., 2003. A tool for model-checking Markov chains. *Int. J. on Softw. for Technology Transfer (STTT)*, Vol. 4(2), 153–172.
- [61] Henk C.T. & Veldman R., 2000. A fast algorithm for the transient reward distribution in continuous-time Markov chains. In *Operation Research Letters*, Vol. 26, 155–158.
- [62] Muhammad A. Q. & William H. S., 1996. A new methodology for calculating distributions of reward accumulated during a finite interval. In *Proc. of the 26th International Symposium on Fault-Tolerant Computing (FTCS'96)*, Japan, 116–125.
- [63] Koushik S., Mahesh V., & Gul A., 2004. Statistical model checking of black-box probabilistic systems. In Rajeev Alur and Doron Peled, editors, *Proc. 16th International Conference on Computer Aided Verification (CAV'04)*, Vol. 3114 of LNCS, Berlin, Springer, 202–215.

- [64] Koushik S., Mahesh V., & Gul A., 2005. On statistical model checking of stochastic systems. In Kousha Etessami and Sriram K. Rajamani, editors, Proc. 17th International Conference on Computer Aided Verification (CAV'05), Vol. 3576 of LNCS, Berlin, Springer, 266–280.
- [65] Web site: <http://jahmm.googlecode.com/>.
- [66] Ferroum A. & Boudour R., 2010. HMM\_Model-Checker : Un vérificateur probabiliste pour systèmes embarqués. Conférence nationale sur la modélisation et la simulation informatiques des systèmes industriels (MSISI'2010), Université de 20 Aout 1955, Skikda, Algérie.
- [67] Joseph G. & David G., 2008. UML 2 analyse et conception. Dunod, Paris.
- [68] Laurent A., 2009. UML 2 de l'apprentissage à la pratique. Ellipses, Ed.
- [69] <https://fr.coursera.org/learn/init-prog-java>.
- [70] <https://www.mysql.com/fr/>.
- [71] [https:// www.netbeans.org/community/releases/82/install.html](https://www.netbeans.org/community/releases/82/install.html).
- [72] <https://www.materiel-informatique.be/easyphp.php>
- [73] Ferroum A. & Boudour R., 2016. A tool for automatic verification of Embedded systems. 3rd International Conference on Embedded Systems in Telecommunications and Instrumentation (ICESTI'16), Annaba, Algeria.
- [74] Tatarewicz J.N., 2001. The Hubble Space Telescope Servicing Mission. In: From engineering science to big science. (Eds) J. Henry – 365-396.

- [75] Segala R., 1995. Modelling and Verification of Randomized Distributed Real Time Systems, Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA. 283p.
- [76] Nokovic B. & Sekerinski E., 2013. pState: A probabilistic statecharts translator, Proceedings of the Embedded Computing (MECO), 2nd Mediterranean Conference on Embedded Computing, 29-32.
- [77] Nokovic B. & Sekerinski E., 2015. A Holistic Approach in Embedded System Development. Proceedings of the International Workshop on Formal Integrated Development Environment, Canada, 72-85.
- [78] Nokovic B. & Sekerinski E., 2014. Verification and Code Generation for Timed Transitions in pCharts. Proceedings of the International Conference on Computer Science, Software Engineering, Montreal, Canada, 1-10.