HMM_Model-Checker pour la vérification probabiliste

HMM_Model-Checker for Probabilistic Verification

Assia Ferroum* & Rachid Boudour

Laboratoire de systèmes embarqués –LASE-, Université Badji Mokhtar.
BP 12, Annaba, 23000, Algérie.

Soumis le : 19/12/2016 *Révisé le : 18/06/2017* *Accepté le : 04/07/2017*

ملخص

لا يزال التحقق الاحتمالي للنظم يجلب المزيد والمزيد من الأتباع في الأوساط البحثية. ليكن لدينا نموذج احتمالي، صيغة من المنطق الزمني لإعطاء مواصفات نظام وخوارزمية استكشاف للتحقق مما إذا كانتهذه الأخيرة محققة أم لا، الهدف هو الوصول إلىنظامآمن. يقدم هذا العمل بعض الجوانب المبتكرة في هذا المجال: نموذج ماركوف الخفي لكفموذج جديد في مجال التحقق ا لاحتمالي، تنفيذ خوارزمية التحقق ا لاحتمالي للنظم الخاصة بالمنطق*POCTL*من أجل ا لتحقق من الخصائص على نموذج ماركوف الخفي، تصميم وتنفيذ في بيئة ن اتبينهلمحقق احتمالي للنظم يسمى " محقق احتمالي للنظم لنموذج ماركوف الخفي "، و لرفع مستوى تجريد النموذج قمنابمج المكتبةالخاصة بنموذج ماركوف الخفيإلى أدا تنا و التي تسمح بخلقنموذج ماركوف الخفيمن المعطيات المقدمة من قبل المستخدم، تليهادراسة حالة لنظام مدمج حقيقي وهو تلسكوب هابل الفضائي، لتعزيز كلماتنا.

الكلمات المفتاحية: التحقق الاحتمالي للنظم–أنظمةمدمجة –نموذج ماركوف الخفي–*POCTL*–تلسكوب هابل الفضائي.

Résumé

La vérification probabiliste des systèmes embarqués continue de compter de plus en plus d'adeptes dans la communauté de chercheurs. Étant donné un modèle probabiliste, une formule de la logique temporelle, décrivant une propriété du système et un algorithme d'exploration permettant de vérifier si cette dernière est satisfaite ou non, la finalité est d'arriver à un système fiable. Ce travail présente quelques aspects novateurs dans le domaine: le modèle de Markov caché (HMM) comme nouveau modèle en vérification probabiliste, une implémentation d'un algorithme pour POCTL afin de permettre la vérification des propriétés sur le modèle HMM, une conception et une implémentation dans l'environnement Netbeans d'un vérificateur probabiliste baptisé «HMM_Model-Checker», et pour élever le niveau d'abstraction du modèle, nous avons intégré la bibliothèque JaHMM qui permet de créer un HMM à partir de paramètres introduits par l'utilisateur, Une étude de cas d'un système embarqué réel: le télescope spatial Hubble, a suivi pour consolider nos propos.

Mots clés : Vérification de modèle probabiliste–Systèmes embarqués–Modèle de Markov caché–Logique probabiliste –Télescope Hubble.

Abstract

Probabilistic verification for embedded systems continues to attract more and more followers in the research community. Given a probabilistic model, a formula of temporal logic, describing a property of a system and an exploration algorithm to check whether the property is satisfied or not, the purpose is to achieve a safe system. This work presents some innovative aspects such as the Hidden Markov Model (HMM) as a new model, an implementation of the algorithm for POCTL to allow properties verification on the HMM model, a design and implementation in the Netbeans environment of a probabilistic model-checker called "HMM_Model-Checker", and to raise the level of model abstraction, we have integrated the JaHMM library which allows to create an HMM from the parameters introduced by the user. A real case study of embedded system: the Hubble Space Telescope, followed in order to consolidate our words.

Key words: Probabilistic model-checking –Embedded systems–HMM–POCTL –Hubble telescope.

* *Auteur correspondant : assiaferroum@yahoo.fr*

1. INTRODUCTION

La plupart des systèmes informatiques utilisés de nos jours sont des systèmes embarqués: téléphones mobiles, véhicules, avions, télescopes, missiles, etc. Un système embarqué interagit largement avec son environnement [1]. Il est important de s'assurer de la sureté 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 occasionnent un préjudice économique important, difficile à supporter pour les personnes voire les états.

La liste des catastrophes connues de tels systèmes est très longue (fortes radiations dues au système Thérac 25, explosion d'Ariane 5, bogue Heart bleed permettant de recueillir des informations secrètes des services web bien connus comme Yahoo, Flickr, Doodle ou encore Zimbra, etc.) [2,3]. 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 [4]. Cependant, il existe un nombre limité d'outils disponibles pour le model-checking probabiliste mais qui n'ont guère abordé la modélisation HMM (Hidden Markov Model), dont la puissance mathématique n'est plus à démontrer et qui a fait ses preuves dans d'autres domaines comme la reconnaissance de la parole, etc.

Dans ce contexte, notre travail va reposer sur la trilogie : un *modèle* HMM modélisant le système réel, une *logique* probabiliste POCTL (Branching-time Temporal logic Probabilistic Observation) spécifiant la propriété attendue et un *algorithme* d'exploration permettant de comparer la propriété à l'état courant de l'espace des états du modèle. Ces aspects ont convergé vers l'implémentation d'un outil nommé HMM_model-checker, appliqué au Télescope spatial Hubble HST (Hubble Space Telescope). Notre contribution diffère des autres approches au niveau des trois entités de base: modèle, logique temporelle probabiliste et algorithme d'exploration.

Ce papier décrit en section 2 l'état de l'art sur le model-checking probabiliste, la section 3 présente notre approche de conception. Dans la section 4, est présenté l'outil conçu et réalisé HMM_Model-Checker, suivi en section 5 par une étude de cas d'un système embarqué réel: HST (Hubble Space Telescope). La section 6 analyse les résultats et les situe par rapport à la littérature. Une conclusion succincte et des perspectives viendront achever ce papier.

2. TRAVAUX CONNEXES

2.1 Model-checking probabiliste

Le model-checking probabiliste est similaire au model-checking classique[5].La différence majeure est que le modèle probabiliste contient des informations supplémentaires sur la probabilité des transitions entre états, ou pour être plus précis, il peut modéliser le comportement stochastique [6,7]. Le model-checking permet de décider si un modèle M satisfait une propriété φ, exprimée comme une formule logique. La satisfaction de φ par M est notée M |= φ. Par extension, le système satisfait ses spécifications si le modèle formel de son comportement satisfait les formules logiques exprimant les propriétés attendues [8].

2.2 Outils de vérification probabilistes

Comme les outils classiques, le model-checker probabiliste prend en entrée un modèle probabiliste DTMC (Discrete Time Markov Chain), CTMC(Continuous Time Markov Chain), etc.décrivant le système et une formule de logique temporelle PCTL (Probabilistic CTL) par exemple, spécifiant la propriété et retourne en sortie une décision : oui, non ou une certaine probabilité. Si le résultat est non, le système ne satisfait pas la propriété [9, 10], plus de détails sont donnés en section 4.
Il existe un nombre limité de model checkers probabilistes. Nous référençons six outils académiques des plus importants. Ils sont accessibles librement et gratuitement via internet: APMC(†), PRISM(‡), ETMCC(§), MRMC(**), YMER(††), VESTA(‡‡) . Nous remarquons qu'aucun de ces model-checkers

(†) - http://www.lri.fr/~syp/APMC.
(‡) -http://www.cs.bham.ac.uk/~dxp/prism/.
(§) -http://www.informatik.uni-erlangen.de/etmcc/.

n'utilise les HMMs comme un formalisme de modélisation et POCTL comme un formalisme de spécification des propriétés. Ainsi, à notre connaissance, notre HMM_Model-Checker serait vraisemblablement le premier model-checker à utiliser HMMs.

3. APPROCHE PROPOSEE

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 HMM [11] 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 le modèle HMM, il est nécessaire d'identifier une ou plusieurs propriétés du modèle sous forme de formules dePOCTL[12].La vérification de la satisfaction des propriétés est assurée par l'algorithme de model-checking[12] pour POCTL sur un HMM, adapté de celui pour PCTL sur un DTMC [13], 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 [13] où l'opérateur Nextest rendu porteur d'une contrainte d'observation. Nous décrirons chacune des entités:

3.1 HMMs

Formellement, un HMM est un sextuplet H, défini comme suit :H= (S, P, L,θ, μ, π)
Où :

- S, ensemble composé de n états :**S** = $\{s_1, s_2, \ldots s_n\}$. L'état courant du HMM à l'instant t est noté q_t ($q_t \in$ S);
- M, symboles observables dans chaque état. L'ensemble des observations est noté **θ**= $\{O_1, O_2, \ldots, O_M\}$. Un élément V_t de **θ**désigne un symbole observé à l'instant t;
- Une matrice **P** de probabilités de transitions entre les états de la chaîne :
$\forall i,j \in [1 .. n]$, $\forall t \in [1 ..T]$: $a_{ij} = P(i, j) = p(q_{t+1} = s_j \mid q_t = s_i)$ avec : $a_{ij} \geq 0, \forall$ i, j et$\sum_{j=1}^{n} a_{ij} = 1$;
- Une matrice **μ**de distribution des probabilités d'observations : $\mu_j(k)$ est la probabilité d'observerle symbole O_k quand le modèle se trouve dans l'état j, soit : $\mu_j(k) = P(V_t = O_k \mid q_t = s_j)$, $1 \leq j \leq n$, $1 \leq k \leq M, \mu(k) \geq 0$ $\forall$j, k et$\sum_{k=1}^{M} \mu_j(k) = 1$;
- Un vecteur **π** de distribution des probabilités de transitions initiales **π**= $\{\pi_i\}$, i =1,.., n ; π_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 : $\pi_i \geq 0 \forall i$ et $\sum_{i=1}^{n} \pi_i = 1$;
- Et finalement une fonction d'étiquetage $L : S \rightarrow 2^{AP}$.

3.2 Logique POCTL

Les logiques temporelles fournissent une manière de raisonnement au sujet du comportement qui change au cours du temps. Nous avons adopté la logique temporelle POCTL dans notre approche car d'une part, POCTL est fondamentalement une prolongation de PCTL où l'opérateur next est équipé d'une contrainte d'observation et d'autre part, POCTL peut également être considéré comme une variante de la logique temporelle ACTL proposée par De Nicola et autres [14], dans laquelle l'opérateur *next* habituel est prolongé pour contraindre l'étiquette d'action de la transition.

Dans un modèle HMM, les états sont cachés et les processus cachés produisent un ordre des observations. Avec POCTL nous pouvons énoncer des propriétés sur des HMMs telles que" il y a au moins une probabilité de 90 % que le modèle produit une séquence d'observations donnée". Il est composé des formules d'état, de formules de chemin et de formules d'état de croyance nous permettant de spécifier des propriétés sur les HMMs. De cette façon, nous pouvons énoncer des propriétés au-dessus des observations, par exemple, $X_o\varphi$ signifie que la prochaine observation est o et le chemin suivant satisfait φ.La syntaxe de base de l'expression d'une propriété (prop) dans notre approche est comme suit, l'ajout est mis en gras:

(**) -http://www.cs.utwente.nl/~zapreevis/mrmc/
(††) -http://www.cs.cmu.edu/~lorens.
(‡‡) -http://osl.cs.uiuc.edu/~ksen/vesta2/.

(prop) ::= true | false | (expr) | **(croyanceprop) |**
! (PROP) |
(PROP) & (PROP) |
(prop) | (prop) |
(prop) => (prop) |
(croyanceprop) ::= $P_{(op)(p)}$ [(pathprop)]|
! (croyanceprop) |
(croyanceprop) & (croyanceprop) |
(pathprop) ::= $X_{(\Omega)}$(prop) |(prop) U (prop) |
(prop) $U_{(time)}$ (prop) |
(time) ::= >=(t) | <=(t) | [(t) , (t)] |

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 un entier non négatif,(Ω) est un ensemble d'observations.

Une propriété est évaluée en respectant un état du modèle. Pour la syntaxe donnée précédemment, toutes les propriétés s'évaluent aux valeurs booléennes, c.-à-d. pour n'importe quel état s d'un modèle HMM, une propriété est évaluée soit « true », soit « false » dans cet état. En équivalence, elle est satisfaite ou non dans cet 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) évaluée à « true »,! (prop) est « true » si (prop) n'est pas « true »,(prop1) & (prop2) est « true » si (prop1) est « true » and (prop2) est « true »,(prop) | (prop) est « true » si (prop1) est « true » or (prop2) est « true », (prop) => (prop) est « true » si (prop1) implique (prop2).

L'opérateur principal pour spécifier les propriétés est l'opérateur P qui permet de raisonner sur la probabilité qu'un certain type de comportement est observé. Informellement, la propriété : $P_{(op)(p)}$[(pathprop)] est satisfaite 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) ».

Il existe trois types de propriétés de chemin (pathprop) pouvant être utilisés par l'opérateur P :

« Next » : $X_{(Ø)}$, « Until » : U et « bounded Until » : $U_{(time)}$.

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 $\pi = s_0, s_1, s_2,....,s_n$. La sémantique de chacun des trois types de propriétés de chemin π est présentée ci-dessous :

3.2.1 Les propriétés de chemins next ($X_{(\Omega)}$ (prop))

- Cas 1 : Ω = Ø (l'ensemble vide) => $X_{(\Omega)}$ (prop)= X(prop)

La propriété X (prop) est « true » pour le chemin π 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 : $P_{<0.01}$ [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 à 0.01 ».

- Cas 2 : Ω= O / O ⊆ Θ => $X_{(\Omega)}$ (prop)= X_O(prop)

X_O(prop) signifie que la prochaine observation est O et le chemin suivant satisfait (prop). Un exemple de ce type de propriété est : $P_{<=0.2}(X_{O1}y=1)$ qui signifie : « La probabilité que la prochaine observation est O_1et le chemin suivant satisfait(y=1) est inférieure ou égale à 0.2 ».

3.2.2 Les propriétés de chemins Until

La propriété [(prop1) U (prop2)] est « true » pour le chemin π 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 : $P_{>0.5}$ [trueUy=1] qu'est "true" dans un état si « la probabilité que y est éventuellement égale à 1 est supérieure à 0.5 ».

3.2.3 Les propriétés de chemins BoundedUntil

Les propriétés de chemin « BoundedUntil » sont une généralisation des propriétés « Until » où une limite additionnelle est imposée sur le temps auquel le second argument (prop2) doit la satisfaire. Dans une propriété de chemin BoundedUntil « (prop1) $U_{(time)}$(prop2) », la spécification de l'intervalle de temps (time) doit être de la forme : « <=(t) » où (t) est une expression évaluée à un entier non-négatif. La propriété « (prop1) $U_{<= (t)}$ (prop2) » est « true » pour le chemin π si pour un entier i<=(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 : $P_{>=0.98}$[true $U_{<=7}$ y=4] qui signifie : « la probabilité d'être égale à 4 et à moins de 7 unités de temps est supérieure où gale à 0.98 ».

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.

3.3 Algorithme d'exploration pour POCTL

La dernière entité de notre approche est l'algorithme d'exploration, disposant de deux entrées, le modèle HMM et la propriété à vérifier spécifiée en POCTL où au niveau de chaque état, vérifie si la propriété est satisfaite ou non et permet de passer d'un état au suivant. Après épuisement de tout l'espace d'états (cet espace présente souvent un problème d'explosion combinatoire du nombre d'états), il fournit un résultat: propriété vérifiée ou pas. Pour ce faire, on a implémenté l'algorithme Sat (Φ) de model-checking probabiliste pour POCTL présenté dans [12]. Ce dernier renvoie l'ensemble des états de H qui vérifient la formule Φ. Il procède de manière récursive pour déterminer cet ensemble.

La vérification d'une formule ϕ sur un modèle H consiste en une exploration judicieuse de l'espace d'états basée sur la sémantique des opérateurs logiques formant ϕ. 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, ¬ q ou p ∧ 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 en déduire ensuite la satisfaction. Ainsi, les calculs se réduisent à deux appels de la fonction L et quelques opérations sur les ensembles (intersection et différence). En d'autres termes, cette approche se veut une implémentation directe du calcul de Sat (ϕ), tel qu'il est donné par la sémantique de POCTL [12].Dans ce qui suit, nous allons présenter les algorithmes nécessaires pour la vérification.

L'algorithme de vérification prend en entrée un modèle HMM H et une formule ϕ, et renvoie l'ensemble des états de H qui vérifient la formule ϕ. Il procède de manière récursive pour déterminer l'ensemble Sat. Voici en détail l'algorithme Sat (H,ϕ) :

```
Algorithme Sat (HMM H, Formuleϕ)
Dépendant deϕ , Choisir :
Cas a
Retourner Sat(a) :{ s∈S /a∈L(s) } ;
Cas :¬ ϕ1
Retourner S/Sat(ϕ1) ;
Cas ϕ1∧ϕ2
Retourner Sat(ϕ1) ∧Sat(ϕ2) ;
Cas P~p(ψ)
Retourner { s∈S /Ps(ψ)~p} ;
Fin Choisir ;
Fin Algorithme.
```

Figure1. Algorithme de Model-checking pour POCTL sur un HMM

Nous verrons clairement que l'algorithme (Fig. 1) est une implantation directe de ce qui est décrit dans la sémantique de POCTL [12]. 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 : $\{s \in S \setminus Ps(\psi) \sim p\}$. Pour ce faire, nous utilisons deux sous-procédures :

La première, calcule Probabilité(s,s'), $s' \in Sat(\phi)$, retourne une fonction de S dans [0, 1] qui donne la probabilité Ps (ψ), pour tout $s \in S$; la deuxième sous-procédure prend une fonction et une valeur et retourne l'ensemble des états pour lesquels Ps (ψ)~p. Étant donné que la fonction ici est la fonction de probabilité P~p (ψ), la sous-procédure retourne l'ensemble des états telle que la probabilité d'aller vers Sat(s') avec la probabilité p indiquée par l'opérateur borné ~p.

3.3.1 Le calcul de Ps (ψ)

Dépendant de ψ, choisir :

- **Cas $X_\Omega\phi$:** Pour tout $s \in S$, il est facile de montrer que [12] : $p_s(X_\Omega\phi) = \mu_s(\Omega) . \sum_{s' \in Sat(\phi)} p(s,s')$, Tq $\Omega \subseteq \Theta$, un sous ensemble d'observations : $\mu_s(\Omega) = \sum_{o \in \Omega} \mu_s(o)$ et $Sat(\phi) = \{s \in S \setminus s \models \phi\}$.

- ***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 : $S^{yes} = Sat(\phi 2)$, $S^{no} = S/(Sat(\phi 1) \wedge Sat(\phi 2))$, $S^{?} = S/(S^{yes} \cup S^{no})$.

Pour tout $s \in S$ faire : $x_s = 0$ si $s \in S^{no}$, $x_s = 1$ si $s \in S^{yes}$, $x_s = \sum_{s' \in S} p(s,s').x_{s'}$ si $s \in S^{?}$.

```
Prob0(sat(ϕ1),sat(ϕ2));
        R := sat(ϕ2);
        done :=false;
        while (done = false);
        R':=RU {s ∈ sat(ϕ1)/∃ s' ∈ R.
        p(s,s')>0 };
        If (R' = R) then done :=true;
        R := R';
        end while;
return S / R;
```

Figure 2. Algorithme Prob0

```
Prob1 (sat (ϕ1),sat(ϕ2),s^no);
R:= S^no;
Done := false;
While (done =false);
R':=RU{ s∈( sat (ϕ1)\ (sat(ϕ2)) /∃ s' ∈ R.P(s,s')>0 };
If (R'=R )then done:=true;
R:=R';
End while;
Return S/R;
```

Figure 3. Algorithme Prob1

Prob0 (Fig. 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 (Fig. 3) calcule l'ensemble S^{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^{no} qui contient les états satisfaisant $\phi 1$ mais pas $\phi 2$.

Ces deux algorithmes formant la première partie de calcul $p_s(\phi 1 \, U \, \phi 2)$. Pour cette raison, nous leur faisons référence comme algorithmes de pré-computation. 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 $p_s(\phi 1 \, U \, \phi 2)$ arrive à être égal 0 ou 1 pour tous les états (c-à-d , $S^{no} \cup S^{yes} = S$), il suffit d'utiliser ces algorithmes.

Pour les propriétés avec un lien arbitraire ~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 typique calcule seulement une approximation et peut être soumis aussi à des erreurs d'arrondi. Nous avons besoin encore de calculer $p_s(\phi 1 U \phi 2)$ pour les états restants $S^{?} = S \setminus (S^{no} \cup S^{yes})$. Cela peut être fait en résolvant le système d'équations linéaire donné par l'équation (1) en considérant les variables $\{x_s \setminus s \in S\} / x_s = p_s(\phi 1 U \phi 2)$.

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

Nous pouvons le réécrire de façon traditionnelle : **A**. x = **b** tel que :**A** = **I-P'** où :**I,** est la matrice

d'identité et**P',** définie par l'équation (2) comme suit :

$$\mathbf{p}'(s,s') = \begin{cases} p(s,s'), si\ s \in S \\ 0 \quad\quad , sinon \end{cases} \quad (2)$$

b, un vecteur colonne sur les états décrit par l'équation (3) :

$$\mathbf{b}(s) = \begin{cases} 1\ , \ si\ s \in S^{yes} \\ 0\ , \ sinon \end{cases} (3)$$

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

4. PRESENTATION DE HMM_MODELCHECKER

Comme il a été présenté en introduction, notre travail consiste à concevoir et implémenter un outil de vérification formelle à l'aide du model-checking probabiliste appliqué au télescope spatial Hubble. Cet outil sera basé sur le modèle HMM fini (espace d'états fini) pour la modélisation du système probabiliste et la logique POCTL pour la spécification des propriétés attendues (Fig.4).

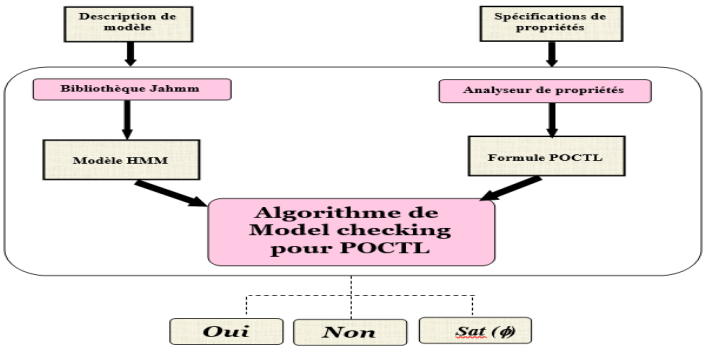


Figure 4. Architecture de l'outil HMM Model Checker

Notre outil baptisé HMM_Model-Checker dont l'architecture est présentée en figure 4, fait appel aux trois entités de base du model-checking: le modèle HMM, la spécification en POCTL et l'algorithme d'exploration. Il a été implémenté sous l'environnement Netbeans, fait appel à la bibliothèque JaHMM(§§) pour une création aisée du modèle et l'analyseur syntaxique conçu pour la correction syntaxique de la propriété : l'utilisateur a la possibilité d'entrer son modèle par le chargement d'un fichier (.hmm), ou par construction en utilisant une boîte de dialogue. A notre connaissance, les outils probabilistes de l'état de l'art utilisent un langage impératif : C, C++, Java, etc. Notre vérificateur formel est un programme complexe qui rassemble plusieurs fonctionnalités, voici quelques-unes des plus importantes:

Charger un HMM H à partir d'un fichier (.hmm), créer un HMM H, faire l'apprentissage des paramètres d'un modèle HMM H, sauvegarder un HMM H dans un fichier (.hmm). charger une séquence d'observations à partir d'un fichier (.seq), générer une séquence d'observations en utilisant un HMM, 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, sauvegarder une séquence d'observations dans un fichier (.seq), rechercher 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, saisir à l'écran une formule de la logique POCTL, Contrôler et assister l'utilisateur à saisir la formule POCTL correctement et rapidement, saisir à l'écran un commentaire sur la propriété à vérifier. calculerSat(Ø), l'ensemble des états qui satisfont la formule et vérifier H |= Ø en accomplissant les actions suivantes :

si Sat (Φ) = S Alors retourner OUI ; sinon si Sat (Φ) = Ø Alors retourner ; sinon retourner Sat (Φ), (l'ensemble des états satisfaisant la propriété).

(§§)- http://jaMMC.googlecode.com/.

En exécutant l'application, l'interface principale illustrée en figure 5 s'affiche à l'écran. L'utilisateur peut entrer son modèle et sa propriété en profitant des facilités de l'aide proposée par notre outil. Il est à noter que l'outil proposé permet de vérifier de manière analogue aux autres model- checkers probabilistes des formules logiques (proposition atomique, conjonction de deux propositions atomiques, négation d'une proposition atomique, Next, Until, boundeduntil).

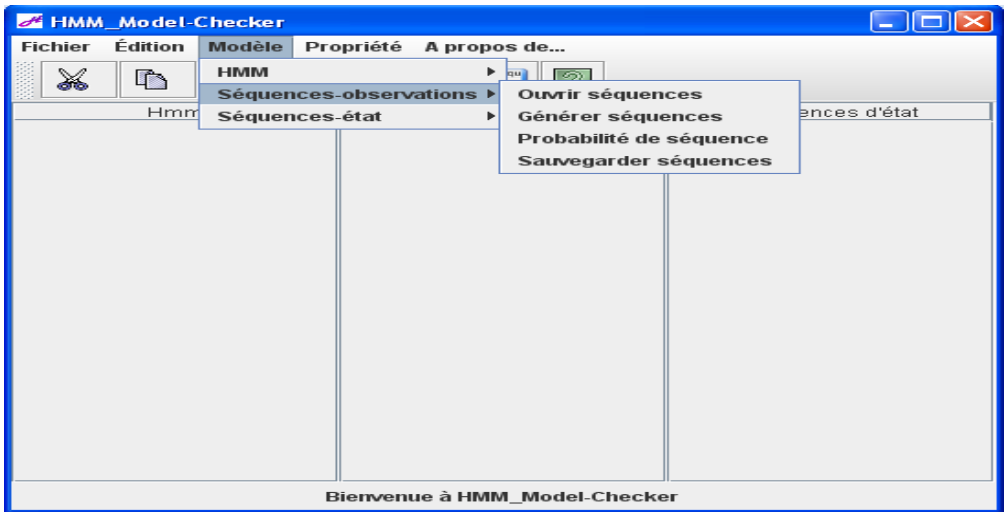


Figure 5. Interface graphique de l'outil HMM_Model Checker

Au meilleur de nos connaissances, il est le premier à raisonner sur les propriétés de l'autre ensemble de processus stochastiques qui produisent des observations.

5. EXPERIMENTATION SUR HST

Dans cette section, nous présentons une étude de cas d'un système embarqué réel qui est le HST [15] pour appuyer notre approche. Ce cas est issu d'une adaptation du modèleDTMC du HST présenté en [16]. IL modélise le comportement d'échec. Pour faciliter la compréhension, nous décrivons brièvement le système réel, constitué de composants qui peuvent probablement échouer, puis sa modélisation en HMM, de sorte que nous puissions prévoir des requêtes comme : « Quelle est la probabilité que la prochaine observation est une observation d'échec d'un gyroscope ? ».

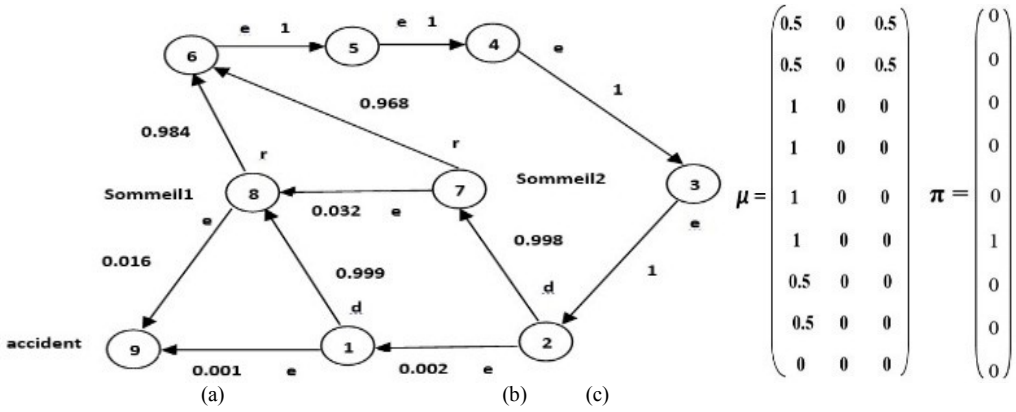
5.1 Description du système HST

Le télescope spatial Hubble dispose d'une unité de contrôle avec six gyroscopes pour la visée du télescope. La redondance est une stratégie saillante de la conception des systèmes embarqués critiques tels que les télescopes spatiaux, puisque l'exécution des réparations peut ne pas être évidente, c'est pourquoi le télescope est conçu de manière à rester en bon fonctionnement avec un minimum de trois de ses six gyroscopes opérationnels. Le télescope se transforme en mode sommeil quand le minimum requis n'est pas atteint, signifiant que les réparations sont nécessaires (sommeil 2 : 2 gyroscopes opérationnels, sommeil 1 : 1 gyroscope opérationnel). Si aucun des gyroscopes n'est opérationnel, le télescope se brise [10].

5.2 Modèle HMM

Le modèle a un total de neuf états, comme illustré en figure 6 : l'ensemble de tous les états est : S= {1, 2, 3, 4, 5, 6, 7, 8, 9}, l'état 6 est considéré le seul état initial où tous les gyroscopes sont opérationnels ; c'est également l'état du télescope après une mission réussie de réparation, l'ensemble des observations O = {e, r, s}, où : e (échec), représente une observation d'un échec de gyroscope ; r (réparation) représente une observation de mission de réparation entreprise avec succès ; s (sommeil) représente une observation signifiant que le télescope va entrer dans le mode sommeil, les propositions atomiques qui sont « true » dans chaque état : les états de 1 à 6, la proposition atomique est le numéro de l'état représentant le nombre de gyroscopes opérationnels (S1: un, S2: deux, S3: trois,

S4: quatre, S5: cinq, S6 : six), l'état S7est marqué sommeil2 : le télescope est en mode sommeil au niveau 2, l'état S8 est marqué sommeil1 qui signifie que le télescope est en sommeil au niveau 1, l'état S9 est marqué accident qui signifie que le télescope est en accident.



(a) (b) (c)

Figure 6. Modèle HMM du HST : (a) Le système de transition avec l'ensemble des observations, (b) Matrice des probabilités des observations et (c) Distribution initiale.

5.3 Vérification de propriété

Pour nous permettre la comparaison avec le travail de Nokovic [17], nous avons considéré la même propriété à vérifier : « 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 ».

5.3.1 Spécification de la propriété en POCTL

$P_{<= 0.008}$ ($X_E$accident)

5.3.2 Application manuelle de l'algorithme

s satisfait $P_{<=0.008}$ (X_e accident) $ssip_s(X_e$ accident) <= 0.008.

On a: $p_s(X_\Omega\phi) = \mu_s(\Omega)$. $\sum_{s' \in sat(\phi)} p(s,s')$, ϕ= 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)

p_{s1} [X_e (accident)] = $\mu_{s1}(e)$. $\sum_{s' \in sat(accident)} p(s_1,s')$= 0.5 (0.001) = 0.0005 <= 0.008

p_{s2} [X_e (accident)] = $\mu_{s2}(e)$. $\sum_{s' \in sat(accident)} p(s_2,s')$ = 0.5 (0) = 0 <= 0.008

p_{s3} [X_e (accident)] = $\mu_{s3}(e)$. $\sum_{s' \in sat(accident)} p(s_3,s')$ = 1 (0) = 0 <= 0.008

p_{s4} [X_e (accident)] = $\mu_{s4}(e)$. $\sum_{s' \in sat(accident)} p(s_4,s')$ = 1 (0) = 0 <= 0.008

p_{s5} [X_e (accident)] = $\mu_{s5}(e)$. $\sum_{s' \in sat(accident)} p(s_5,s')$= 1 (0) = 0 <= 0.008

p_{s6} [X_e (accident)] = $\mu_{s6}(e)$. $\sum_{s' \in sat(accident)} p(s_6,s')$ = 1 (0) = 0 <= 0.008

p_{s7} [X_e (accident)] = $\mu_{s7}(e)$. $\sum_{s' \in sat(accident)} p(s_7,s')$ = 0.5 (0) = 0 <= 0.008

p_{s8}[X_e (accident)] = $\mu_{s8}(e)$. $\sum_{s' \in sat(accident)} p(s_8,s')$ = 0.5 (0.016) = 0.008 <= 0.008

p_{s9}[X_e (accident)] = $\mu_{s9}(e)$. $\sum_{s' \in sat(accident)} p(s_9,s')$ = 0 (0)= 0 <= 0.008

Nous venons de voir d'après la trace de l'algorithme que la formule $P_{<= 0.008}$ [X_e (accident)] est satisfaite dans tous les états du modèle (s_i , i=1 .. 9). Ainsi la formule (X_e accident) voire la propriété (sûreté) est vérifiée pour tout le modèle, et par conséquent nous déduisons la sûreté de fonctionnement du système.

5.3.3 Vérification automatique

Après implémentation et lancement de l'exécution de l'outil sur les mêmes modèles et propriété, nous obtenons la fenêtre montrée sur la figure 7.

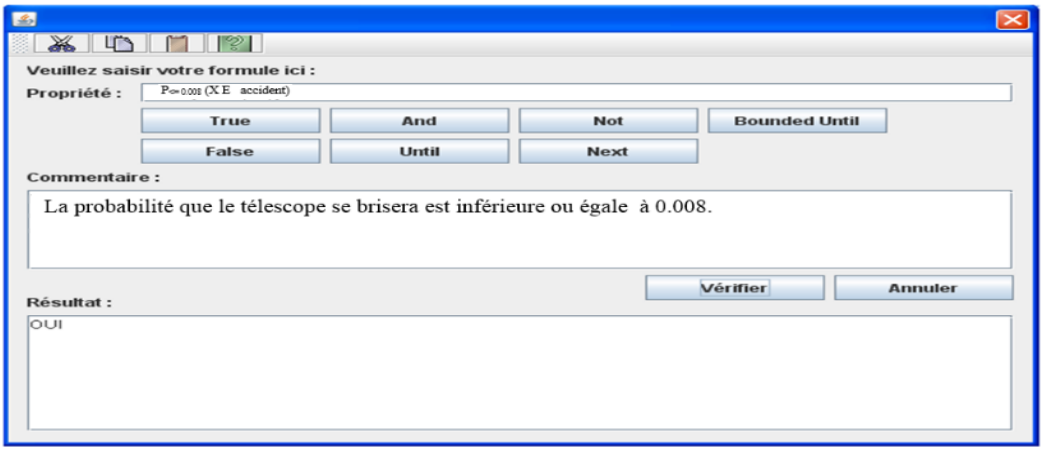


Figure 7. Vérification de la propriété $P_{<= 0.008}$ (X_e accident)

6. RESULTATS 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 [18]. 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. Les résultats obtenus sont meilleurs que ceux fournis par l'outil pState [17] : 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.

Ce travail peut être vu 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.

7. CONCLUSION ET PERSPECTIVES

Dans ce papier, 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. En plus de l'introduction du HMM comme nouveau formalisme de modélisation, nous avons conçu et réalisé un outil appelé HMM_Model-Checker. Il permet de vérifier des propriétés définies en logique POCTL, sur un modèle HMM avec une interface assez conviviale.

Pour élever le niveau d'abstraction de modèle et de la propriété, nous avons intégré 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. Nous avons expérimenté le travail sur une étude de cas d'un système embarqué réel qui est le HST et les résultats obtenus sont prometteurs.

A l'avenir, nous espérons le doter de nouvelles fonctionnalités, puis de l'étendre à d'autres propriétés afin de l'expérimenter sur de nouvelles applications réelles, ensuite l'intégrer dans PRISM avec d'autres outils APMC et YEMER pour lui permettre de supporter plus de modèles et de techniques de vérification approximative et statistique en vue d'obtenir un environnement de vérification probabiliste consistant, et enfin pour élever le niveau d'abstraction de modèle et de propriété, nous comptons intégrer à notre environnement l'outil pState utilisant les pCharts, une version étendue de machines à états hiérarchiques pour la description de modèle et la spécification de propriété en même temps.

REFERENCES

[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] 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.

[4] 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.

[5] Blondin M., 2016. Algorithmique et complexité des systèmes à compteurs. Thèse de doctorat en informatique, Université de Paris-Saclay, France, 236p.

[6] Forejt V., Kwiatkowska M., Norman G. & Parker D., 2011. Automated verification techniques for probabilistic systems,

Proceedings of the Formal Methods for Eternal Networked Software Systems (SFM'11), 53–113.

[7] Kwiatkowska M., Norman G. & Parker D., 2009. Prism: probabilistic model checking for performance and reliability analysis, *SIGMETRICS*, vol. 36, no. 4, 40–45.

[8] Bourdil P.A., 2015. Contribution à la modélisation et la vérification formelle par model-checking – Symétries pour les Réseaux de Petri Temporels. Thèse de doctorat en informatique, Université de Toulouse, France, 148p.

[9] Graja Z., 2015. Vérification formelle des systèmes multi-agents auto-adaptatifs. Thèse de doctorat en informatique, Université de Toulouse, France, 157p.

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

[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] Zhang L., Hermanns H. & and Jansen D. N., 2005. Logic and Model Checking for Hidden Markov Models. Proceeding of FORTE, *LNCS 3731*, 98 -112.

[13] Hansson H. & Jonsson B., 1994. A logic for reasoning about time and probability, *Formal Aspects of Computing*, Vol. 6(5), 512-535.

[14] 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.

[15] Tatarewicz J.N., 2001. The Hubble Space Telescope Servicing Mission. In: From engineering science to big science. (Eds) J. Henry – 365-396.

[16] Segala R., 1995. Modelling and Verification of Randomized Distributed Real Time Systems, Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA. 283p.

[17] 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.

[18] Nokovic B. & Sekerinski E., 2013. pState: A probabilistic statecharts translator", Proceedings of the Embedded Computing (MECO), 2nd Mediterranean Conference on Embedded Computing, 29-32.

[19] 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.