Un vérificateur symbolique efficace

An efficient symbolic model checker

Ahlem Ferdenache* & Rachid Boudour

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

Soumis le 01/09/2015 *Révisé le 17/05/2016* *Accepté le 14/06/2016*

ملخص

في هندسة الكمبيوتر الحالية، الهدف هو الوصول إلى أساليب وأدوات التي يمكن تشغيلها في التدقيق على نموذج تلقائيا: التدقيق الرمزي على نموذج. مشكلة هذا الأخير تكمن في نموذج الحالات الكبير جدا.أحد الحلول يتمثل في استخدام فحص النموذج الرمزي جنبا إلى جنب مع بنيات المعطيات المدمجة. الغرض من هذه الورقة هو تصميم وتنفيذ أداة جديدة وقوية للتحقق من الخصائص الهامة في الأنظمة المشحونة الحرجة على أساس مفهوم الحالة الرمزية وبنيات المعطيات دبم (الفرق مصفوفة ملزمة). يتم التعبير عن مواصفات النظام باستخدام الأوتوماتا الموقوتة والخاصيات باستخدام لغة المنطق في الوقت الحقيقي. لقد تم بعد مقارنة النتائج المتحصل عليها مع تلك الموجودة والمنشورة.

الكلمات المفتاحية: *فحص رمزي* -الأوتوماتا الموقوتة. *المنطق في الزمن الحقيقي – اني– د ب م*

Résumé

En génie informatique d'aujourd'hui, l'objectif est d'arriver à des méthodes et des outils qui peuvent exécuter la vérification de modèle automatiquement : le model checking. Le problème du model checking est le grand espace d'états. Une des solutions est d'utiliser le model checking symbolique combinée à des structures de données compactes. Le but de ce papier est de concevoir et mettre en application un nouvel outil performant, pour vérifier des propriétés importantes dans les systèmes critiques basés sur la notion d'état symbolique ainsi que des structures de données DBM (Difference Bound Matrices). Les spécifications sont exprimées à l'aide d'automates temporisés pour le système et de logique temps réel pour les propriétés. Les résultats obtenus sont comparés à ceux de la littérature.

Mots clés :*vérification symbolique - automates temporisés - TCTL–à la volée - DBM.*

Abstract

In computer systems, the goal is to achieve methods and tools that can check models automatically: model checking. The model checking problem is the large states space. One solution is to use the symbolic model checking combined with compact data structures. The main of this paper is to design and implement a powerful new tool to check important properties in critical systems based on the concept of symbolic state and DBM data structures (Difference Bound Matrices). The specifications are expressed using timed automata system and real-time logic for properties. The obtained results are compared with those in the literature.

Key words: *symbolic model checking - timed automata - TCTL–on the fly - DBM.*

* *Auteur correspondant : ahlarosenet@yahoo.fr*

1. INTRODUCTION

La complexité croissante des applications informatiques a malheureusement pour corollaire l'existence de comportements anormaux aux conséquences parfois irréparables. Un exemple douloureux, est l'échec du missile «Patriot», le 25 février 1991, à Dahran en Arabie Saoudite. Une batterie américaine de missiles Patriot ne put suivre et intercepter un missile «Scud» irakien qui frappa un baraquement de l'armée. Le calcul du temps depuis l'amorçage était faux. Une fois converti, il conduisit à une incertitude de 500 mètres environ. Le missile Scud s'est trouvé hors de portée. Par conséquent, une perte effroyable humaine évaluée à 28 morts et plus d'une centaine de blessés [1] a eu lieu. Une autre catastrophe célèbre, est l'explosion de Ariane V, à Kourou (Guyane) le 4 juin 1996, due principalement à la réutilisation d'une bibliothèque logicielle de manipulation de nombres, fonctionnelle pour Ariane IV. Cette bibliothèque était utilisée pour traiter les informations de vitesse instantanée du lanceur. La vitesse atteinte par le lanceur Ariane V était supérieure à celle d'Ariane IV, et n'était pas représentable dans le codage des nombres de la bibliothèque. Après une trentaine de secondes du lancement, la vitesse du lanceur a dépassé l'intervalle de représentation, le logiciel de bord a levé une exception qui a entrainé une suite d'ordres incohérents menant à l'explosion du lanceur. Une lourde perte a été évaluée à 500 millions d'euros sans compter le coût du développement qui se chiffre à 7milliards d'euros [2]. Une longue liste de missions spatiales avortées, des équipements informatiques et médicaux défaillants, dus pour la plupart d'entre eux à une erreur du logiciel ont été enregistrés [3]. Ainsi, ces systèmes peuvent présenter un aspect critique dans le sens où certaines erreurs génèrent des conséquences graves, leur vérification s'avère donc indispensable. La vérification se propose d'assurer le fonctionnement correct d'un système (Est-ce-que nous construisons le produit correctement ?). En effet, la solution adoptée consiste à explorer et à analyser systématiquement l'espace des états. Cet espace devient très grand dès que le nombre de composants augmente, rendant la décision de satisfaction de propriété du système étudié difficile à obtenir voire impossible. Pour contourner cette difficulté, nous choisissons entre autres d'agir sur l'algorithme d'exploration de l'espace des états et les structures de données comme techniques d'abstraction afin de rendre cette vérification possible.

Ce papier propose de valider cette solution en développant un noyau de vérification en Visual C++, basé sur la vérification symbolique et la logique TCTL (Timed Computation Tree Logic).

Le reste de l'article est structuré en sections comme suit : Après une brève introduction, suit la deuxième section qui brasse un survol sur la technologie de vérification. La troisième section montre une vue panoramique sur la morphologie de notre système. Quant aux quatrième et cinquième sections, elles décrivent la méthodologie adoptée dans tous ses états et le flux de conception avec des détails d'implémentation. La sixième section propose des études de cas, pour illustrer les potentialités du système et discussion de résultats sur le plan de la complexité du model checking, défi majeur pour les développeurs d'algorithmes. Une conclusion succincte avec des directions de ce travail en cours et des futures investigations terminent ce papier.

2. ETAT DE L'ART

Historiquement, le model checking pour les systèmes réactifs a été introduit dans les années 80s, principalement aux Etats-Unis [4]. Il consistait à vérifier des propriétés écrites en logique temporelle sur des systèmes d'états finis. Les inconvénients rencontrés se résumaient dans la terminaison, beaucoup de contrôle et peu de données. Mais dans la pratique, les systèmes sont plutôt à états infinis. L'introduction de la technique symbolique basée sur les BDD (Binary Decision Diagram) a donné un nouveau souffle à ce type de vérification [5]. Parallèlement, le model checking pour les systèmes temps réels a débuté dans les mêmes années 80s. L'introduction des automates temporisés, a donné à son tour un nouvel élan à cette dernière.

La figure 1 résume les techniques symboliques opérant des calculs sur des ensembles d'états.

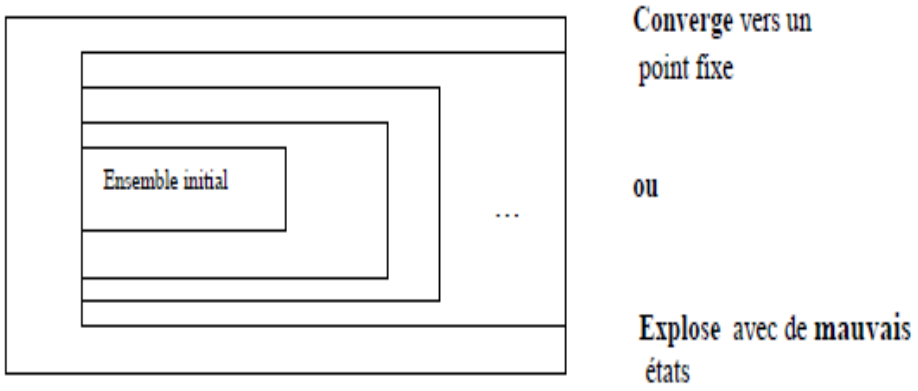


Figure 1 : Techniques symboliques opérant des calculs sur des ensembles d'états

Actuellement, on assiste à plusieurs extensions et améliorations, toutes basées sur la trilogie modèle-spécification-algorithme, prises séparément ou combinées. En réalité, des spécifications bonnes et complètes sont encore difficiles à établir : on vérifie seulement des propriétés simples. Le but n'est plus donc de prouver que le système est absolument correct mais d'aspirer à des outils qui peuvent assister le développeur à trouver des erreurs et à avoir plus de confiance dans sa conception.

Le model checking suscite de plus en plus de convoitises. A l'instar des autres approches de vérification, il continue à évoluer et peut leur être complémentaire. Son émergence due en partie à son utilisation précoce dans le cycle de développement, est limitée par deux aspects pratiques : le langage de spécification et la taille du système. Le model checking comme décrit dans les travaux des références [6] et [8] exige généralement qu'on décrive le système dans un langage spécifique. Cette description est nommée modèle ce qui nécessite donc un effort supplémentaire. Comme il arrive fréquemment que le langage de modélisation est moins riche que le langage de programmation, le modèle n'est souvent qu'une abstraction du système et se pose alors la question de ce qu'est convenu d'appeler : un fossé sémantique. Le model checking analyse exhaustivement tous les états de la structure de Kripke (voir section 3.1) et son espace d'états devient exponentiel dès que le nombre de composants augmente ce qui nécessite donc une mémoire très grande pour stocker tous les états. En d'autres termes, il ne permet de travailler aisément que sur des systèmes plus petits. Souvent, un système industriel est trop gros pour le model checking et on doit construire un modèle réduit, une abstraction. Pour contourner cette faiblesse, il y a une tendance alternative à utiliser le model checking symbolique. Nous rappelons qu'un état symbolique (ensemble des prédécesseurs de l'état courant par exemple) comme étant un couple (s,ψ), $s \in S$ est un état de l'automate temporisé et $\psi \in \psi(H)$ est une contrainte temporelle (S ensemble des états symboliques, H ensemble des horloges). L'état symbolique (s,ψ) représente l'ensemble des états $(s,v) \in Q$ (Q ensemble des états)tel que v satisfait ψ, plus formellement : $[(s, \psi)] = \{ (s,v) \in Q / v$ satisfait $\psi \}$.

En d'autres termes, au niveau de la représentation des données, on cherche à manipuler des ensembles d'états au lieu des états simples et à les représenter par des formules en logique propositionnelle. On fait la même chose pour les transitions. Les algorithmes utilisent deux méthodes : analyse en arrière, consistant à partir de l'état terminal et à l'aide des prédécesseurs à converger vers l'état initial ou un point fixe (Fig.1) ou bien l'analyse en avant à l'aide de successeurs consistant à atteindre l'état final, un problème de terminaison peut se poser [8]. En général, construire une abstraction qui soit de taille suffisante pour les outils et qui ne soit pas une simplification abusive de la réalité nécessite une expertise qu'on ne trouve que dans le milieu académique. Il faut néanmoins signaler un avantage, pas des moindres du model checking, est qu'il peut être accompli sur un modèle lors de la phase de conception bien en amont du développement lui-même, contrairement à d'autres approches. De nombreux outils existent (non) commerciaux, nous citons les plus connus (Tab. 1) qui peuvent être utilisés librement pour un usage non commercial [9-13].

Tableau 1 : Principaux outils de model checking

Spécification système	Spécification propriété	Algorithme	Outil
Automate de Büchi	PLTL	Check 0	SPIN
Structure de Kripke	CTL	Check that	SMV
Automate temporisé	TCTL	[s0]=iSat(0), check that	Uppaal
Automate temporisé	TCTL	Back,forward reachability	Kronos
Automates temporiséshybrides	TCTL	Back,forward reachability	HyTech

3. CONCEPTS DE BASE

3.1 Automates temporisés

L'automate temporisé a été étudié sous de multiples facettes, tant au niveau de la théorie des langages qu'au niveau des modèles temporisés pour la spécification et la vérification. En effet, des problèmes tels que le non déterminisme, la minimisation [14], le pouvoir d'expression des horloges et la caractérisation logique des langages temporisés [15], ont été étudiés. Ce modèle a été utilisé avec succès dans la spécification et la vérification de systèmes temporisés [16].

Une structure de Kripke est un modèle de calcul, proche d'un automate fini non-déterministe où des étiquettes sont associées aux places avec des logiques temporelles [8]. Formellement, une *structure de Kripke* est un quintuplet (S, R, I, L,λ) avec S désignant un ensemble d'états, I un ensemble d'états initiaux $I \subseteq S$, R un ensemble de transitions $R \subseteq S \times S$, L un ensemble d'étiquettes et λ une application de S dans π qui associe à chaque état un élément de π, π est un ensemble de propositions atomiques. *Un chemin* dans le modèle de Kripke est une séquence infinie $\sigma = s_0, s_1, s_2, \ldots \in S^*$ et $s_0 \in I$ et $(s_i\ s_{i+1}) \in R$. Un état s est *atteignable* dans M s'il y a un chemin partant d'un état initial à s.

Un système peut être composé de plusieurs automates [17].

3.2 Logique temps réel

Pour énoncer formellement des propriétés temps réel, plusieurs approches sont envisageables. Les propriétés simples peuvent être exprimées en termes d'atteignabilité, les propriétés plus compliquées peuvent faire appel à la technique des automates observateurs ou d'utiliser une logique temporisée. Le lecteur intéressé trouvera des détails dans la référence [7]. Nous nous intéressons à la troisième approche.

Les logiques temporelles antérieures à TCTL, ne permettent pas (ou pas facilement) d'exprimer des aspects quantitatifs sur le temps. Elles se limitent en fait à des aspects « avant/après ». Cette limitation est parfois gênante : par exemple, on souhaite souvent pouvoir écrire aisément le fait qu'il existe un chemin menant à une certitude avant un délai maximal. Les logiques temps réel permettent d'exprimer ce genre de propriétés pour un modèle de temps continu, sur des automates temporisés. Si on prend l'exemple d'un système quelconque avec une alarme, on veut exprimer que la réponse se produit dans moins de 5 unités de temps après l'alarme, on écrira $AG(\text{alarme} \Rightarrow AF_{\leq 5}$ réponse). De telles logiques sont mises en œuvre dans des outils tels que HyTech [11], Uppaal [12] et Kronos [13]. Une démarche naturelle consiste à étendre les opérateurs∪, F, …, de la logique temporelle avec des informations quantitatives sur l'écoulement du temps, comme proposé initialement dans ma référence [18]. Ainsi la formule $P \cup_{< 2} Q$ énonce que P est vraie jusqu'à ce que Q soit vraie, et que Q sera vraie en moins de deux unités de temps (depuis l'état courant). Appliquée à CTL, cette approche conduit à une logique temporisée du temps arborescent : TCTL (*TimedCTL)*. Un complément d'informations se trouve dans la référence [19].

Les propositions atomiques manipulées dans les formules de TCTL pourraient contenir des tests sur la valeur des horloges de l'automate. Par exemple, on pourrait envisager d'utiliser la formule $AG_{(h<5)}$ pour vérifier que l'horloge h du système a toujours une valeur inférieure à 5. La logique TCTL permet d'énoncer certaines propriétés temps réel, par exemple : $AG\ (Pb \Rightarrow AG_{(\leq 5)}$ alarme) : permet d'énoncer que si un problème arrive, l'alarme se déclenchera immédiatement et durera au plus 5 unités de temps. Les propriétés exprimables en TCTL sont nombreuses. Citons quelques-unes :

absence globale : $AG\ \neg\varphi$, absence postérieure : $AG\ (\varphi \rightarrow AG\ \neg\Psi)$, *déclenchement* : $AG\ (\varphi \rightarrow_{(<5)} AF\ \Psi)$, *existence inévitable* : $AF_{(>=2)}\varphi$, *existence récurrente* : AGAF φ, etc.

Citons deux approches pour vérifier les propriétés TCTL, interprétées sur un automate temporisé: approche basée sur la notion

d'automate de régions (des détails sont indiqués dans [13]) et approche symbolique. Elle sera abordée plus loin.

3.3 Algorithme de décision

L'un des critères de développement pour le model checking est la décidabilité, c'est-à-dire qu'il soit possible de développer des algorithmes qui calculent si le modèle du système vérifie ou non la spécification de la propriété. Il existe des variétés d'algorithmes de model checking selon le formalisme utilisé pour modéliser le système ainsi que le type de propriété considérée et son langage de spécification. 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 de model checking, mais elle ne lui est pas équivalente. De plus, le développement des algorithmes va de pair avec la position de structures de données en vue de leur implémentation. Ces structures doivent être relativement compactes (pour limiter les problèmes de mémoire) et doivent permettre de réaliser efficacement toutes les opérations apparaissant dans l'algorithme

Après avoir défini les automates temporisés et la logique TCTL, on cherche à obtenir un algorithme de model checking pour décider automatiquement si un automate temporisé vérifie-t-il une formule ? Dans ce cadre, la difficulté évidente est qu'un automate temporisé possède un nombre infini de configurations puisqu'il existe un nombre infini de valuations d'horloges possibles. Cette infinitude est due à deux raisons fondamentales : Les valeurs des horloges ne sont pas bornées, et même si elles sont restreintes à un intervalle borné, l'ensemble des réels est dense (R+ est le plus utilisé dans la littérature).

4. FLOT DE CONCEPTION

La dernière phase de ce processus de développement est l'implémentation en vue de la réalisation d'un outil utilisable. Bien sûr, même si cet outil apparaît alors comme une boite noire où il suffit d'appuyer sur un bouton pour avoir une réponse, une bonne connaissance des bases théoriques de l'outil sont nécessaires pour utiliser au mieux toutes ses possibilités.

4.1 Approche symbolique

Soit en entrée, un modèle du système à vérifier sous forme d'un système de transitions d'états étiquetés (structure de Kripke), et une propriété spécifiée par une formule en TCTL dans notre cas, le model checker doit fournir une réponse (négative/positive) à la question : Est-ce-que le modèle vérifie la propriété considérée ? Lorsqu'il détecte qu'un modèle ne satisfait pas une propriété, il exhibe un comportement du modèle qui viole la propriété, appelé contre-exemple. La figure 2 montre le principe du modèle chacking symbolique [17].

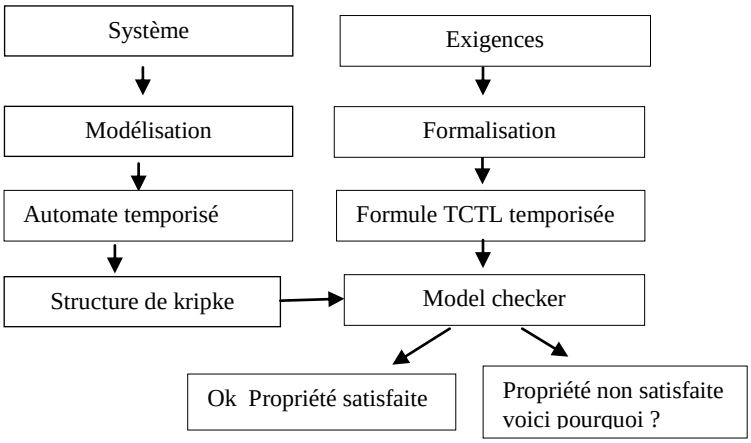


Figure 2 : Principe du Model checking symbolique [17]

Cette approche utilise un algorithme de model checking symbolique qui calcule l'ensemble caractéristique d'une formule TCTL c'est-à-dire l'ensemble des configurations i vérifiées par cette formule. Le principal problème de l'approche symbolique concerne la procédure

de décision. En effet, dans ce cas, décider si un ensemble d'états est inclus dans un autre, revient à décider si un prédicat implique un autre. Afin de résoudre ce problème, nous proposons une représentation des ensembles caractéristiques des formules qui est à la base d'une procédure de décision mise en œuvre de façon efficace. L'algorithme de model checking symbolique (Fig. 2) est réduit à quatre étapes :

1) Représenter les prédicats d'états : un prédicat d'état est une conjonction ou une disjonction de propositions atomiques et contraintes temporelles, par exemple : $\alpha \wedge x \leq 25$; où : a est une proposition atomique et x est une horloge
2) Représenter les contraintes temporelles sous forme matricielle : il s'agit d'une forme permettant de représenter des contraintes temporelles. Illustrons-la à l'aide d'un exemple. Soit à considérer la contrainte :

$1 \leq x \leq 4 \wedge z \leq 5 \wedge z - x \leq 0 \wedge x - z \leq 2$

Elle peut être représentée entre autres, au moyen de la matrice M, matrice aux différences bornées ou DBM (Differences Bounded matrix) suivante :

$$M = \begin{pmatrix} & 0 & x & z \\ 0 & (0,\leq) & (-1,\leq) & (0,\leq) \\ x & (4,\leq) & (0,\leq) & (2,\leq) \\ z & (5,\leq) & (0,\leq) & (0,\leq) \end{pmatrix}$$

Chaque entrée de la matrice représente la borne supérieure de la différence entre deux horloges. Ainsi, l'entrée M xz est égale à (2, ≤) représente la contrainte x – z ≤ 2. Un avantage, pas des moindres des DBM, est leur aptitude de normalisation. En effet, il est possible d'obtenir une matrice unique regroupant les contraintes les plus fortes entre toutes les horloges en appliquant l'algorithme de Floyd pour obtenir tous les chemins les plus courts entre les sommets d'un graphe.

3) Calcul de l'opérateur⊳

Soit $A = (S, H, E, S_I, \delta, P)$,un automate temporisé etQ_1,Q_2 deux caractéristiques de deux prédicats d'état. Le calcul de l'opération Q1⊳ $Q2$ consiste à déterminer :

- Tous les *prédécesseurs instantanés* des états de Q_2,c.-à-d. tous les états de Q_2 plus tous ceux qui peuvent atteindre un état de Q_2 par une transition discrète.
- Tous les *prédécesseurs temporels* de ces derniers, c.-à-d., tous les états qui peuvent atteindre un prédécesseur instantané d'un état de Q_2, par une transition temporelle, telle que tous les états intermédiaires sont dans Q_1 ou Q_2.

4) Evaluer symboliquement des formules TCTL.

Etant donné une formule φ en TCTL et un automate temporisé, l'algorithme symbolique consiste à calculer pour φ l'ensemble $S(\varphi)$ d'états symboliques qui représentent l'ensemble caractéristique de φ (c.-à-d. l'ensemble des états où cette propriété est vérifiée).

4.2Illustration de l'algorithme

Pour comprendre le fonctionnement de l'algorithme, déroulons-le sur un exemple. Considérons le cas du contrôleur de température décrit dans la sous-section 5.1.2 et une propriété à vérifier exprimée en TCTL :

$\varphi = r \rightarrow \neg E \neg b \cup (r \wedge z > 35)$

Cette propriété signifie qu'il n'est pas possible d'atteindre un état qui satisfait r en un temps supérieur à 35 à partir d'un état satisfaisant r sans passer par un état satisfaisant b, (voir illustrations en Fig. 4)

Déroulement de l'algorithme :

On a : $X_0 = Q_2$

$X_{j+1} = X_j \vee Q_1 \rhd X_j$

Avec :

$Q_1 = S(\neg b) = (0,T) \cup (1,x = 0) \cup (3,x \leq 35 \wedge y \leq 20) \cup (4,y \leq 35) \cup (5,y=0) \cup (6,T)$

$Q_2 = S(r \wedge 35 < z) = (1, x = 0 \wedge 35 < z) \cup (5, y = 0 \wedge 35 < z)$

Dans la première itération on obtient :

$X_1 = X_0 \vee Q_1 \rhd X_0$

$= (0, true) \cup (1, x = 0 \wedge 35 < z) \cup (3, y < 20 \wedge y - z < -15) \cup (4, x < 35 \wedge x < z) \cup (5, y = 0 \wedge 35 < z)$

Dans la deuxième itération on obtient :

$X_2 = X_1 \vee Q_1 \rhd X_1$

$= (0, true) \cup (1,x = 0 \wedge 35< z) \cup (3, (y = 20 \wedge x - y < 15 \wedge x < z) \vee (y \leq 20 \wedge x < z \wedge z - y \leq 15) \vee (y < 20 \wedge y - z < -15)) \cup (4, x < 35 \wedge x < z) \cup (5, y = 0 \wedge 35 < z)$

Dans la troisième itération on obtient :

$X_3 = X_2 \vee Q_1 \rhd X_2$

$= (0, true) \cup (1,x = 0 \wedge 35< z) \cup (3, (y = 20 \wedge x - y < 15 \wedge x < z) \vee$

$(y \leq 20 \wedge x < z \wedge z - y \leq 15) \vee (y < 20 \wedge y - z < -15)) \cup$

$(4, x < 35 \wedge x < z) \cup (5, y = 0 \wedge 35 < z)$

On trouve alors $X_3 = X_2$. On a donc :

$S(\neg E \neg b U (r \wedge z > 35)) = X_2 = (0, T) \cup (1, x = 0 \wedge 35 < 0) \cup (3, (y = 20 \wedge x - y < 15 \wedge x < 0) \vee (y \leq 0 \wedge x < 0 \wedge 0 - y \leq 15) \vee (y < 20 \wedge y - 0 < -15)) \cup (4, x < 35 \wedge x < 0) \cup (5, y = 0 \wedge 35 < 0)$

$= (0, true)$

Donc, $S(r) \subseteq \neg (0, true)$. Par conséquent le système satisfait la propriété.

Remarque. Cet algorithme constitue le noyau de notre vérificateur et est équivalent à celui mis en œuvre dans l'outil *KRONOS* [13].

5. EXPERIMENTATIONS

Pour vérifier notre outil, nous avons choisi deux exemples de systèmes temps réels et un troisième exemple abstrait. Les exemples considérés, sont voulus simples pour l'intérêt de la présentation.

5.1 Exemples

5.1.1 Souris à un seul bouton

Cet exemple consiste à spécifier un système qui reconnaît les clics simples et les clics doubles d'une souris avec un seul bouton [20]. Cette application est intéressante car le comportement du processus dépend exclusivement des temps relatifs entre les événements. Il y a deux types d'évènements : le premier consiste à presser le bouton de la souris et le second à le relâcher. Presser le bouton est signalé par l'action p et le relâcher est désigné par l'action r. Un clic simple est défini lorsque le temps écoulé entre presser et relâcher le bouton, est inférieur à une certaine constante $t_c = 3$ unités de temps par exemple. Un clic double est défini par deux clics simples séparés d'au plus $t_{cd} = 2$ unités de temps.

5.1.2 Contrôleur de température

Nous décrivons ici le contrôleur de température d'un réacteur [21]. La température du réacteur est obtenue périodiquement par un capteur. La tâche du contrôleur consiste à réfrigérer le réacteur lorsqu'il reçoit le signal r du capteur. L'opération de réfrigération est mise en œuvre au moyen de deux barres qui doivent être utilisées dans l'ordre. Lorsque le contrôleur reçoit le signal r du capteur, il commence l'opération de mouvement de la barre 1, ce qui prend un certain temps t_1. Ensuite, il répète l'opération avec la barre 2, ce qui prend t_2 unités de temps. L'arrivée d'un signal r pendant la temps de réfrigération est considéré comme une erreur. De plus, le capteur n'est pas fiable à cent pour cent, c'est-à-dire qu'il peut tomber en panne et risque de ne pas envoyer de signal r au contrôleur. Dans ce cas, et pour des raisons de sécurité, si le temps écoulé depuis le dernier signal r reçu est supérieur à $t_{max} = 35$ unités de temps par pure supposition, le contrôleur commence une nouvelle opération de réfrigération.

5.1.3 Exemple abstrait

C'est un exemple d'un automate temporisé modélisant un système quelconque composé de 3 états de contrôle,de 4 transitions entre ces états et utilise 3 horloges [22]. Cet exemple est considéré à des fins de comparaison.

5.2 Propriétés TCTL

En pratique, les propriétés de sûreté sont, en général, les plus importantes pour la correction du système. Elles méritent que leur soient consacrés des efforts accrus en termes de temps, de priorité, de rigueur, etc. Il se trouve qu'elles sont souvent plus faciles à vérifier. La propriété de vivacité, à l'opposé de la *vivacité bornée,* garantit seulement que quelque chose aura lieu sans donner d'information sur les délais. C'est sous ces considérations que nous avons optées pour les propriétés de *bonne temporisation*, de *vivacité bornée* et de *sûreté.*

5.2.1Propriété de bonne temporisation

Elle est énoncée comme suit : $\varphi = (EFw = \text{constante} > 0)$, expérimentée sur l'exemple 1 (w est une horloge). Cette propriété sert à vérifier que le système est bien modélisé, c'est-à-dire qu'on ne peut pas atteindre un état de blocage où le système ne peut plus progresser. Ce blocage peut être causé par des contraintes d'horloges si elles ne sont pas bien exprimées. Par exemple, soit un invariant d'état 0 : $x \leq 5$ et une condition de transition de l'arc [0,1] : $x < 3$. Dans l'état 0, le système peut séjourner jusqu'à 5 unités de temps. Si on suppose que le système reste dans l'état 0 pendant 4 unités de temps, il ne peut plus franchir l'arc car la condition de transition n'est plus vérifiée et par conséquent le système est bloqué. La propriété de bonne temporisation se trouve ainsi non vérifiée et il faut alors engager une correction (voir Fig. 3).

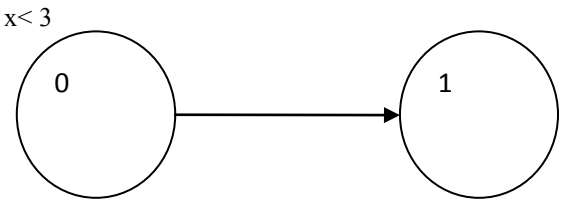


Figure 3 : Illustration de la propriété de bonne temporisation

5.2.2 Propriété de vivacité bornée

Pour la souris, nous souhaitons vérifier la propriété suivante : si le bouton a été pressé et relâché une fois alors inévitablement avant un temps égal à tcs+ tcd, le système détecte un clic simple ou un clic double. Cette propriété est exprimée en TCTL par la formule :

$\varphi = (p1 \wedge r1 \wedge x = 0) \rightarrow A\, F_{< 5}\, (cs \vee cd)$.

5.2.3 Propriété de sûreté

La spécification du contrôleur est établie pour des raisons de sécurité. Si le contrôleur ne reçoit pas un nouvel ordre de réfrigération avant un temps tmax depuis le dernier ordre reçu, alors il doit aussi réfrigérer. Cette propriété s'exprime par la formule suivante :

$\varphi = r \rightarrow \neg\, E\, \neg\, b\, U\, (r \wedge z > 35)$.

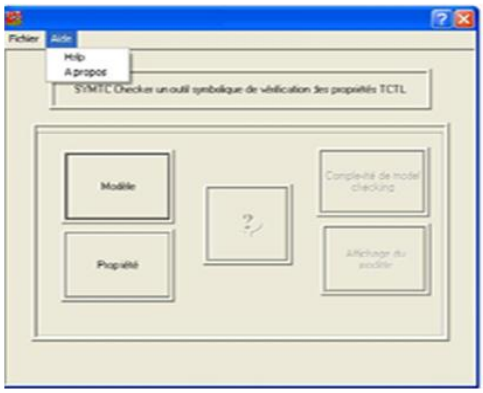
Cela veut dire qu'il n'est pas possible d'atteindre un état qui satisfait r en un temps supérieur à 35 unités de temps à partir d'un état qui satisfait r sans passer par un état qui satisfait b.

5.3 A propos de l'outil

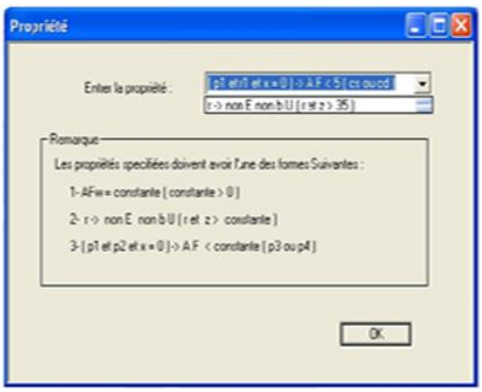
Pour avoir une meilleure idée sur notre outil, nous présentons quelques écrans d'utilisation dont le principal est décrit à la figure 4.1 et qui comporte :

- Un bouton Modèle qui permet d'entrer le modèle à vérifier.
- Un bouton Propriété qui permet d'entrer la propriété à vérifier.
- Trois boutons désactivés et qui seront activés ultérieurement.
- Un sous menu Help pour une bonne utilisation de l'outil.

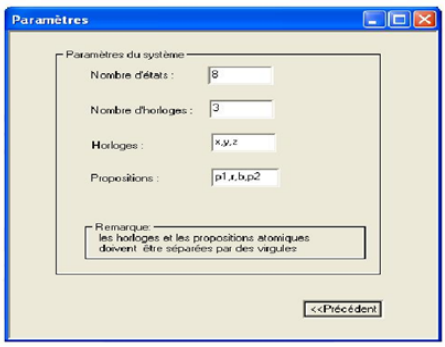
L'utilisateur peut saisir une propriété tctl ou choisir dans la liste déroulante de la figure 4.2. Cette boite de dialogue nous permet de spécifier la taille du modèle ainsi que les horloges et les propositions atomiques utilisées dans le système (Fig.4.3). A la Fig. 4.4, nous pouvons spécifier les propriétés atomiques vérifiées pour chaque état. Dans le cas où l'invariant est exprimé par plus d'une contrainte d'horloge, l'utilisateur doit taper et/ou dans le champ de saisie et/ou/rien pour spécifier les contraintes sur les autres horloges, comme le montre la Fig.4.5. La Fig.4.6, à son tour permet d'introduire les paramètres de chaque transition du modèle. Après toutes ces étapes, le bouton Affichage du modèle de la fenêtre principale devient actif pour permettre de visualiser le modèle entré, ainsi que le bouton '? '



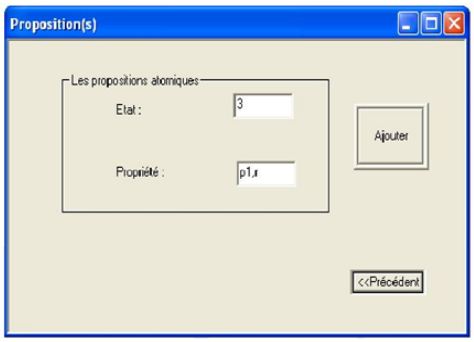
4.1



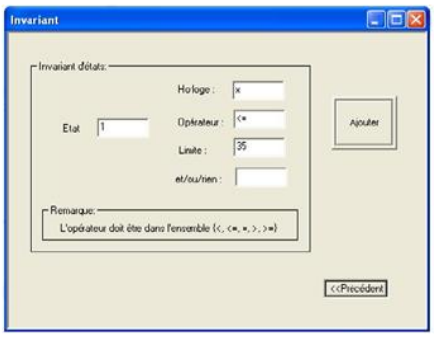
4.2



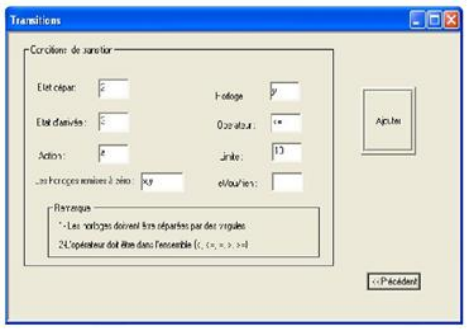
4.3



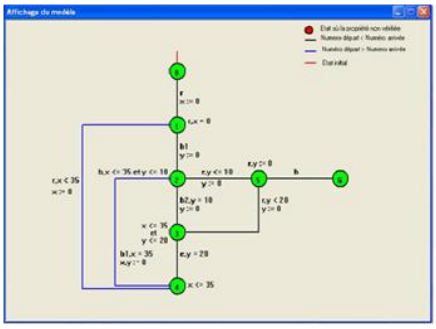
4.4



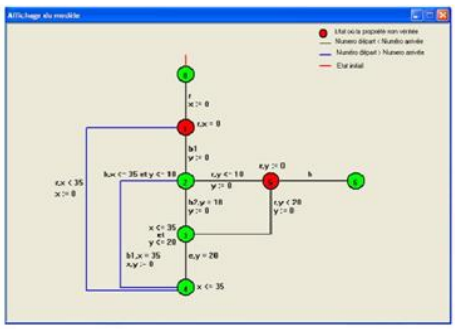
4.5



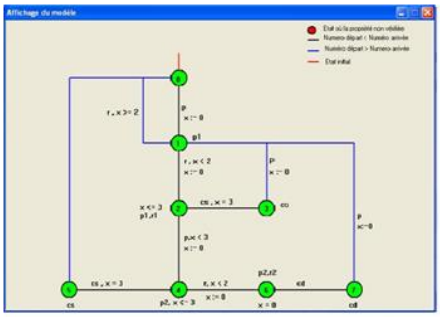
4.6



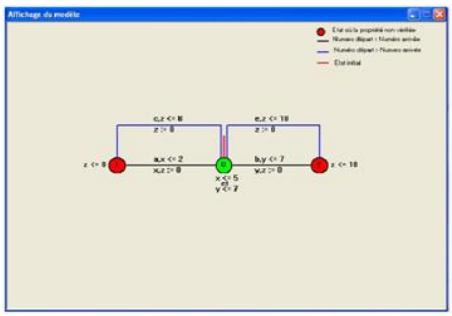
4.7



4.8



4.9



4.10

Figure 4 **:** (4.1) Fenêtre principale, (4.2) fenêtre propriété, (4.3) fenêtre paramètres,(4.4) fenêtre des propositions atomiques, (4.5) fenêtre des invariants d'état, (4.6) fenêtre des transitions du modèle, (4.7) La visualisation du modèle après la vérification , (4.8) Affichage du modèle pour la souris, (4.9) Affichage du modèle pour 'propriété vérifiée, (4.10) Affichage du modèle pour 'propriété non vérifiée'

6. RESULTATS ET DISCUSSION

Une fois que le modèle et la propriété tctl sont entrés, nous pouvons cliquer sur le bouton '?' pour visualiser le résultat de la vérification.

1. Pour le troisième exemple, nous avons vérifié une propriété de bonne temporisation et nous avons obtenu les résultats suivants : la visualisation du modèle après la vérification est représentée sur la figure 4.10.
2. Pour la souris, le résultat de la vérification de la propriété de vivacité bornée est illustré sur la figure 4.9.
3. Pour le contrôleur de température, l'outil retourne un résultat positif pour la propriété de sûreté et un résultat négatif pour la propriété de vivacité bornée (se reporter aux figures 4.7 et 4.8).

Les résultats obtenus sont synthétisés dans le tableau 2 et sur la figure 5. D'après la figure 5, où est illustrée une complexité en fonction du nombre de nœuds visités et espace occupé en octets pour chacune des propriétés Pi (i=1,2,4) dans l'ordre du tableau 2, on peut déduire aussi que cette complexité du model checking est une fonction des paramètres tels que le nombre d'états du modèle, le nombre d'horloges et le type de la propriété à vérifier [23]. L'outil est expérimenté sur trois exemples de caractéristiques différentes (en termes de nombres d'états, nombre d'horloges), pour le modèle de la souris. Le résultat obtenu pour la propriété de vivacité bornée est le même résultat obtenu dans les références [20] et [24]. Pour le modèle du contrôleur de température le résultat positif, retourné par notre outil pour la propriété de sûreté est identique à celui obtenu dans [24]. Les autres résultats obtenus sont spécifiques à notre outil et peuvent servir de base de comparaison à d'autres travaux.

Notre outil permet de vérifier des propriétés tctl interprétées sur des modèles de systèmes temps réels. Dans le cas où la propriété est vérifiée, il retourne aussi des statistiques sur la complexité du model checking. Sinon, il retourne le schéma du modèle avec l'indication des états causant la violation de la propriété en vue d'une correction [25]. Le tableau 2 résume un récapitulatif des résultats retournés par l'outil en question.

iTableau 2 : Récapitulatif des résultats retournés par notre outil

Exemple	Propriété	Résultat
Souris	EFy=5	Satisfaite
	$(p1 \wedge r1 \wedge x = 0) \rightarrow A\ F_{<5}\ (cs \vee cd$	Satisfaite
	$p1 \rightarrow \neg\ E\ \neg\ cs\ U\ (r1 \wedge y > 5)$	Non satisfaite
Contrôleur température	EFz=10	Satisfaite
	$r \rightarrow \neg\ E\ \neg\ b\ U\ (r \wedge z > 35)$	Satisfaite
	$(r \wedge b \wedge x = 0) \rightarrow A\ F_{< 35}(r \vee b)$	Non satisfaite
Exemple standard	EFw = 1	Non satisfaite

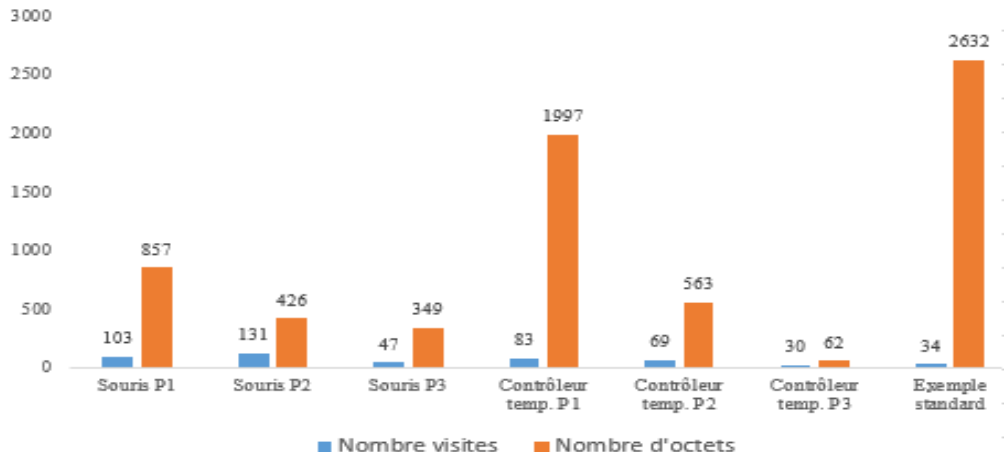


Figure 5 : Mesure de complexité pour les propriétés

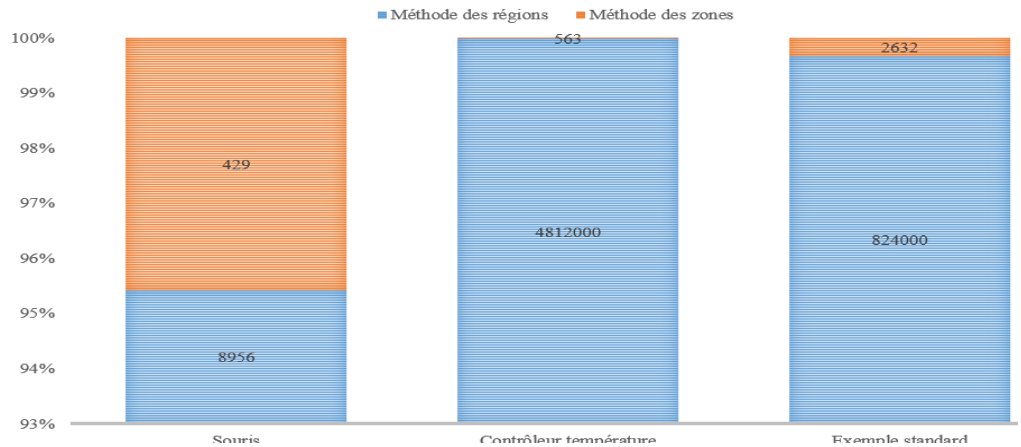


Figure 6 : Comparaison entre les méthodes *régions* et *symbolique (espace enoctets)*

La figure 6 établit le parallèle entre les approches par régions et symbolique et nous constatons que le gain en espace mémoire est très important. Il va de 20 à 8547 fois moins, proportions montrées en pourcentages sur la figure 6. Les résultats sont comparés à ceux obtenus par Kronos.

Notre approche a permis de réduire l'espace mémoire requis ainsi que le temps d'exécution. Cette réduction importante est faite grâce à une implémentation spécifique de la procédure de décision. Elle consiste tout d'abord en un parcours de tous les états du modèle en sauvegardant les invariants des états dans une matrice, puis l'algorithme est exécuté seulement une fois qu'on accède à la matrice pour récupérer les données (on ne fait pas à chaque fois un parcours qui va augmenter le temps d'exécution). Pour l'espace mémoire, on a utilisé la notion d'ensemble caractéristique qui consiste à réduire l'espace nécessaire à l'exécution. Cet ensemble ou zone est représenté par une DBM. Les DBM comme mentionnées précédemment, sont des structures de données efficaces mais présentant des difficultés de manipulation d'unions de zones, nécessaires dans tous les algorithmes de model checking. L'outil, est semblable à kronos. IL implémente un algorithme de model checking pour une logique temporelle temporisée. En particulier, il n'est pas limité, comme UPPAAL et HYTEC, à des propriétés d'atteignabilité et permet la vérification de propriétés telles que la sûreté, la vivacité et la bonne temporisation. En plus, par rapport à Kronos, il dispose d'une interface graphique de modèles temporisés et d'un module de simulation. Il est d'utilisation aisée et peut cibler un public assez vaste. Ses structures de données sont compactes et dynamiques contribuant impérieusement à la réduction significative de la complexité.

7. CONCLUSION

Notre outil de vérification, peut être perçu comme une aide à la vérification partielle, c'est-à-dire que des propriétés peuvent être vérifiées individuellement.

Notre contribution réduit la complexité. Elle consiste à utiliser une structure dynamique pour le calcul de l'opérateur $\triangleright$, où le modèle est décrit par une matrice de pointeurs dans laquelle chaque pointeur référence un enregistrement et toutes les valuations (valeurs prises par l'horloge) au sein d'un état donné sont décrites par une contrainte d'horloge constituant une borne pour toutes les valuations possibles. Ceci a permis de réduire le nombre d'états symboliques calculés par l'algorithme et par conséquent la complexité en espace et en temps d'exécution. Dans l'état actuel des choses, l'outil peut être assimilé à un stimulateur de confiance dans une conception.

Notre travail en cours, porte sur l'amélioration de l'algorithme pour générer un contre-exemple minimal en temps polynomial et en espace linéaire (pour la logique TCTL) en vue de faciliter son analyse pour différents traitements utiles: une correction sur le modèle ou sur la propriété même voire une génération automatique de cas de tests [26-27]. Ce travail devrait aboutir à la création d'un prototype en vue de son intégration dans une plateforme de vérification. Comme travail futur complémentaire, nous nous intéressons à l'introduction du besoin de définir des procédures structurées pour l'extraction automatique des propriétés des systèmes. Ce qui permet d'accélérer le processus de vérification, d'augmenter la productivité et

d'avoir un niveau de confiance plus élevé vis-à-vis des propriétés des systèmes temps réels.

REFERENCES

[1] Spencer C. T., 2010. The Encyclopedia of Middle East Wars: The United States in the Persian Gulf, Afghanistan, and Iraq Conflicts [5 volumes]: The United States in the Persian Gulf, Afghanistan, and Iraq Conflicts, 1887p.

[2] Lions, J. L., 1996. ARIANE 5 Échec du vol Ariane 501 Rapport de la Commission d'enquête.

[3] Bois, S., 2007. Les 10 plus grandes catastrophes liées à l'informatique, Colin Barker de ZDNet.

[4] Pnueli A.,1997. The temporal logic of programs, 18th IEEE symposium on Foundations of Computer Science FOCS77, 46-57.

[5] Bryant Randal E., 2003. Graph based algorithms for boolean function manipulation , *IEEE Transactions on computers*, Vol 35(8), 677-691.

[6] Abadi M. Abadir, M.; Albin, K.; Havlicek, J.; Krishnamurthy, N. & Martin, A., 2003. Formal Verification Successes at Motorola, *Formal Methods in System Design*, Vol. 22(2), 117-123.

[7] Berard B., Bidoit B., Finkel A., Laroussinie L., Petit A., Petrucci L., Schnoebelen P. & Mckenzie P., 2001. Systems and Software Verification, Model-Checking Techniques and Tools, Berlin-Heidelberg: Springer Verlag.

[8] Clarke E., Edmund M., Orna G. & Doron A. P., 1999. Model Checking, Cambridge, The MIT Press, Cambridge.

[9] http://www.cs.cmu.edu/_modelcheck/smv.html

[10] http://spinroot.com/spin/old.html

[11] Encrenaz-Tiphene E., 2014. Méthodes formelles pour la vérification des systèmes embarqués. *Techniques de l'ingénieur.*

[12] Larsen K.G., Pettersson P. & Yi W., 1997. UPPAAL in nutshell, *Journal of Software Tools for Technology Transfer*, Vol. 1(1-2), 134-152.

[13]Yovine S., 1997. Kronos: A verification tool for real-time systems, *Journal of Software Tools for Technology Transfer*, Vol. 1(1-2), 123-133.

[14] Alur R. & .Henzinger T.A. , 1992. Logic and models of real time: A survey. In Real-Time: Theory in Practice, *Computer Science,* Vol. 600, 74-106.

[15] Rolf D., 2004. Advanced Formal Verification. Kluwer Academic Publishers. eBook ISBN: 1-4020-2530-0 ISBN, 277p.

[16] Henzinger T. A., Nicollin X., Sifakis J. & Yovine S., 1994. Symbolic model checking for real-time systems, *Information and Computation*, Vol. 111, 193-244.

[17] Baier, C. & Katoen, J.P., 2008. Principles of Model Checking. Cambridge, Mass. :MIT Press.

[18] Koymans R., 1990. Specifying real-time properties with metric, *Real -Time Systems*, Vol. 2, 255-299.

[19] Laroussinie F., Schnoebelen P. & Turuani M., 2000. On the Expressive and Complexity of Quantitative Branching-Time Temporal Logics, *Computer Science*, Vol. 1776, 437- 446.

[20] Shaw A., 1992. Communicating real-time state machines, *IEEE Transactions on Software Engineering*, Vol. 18(9), 805-816.

[21] Jaffe M., Leveson N., Heimdahl M. & Melhart B., 1991. Software requirements analysis for real-time process-control systems, *IEEE Transactions on Software Engineering*, Vol. 17, 241-258.

[22] Manuel Núñez Matthias Güdemann (Eds.), 2015. Formal Methods for Industrial Critical Systems. Proceeding of 20th international xorkchop, FMICS 2015, LICS 9128, Oslo Norvège.

[23] Christos H. Papadimitriou, 1994. Computational complexiry, Addison Wesley ISBN 0-201-53082-1, 540p.

[24] Yovine S., 1998. Model-Checking Timed Automata. In School on Embedded Systems, *Computer Science,* Vol. 1494.

[25] Yovine S., 1992. Méthodes et outils pour la vérification symbolique de systèmes temporisés.

[26] Bochot Th, Virelizier P., Waeselynck H. & Wiels V. Paths to property violation: a structural approach for analyzing counter-examples, HASE 2010.

[27] Morb´e G., Miller C., Scholl C., & Becker B., 2014. Combined Bounded and Symbolic Model Checking for Incomplete Timed Systems, E. Yahav (Ed.): HVC 2014, LNCS88.