TSMV : model checking symbolique de systèmes simplement temporisés

Nicolas MARKEY

nmarkey@ulb.ac.be

Département d'Informatique
Université Libre de Bruxelles
Bd du Triomphe – CP 212
1050 BRUXELLES – BELGIQUE

Résumé

TSMV [5] est une extension temporisée du model checker symbolique NUSMV [4]. Cette extension permet de vérifier des propriétés quantitatives (exprimées dans la logique RTCTL) sur des structures de Kripke dont chaque transition est étiquetée par un intervalle indiquant les durées (entières) possibles pour cette transition. Nous présentons ici cet outil sur deux exemples : le premier traite un problème simple, et met en avant l'efficacité de TSMV par rapport à d'autre outils. Le deuxième exemple est plus complexe, et nous permet d'illustrer l'utilisation de TSMV pour des problèmes de *condition counting*, *i.e.* de décompte du nombre de fois qu'une propriété est satisfaite au cours d'une exécution.

1 Présentation de TSMV[1]

Le *model checking symbolique* est une technique de vérification formelle, introduite par McMillan [6], et permettant de vérifier efficacement que des systèmes de grande taille possèdent certaines propriétés [1]. Plusieurs implémentations (*e.g.* NuSMV[2] et Verus[3] [2]) permettent en plus de vérifier des propriétés quantitatives, en prenant pour notion de temps le nombre de transitions franchies. TSMV étend cette possibilité au modèle des *structures de Kripke avec durées* : chaque transition de la structure de Kripke est étiquetée par un ensemble d'entiers représentant les durées possibles que prend cette transition (la durée pouvant être nulle).

Tout comme NuSMV, TSMV permet de vérifier des propriétés de RTCTL (*i.e.* de la logique temporelle du temps arborescent (CTL) dans laquelle la modalité *Until* peut être indicée par une contrainte temporelle de la forme $[a..b]$, ou $\leq a$, $= a$ ou $\geq a$) avec condition d'équité. Lorsque c'est possible, TSMV fournit un contre-exemple si la propriété n'est pas vérifiée. Il permet aussi de calculer les durées minimale et maximale qui séparent deux ensembles d'états.

2 Application à deux exemples

2.1 Vérification du Bus PCI [3]

Description. Le bus PCI (utilisé dans la plupart des ordinateurs actuels) est un bus de communication entre divers composants d'un ordinateur. Ce bus a été modélisé et étudié par Campos, avec l'outil Verus. Un modèle pour NuSMV a également été construit. Le système est relativement complexe : plusieurs (quatre, dans l'exemple étudié ici) composants informatiques (un pont vers un bus ISA, un contrôleur SCSI, un contrôleur vidéo et un processeur) sont reliés à un bus de communication. Un *arbitre* reçoit les requêtes envoyées par les différents composants pour obtenir l'accès au bus, et donne l'accès à ces composants suivant une certaine politique (*round robin* ou *priorité fixe* ici). Lorsqu'un composant à obtenu l'accès au bus, il attend que celui-ci soit libéré par le composant qui l'utilise, et il peut alors faire sa transaction (pendant au plus 15 cycles d'horloge). Le modèle NuSMV peut être téléchargé sur `http://nusmv.irst.itc.it/examples/smv-dist/pci4p.smv`.

Campos *et al.* vérifient, sur ce modèle, plusieurs propriétés temporelles, calculant le nombre de « cycles d'horloges » du processeur entre, par exemple, une demande d'accès au bus par un composant et son obtention. Ils montrent ainsi que, si l'arbitrage se fait par une politique *round-robin*, le délai d'accès au bus est compris entre 2 et 118 cycles pour le bus ISA et le contrôleur SCSI, et entre 2 et 56 cycles pour le contrôleur vidéo et le processeur (la différence est liée au fait que, dans le modèle choisi, le pont ISA et le contrôleur SCSI subissent deux phases d'arbitrage, voir [3, figure 7]). Si l'arbitrage se fait avec priorités fixes, alors seul le composant de priorité maximale est sûr d'obtenir l'accès au bus. Un algorithme spécial de *condition counting*, disponible dans l'outil Verus, est également utilisé pour calculer le nombre maximal de transactions pouvant être effectuées sur le bus entre le moment où un composant demande l'accès au bus et le moment où il l'obtient : 5 transactions lorsque le composant est le bus ISA ou le contrôleur SCSI, et 2 si c'est le contrôleur vidéo ou le processeur. La différence s'explique de la même façon que précédemment.

[1] `http://www.lsv.ens-cachan.fr/~markey/TSMV/`
[2] `http://nusmv.irst.itc.it/`
[3] `http://www-2.cs.cmu.edu/~modelcheck/verus.html`

Étude avec TSMV. Nous considérons le modèle du bus PCI pour NuSMV, en y ajoutant simplement la condition `duration = 1` pour tout le modèle :

```
VAR
  duration: 0..1;
TRANS
  next(duration) = 1;
```

Nous retrouvons alors les mesures effectuées par Campos *et al.* De plus, sans algorithme spécialisé, TSMV permet de faire du *condition counting* : il suffit en effet de mettre la variable `duration` à zéro pour toutes les transitions, sauf pour les transitions arrivant dans un état vérifiant une certaine condition. Nous retrouvons ainsi les résultats de Campos : il y a au plus 5 transactions entre une demande et une autorisation d'accès au bus pour le pont ISA et pour le contrôleur SCSI, et 2 pour le contrôleur vidéo et le processeur.

Cette technique peut être mise en œuvre pour, par exemple, mesurer la quantité de données qui peuvent être transmises entre deux événements. Par exemple, la quantité d'information que le processeur peut envoyer sur le bus entre une requête et une autorisation d'accès de chacun des composants du système. Il nous suffit pour cela d'ajouter les lignes suivantes dans le modèle original :

```
(dans le module bus_master)
  VAR transmitting: boolean;
  ASSIGN init(transmitting) := FALSE;
         next(transmitting) :=
                 (next(count)<count);

(dans le module main)
  VAR duration: 0..1;
  ASSIGN next(duration):= case
         processor.transmitting: 1;
         1:                      0;
  esac;
  COMPUTE MAX[req0, grant=0]
  COMPUTE MAX[req1, grant=1]
  COMPUTE MAX[req2, grant=2]
  COMPUTE MAX[req4, grant=4]
```

Le résultat est de 30 pour tous les composants, sauf le processeur, qui ne peut transmettre que 15 bits entre une requête et une autorisation d'accès au bus. Rappelons que, dans le modèle étudié, un composant peut transmettre au plus 15 bits d'information en une transaction.

2.2 Le problème du pont [7]

Description. Quatre personnes souhaitent traverser un pont, de nuit. Elles ne disposent que d'une lampe, indispensable pour traverser le pont. Par ailleurs, le pont ne peut supporter au plus que deux personnes à la fois. Enfin, nos quatre aventuriers se déplacent à des vitesses différentes : il leur faut respectivement (au moins) 5, 10, 20 et 25 unités de temps pour faire la traversée.

Le problème consiste à trouver la durée minimale nécessaire pour que nos quatre personnes se retrouvent (simultanément) de l'autre côté du pont. La difficulté du problème tient au fait que, après que deux personnes ont traversé le pont (le plus rapide s'adaptant à la vitesse du plus lent), il faut que l'une d'elles ramène la lampe aux personnes restées de l'autre côté. Il faut donc bien choisir l'ordre de passage, afin de ne pas perdre trop de temps dans ces aller-retours.

Implantation et résultats. Une modélisation complète de ce problème est proposée sur la page web de TSMV, `http://www.lsv.ens-cachan.fr/~markey/TSMV/`.

Ce modèle peut être traité avec NuSMV et Verus. Avec ces outils, le codage des durées se fait en intercalant des états intermédiaires, afin d'avoir autant de transitions (de durée 1) que les durées possibles de chaque transition. Ce codage coûte cependant très cher : le tableau 1 donne les temps de calcul et les quantités de mémoire nécessaire pour le calcul du temps minimal pour faire traverser les quatre personnes, lorsque leur temps de traversée individuel est multiplié par une constante (par 10 sur la ligne « bridge $\times$ 10 » par exemple). Par ailleurs, ajouter des états intermédiaires a des « effets de bord » lors de l'interprétation de formules de RTCTL, et cela nécessite souvent de compliquer le modèle pour contourner ces problèmes.

	TSMV		NuSMV		Verus	
	temps	mémoire	temps	mémoire	temps	mémoire
bridge	0.02 s.	1.3 MB	0.21 s.	9.5 MB	7.14 s.	16.5 MB
bridge $\times$ 10	0.04 s.	1.3 MB	23.24 s.	18.5 MB	259.54 s.	39.2 MB
bridge $\times$ 20	0.07 s.	1.3 MB	120.13 s.	18.5 MB	573.05 s.	44.1 MB
bridge $\times$ 50	0.22 s.	9.0 MB	2 209 s.	35.0 MB	3 626 s.	55.0 MB
bridge $\times$ 100	0.45 s.	11.0 MB	14 296 s.	65.0 MB	17 870 s.	59.2 MB

TAB. 1 – Comparaison de TSMV, NuSMV et Verus sur le problème du pont

3 Conclusion

L'outil TSMV est une extension du model checker symbolique NuSMV qui permet de vérifier des systèmes simplement temporisés. Les algorithmes implémentés dans TSMV lui permettent d'être (presque) insensible à l'échelle de temps choisi. De plus, en utilisant des transitions de durée nulle, TSMV devient un outil de *condition counting*, permettant non plus de compter le temps, mais de compter le nombre d'occurrences d'un événement.

Nous aimerions que TSMV soit intégré dans l'outil NuSMV, ce qui nécessitera probablement encore quelques modifications mais augmentera les performances. Nous pensons qu'il est encore possible d'améliorer les algorithmes afin que TSMV soit *réellement* insensible à l'échelle.

Références

[1] J. Burch, E. Clarke, K. McMillan, D. Dill et L. Hwang. Symbolic model checking: 10^{20} states and beyond. *Inf. & Comp.*, 98(2), p. 142–170, Academic Press, juin 1992.

[2] S. Campos et E. Clarke. The Verus Language: Representing Time Efficiently with BDDs. *Theor. Comp. Sci.*, 253(1), p. 95–118, Elsevier Science, fév. 2001.

[3] S. Campos, E. Clarke, W. Marrero et M. Minea. Verifying the performance of the PCI local bus using symbolic techniques. *In Proc. 5th Intl Conf. Computer Design (ICCD'95)*, p. 72–78. IEEE Comp. Soc. Press, oct. 1995.

[4] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani et A. Tacchella. NuSMV2: An OpenSource Tool for Symbolic Model Checking. *In* E. Brinksma et K. Larsen, éds, *Proc. 14th Intl Conf. Computer Aided Verification (CAV 2002)*, *Lect. Notes Comp. Sci.* 2404, p. 359–364. Springer-Verlag, juil. 2002.

[5] N. Markey et Ph. Schnoebelen. Symbolic Model Checking of Simply-Timed Systems. Rapport technique 03-14, LSV, ENS Cachan, France, oct. 2003.

[6] K. McMillan. *Symbolic Model Checking – an Approach to the State Explosion Problem*. Thèse de doctorat, CMU, Pittsburgh, Pennsylvania, USA, 1993.

[7] G. Rote. Crossing the Bridge at Night. *EATCS Bulletin*, 78, p. 241–246, EATCS, oct. 2002.