Vérification formelle d'un réseau sur puce : Application de DEv-Promela.

Abdelhak Khemiri, Aznam Yacoub, Maamar El Amine Hamri

Vérification formelle d'un réseau sur puce : Application de DEv-Promela

A. Khemiri, A. Yacoub et M. Hamri
Aix-Marseille Univ, Université de Toulon, CNRS
LIS, Marseille, France
{abdelhak.khemiri, aznam.yacoub, amine.hamri}@lis-lab.fr

Résumé

Dans ce papier, nous présentons une approche pour vérifier et valider des systèmes dynamiques complexes qui reposent à la fois sur une vérification formelle des propriétés temporelles exprimées et une simulation (tester un scénario, quantifier des variables ou aider à la prise de décision) de la dynamique du système modélisé.

En effet, nous montrons qu'il est possible de combiner un vérificateur et une simulation pour tirer les avantages et combler mutuellement les inconvénients de chacun. Nous nous appuyons sur le langage DEv-PROMELA qui, lui, repose sur deux formalismes développés par deux communautés différentes : PROMELA pour vérifier formellement des systèmes asynchrones et DEVS pour modéliser et simuler des systèmes dynamiques.

Enfin, une étude de cas décrivant un réseau sur une puce électronique est présentée et les résultats obtenus sont analysés et commentés.

1 Introduction

La vérification formelle de systèmes logiciels et de programmes informatiques est un champ de recherche très actif avec des avancées théoriques récentes. Des progrès considérables en termes de méthodes et d'outils ont été réalisés rapprochant de plus en plus les techniques de vérification formelle des développeurs informatiques.

Prônant le rapprochement entre la simulation et la vérification formelle pour vérifier et valider des spécifications de système [6, 5], nous avons proposé le langage DEv-PROMELA (*Discrete Event Process MetaLanguage*) [8]. Ce langage, préservant les propriétés structurelles de PROMELA d'une part et les propriétés temporelles des DEVS d'autre part, décharge l'utilisateur de la construction d'abstractions formelles à partir des spécifications pour démontrer une propriété quelconque. En effet le modèle PROMELA est extrait automatiquement à partir d'une spécification DEv-PROMELA évitant l'introduction de nouvelles erreurs. En revanche, le modèle DEVS extrait automatiquement aussi permet de jouer des scénarios et réaliser des mesures par simulation.

Afin de montrer l'utilité de ce langage pivot et de son cadre de travail, nous décrivons un réseau sur puce électronique et nous étudions ses propriétés par une vérification formelle hybride. Cette étude repose à la fois sur la simulation DEVS et le model checking pour traiter deux difficultés rencontrées lors de la recherche d'erreurs dans des systèmes complexes. Le premier objectif est de repousser le plus loin possible le problème d'explosion combinatoire de l'espace d'états. Enfin, le second objectif vise à réduire la différence (distance) entre le modèle formel de vérification et le code exécutable final.

2 Rappels sur DEv-Promela

DEv-PROMELA (Discrete Event Process MEta LAnguage) est une extension de PROMELA, un langage développé pour la description et la vérification formelle de processus asynchrones. DEv-PROMELA quant à lui est développé pour la description et la vérification de systèmes à événements discrets complexes, en utilisant une technique de vérification formelle à base de model checking et une simulation à événements discrets. Son vérificateur repose sur l'interpréteur SPIN © pour vérifier toute propriété exprimée en LTL (Linear Temporal Logic) et le simulateur DEVS pour générer des traces d'exécution à partir de scénarios exprimés initialement.

Le langage DEv-PROMELA intègre des concepts de la modélisation DEVS permettant de raffiner une spécification PROMELA. En effet, les instructions peuvent être datées, déclenchées suite à l'occurrence d'un événement ou émettre un événement entre des processus. L'intérêt d'un tel langage est qu'il permet d'avoir un modèle unique où une vérification formelle des propriétés du système et une simulation de scénarios par une génération de trace d'exécution peuvent être effectuées. Pour ce faire, deux modèles sont extraits à partir de la description DEv-PROMELA : (1) le modèle PROMELA garantissant la structure du modèle DEv-PROMELA et permettant d'effectuer une vérification formelle, et (2) le modèle DEVS garantissant la dynamique du modèle DEv-PROMELA pour une simulation.

Grâce aux relations de morphisme qui existent entre DEv-PROMELA et PROMELA d'une part, et DEv-PROMELA et DEVS d'autre part, les deux modèles extraits de vérification et de simulation re-décrivent fidèlement le modèle initial DEv-PROMELA. Les conclusions déduites restent vraies pour le modèle DEv-PROMELA à vérifier. Ainsi, toute propriété prouvée au niveau du modèle PROMELA extrait, est une propriété prouvée pour le modèle DEv-PROMELA. De même, toute trace d'exécution générée par le modèle DEVS est générée par le modèle DEv-PROMELA.

Ces règles de construction des modèles PROMELA et DEVS, ainsi que la préservation des propriétés structurelles et dynamiques de chaque modèle sont détaillées dans les travaux de thèse d'Aznam Yacoub [4].

2.1 PROMELA

PROMELA est un langage de spécification de processus informatiques. Il permet la vérification de protocoles synchrones et asynchrones entre processus. Basé sur le langage de garde de Dijikstra [2], sa syntaxe est très proche d'un langage impératif (le langage C, etc.) rendant son utilisation facile par rapport à d'autres langages formels. En effet, ce langage a été développé dans le but d'une transformation possible et facile d'une implémentation vers un modèle de vérification et de réduction d'erreurs de codage.

Bien que PROMELA possède une syntaxe et une sémantique très proche du langage C, la différence réside au niveau de l'abstraction, extraite à partir du code, qui intègre des comportements (exécutions) non-déterministes à l'inverse d'un programme C. Notons qu'une spécification PROMELA se décline en deux parties : (1) une spécification fonctionnelle et comportementale décrivant le système modélisé, et (2) des propriétés à vérifier.

Afin de combler le manque de concepts et d'éléments temporels essentiels à la quantification (mesure) d'une exécution, quelques extensions de PROMELA ont été proposées. Nous citons real-time PROMELA, discrete-time PROMELA et timed PROMELA (une étude comparative de ces extensions est fournie dans [7]). Ces extensions s'accordent sur une représentation discrète du temps qui peut ralentir considérablement une simulation et provoquer des erreurs de précision des résultats obtenus (un événement qui ne se produit pas à la bonne date d'occurrence, etc.). Par la suite, nous préconisons une modélisation DEVS à base d'événements offrant une représentation continue du temps.

2.2 DEVS

DEVS offre un cadre de modélisation et simulation de systèmes dynamiques dirigé par des événements [9]. Il sépare les besoins de modélisation à la charge de l'utilisateur (modélisateur) de la mécanique de simulation réutilisable comme telle dont le but est de reproduire la dynamique du système modélisé. En revanche, la modélisation consiste à décrire un système par un ensemble de composants interconnectés entre eux pour l'envoi et la réception d'événements. Les composants sont décrits à l'aide de modèles atomiques ou couplés. Un modèle atomique est une structure algébrique $M = (X, Y, S, \delta_{ext}, \delta_{int}, \lambda, D)$ qui décrit un comportement par un ensemble d'états (segments) S. En effet, l'état du modèle M évolue dans l'un des deux cas : (1) suite à l'occurrence d'un événement externe $x \in X$ provoquant l'exécution de la fonction $\delta_{ext}()$, ou (2) à la différence entre les dates d'occurrence de deux événements successifs dépassant la durée de vie de l'état courant $D(s)$. Un modèle couplé $MC = (X, Y, D, M_{d \in D}, EIC, EOC, IC, select)$ est un réseau de modèles interconnectés par trois différents types de relations de couplage (EIC, EOC, IC). Un tel modèle est fermé sous couplage, ie., il existe son équivalent en atomique produisant les mêmes comportements. Ainsi, un modèle couplé est légitime à la simulation.

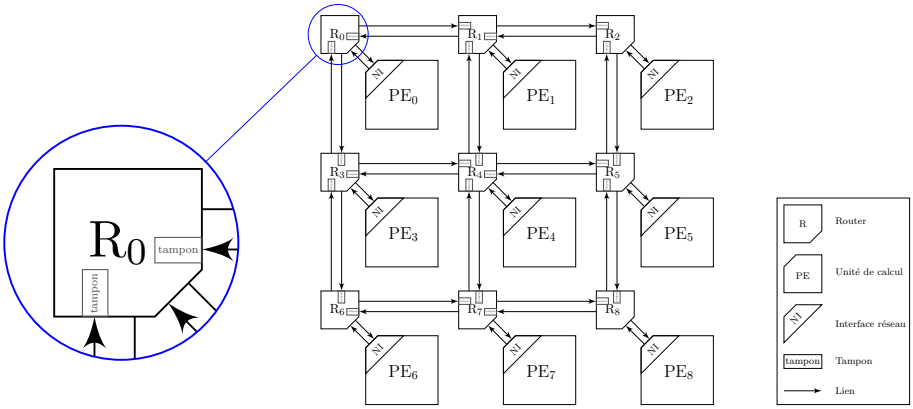

FIGURE 1 – Exemple d'un réseau sur puce maillé en 2 dimensions de taille 3×3

De nombreuses extensions DEVS ont été développées dans le but de répondre à des problèmes particuliers. Parmi elles, nous citons PDEVS pour décire des processus parallèles, DS-DEVS pour décrire des systèmes à structure variable, RT-DEVS pour décrire des systèmes temps réel, etc. Notons aussi que peu de travaux d'analyse formelle des modèles de simulation DEVS ont vu le jour. Bien que ce formalisme ait pour vocation la description de modèles simulables, quelques travaux ont étudié les aspects théoriques de ce formalisme : rapprochement entre DEVS et Timed Automata, analyse statique des modèles DEVS, proposition d'une hiérarchie de formalismes DEVS, etc.

3 Etude de cas : réseau sur puces électroniques

3.1 Description

Un NoC (*Network on Chips*) est un sous-système de communication sur un circuit intégré (puce), entre des coeurs de propriété intellectuelle. Ces différents composants communiquent alors entre eux en échangeant des paquets au travers d'un réseau d'interconnexions, fournissant une structure de communication réutilisable. Figure 1 décrit un exemple de NoC maillé en 2 dimensions (*mesh*) de taille 3×3 avec l'ensemble des éléments qui le compose.

Rappelons que les principaux composants d'un NoC sont : (i) les unités de calcul, (ii) les routeurs qui dirigent les données dans le réseau en accord avec un protocole et une stratégie de routage prédéfinis (iii) les liens qui connectent physiquement les routeurs deux à deux. Un lien peut être unidirectionnel ou bidirectionnel et peut fournir plusieurs canaux virtuels de communication, (iv) les interfaces réseaux dont le but est de séparer et faire l'interface entre le protocole de communication au sein du réseau, et le protocole dans les unités de calcul, et (v) les tampons mémoires qui peuvent être ajoutés en entrée et/ou en sortie des routeurs selon la stratégie de routage mise en place.

L'organisation de l'interconnexion entre les composants au sein d'un réseau est donnée par la topologie de ce dernier. Bien qu'il existe plusieurs topologies, celle de type maillé en 2 dimensions est la plus courante (comme illustré dans Figure 1).

Un des éléments importants du protocole est la manière dont les messages transitent. Dans ce qui suit, nous considérons uniquement la technique de commutation de paquets. Dans cette technique, les unités de calcul s'échangent des messages préalablement divisés en paquets, au travers du réseau. Un paquet correspond alors à la plus petite unité de communication qui contient aussi des informations de routage et de séquençage. Chaque paquet est lui même formé d'un ensemble de *flit* (*FLow control unIT*) qui correspond à la plus petite unité de contrôle de flux sur un lien. Le paquet débute alors par une entête (*header*) qui contient des informations relatives au routage. Le corps du paquet quant à lui contient la charge utile qui correspond au message à transmettre. Enfin, le paquet se termine par une queue, indiquant sa fin.

A cette stratégie de commutation, vient s'ajouter une politique de mémorisation au sein des routeurs. Bien qu'il existe plusieurs politiques de mémorisation, une des plus connues est la mémorisation *wormhole* [1]. Dans cette dernière, les routeurs ne mémorisent pas le paquet en entier avant de le transmettre au routeur suivant. Un *flit* d'entête alors détermine la direction de routage et se transmet en premier lieu. Les autres *flits* le suivent en pipeline.

Un autre élément essentiel du protocole de communication est la stratégie de routage. Cette dernière a pour but de déterminer le chemin à suivre entre la source et la destination d'un paquet. L'un des algorithmes de routage les plus utilisés dans le cas d'une topologie maillé en 2 dimensions est le routage ordonné par dimension X-Y. Il impose que les routeurs acheminent un paquet sur la dimension X puis sur la dimension Y. Cette stratégie a la particularité d'être déterministe (ie., pour un couple (source, destination), le chemin est toujours le même). Plus de détails concernant les réseaux sur puces sont présentés dans [1].

Notons que cette technique de routage est reconnue pour exclure le risque d'interblocage. En mettant les ressources du réseau dans un ordre particulier et en exigeant que les paquets demandent et utilisent ces ressources dans un ordre strictement monotone, l'attente circulaire (condition nécessaire à l'apparition d'un interblocage) est évitée. De nombreuses propriétés peuvent être vérifiées à l'aide de modèles formels, allant de l'évaluation du rendement à la correction de la communication, y compris la vérification fonctionnelle et l'analyse comportementale. Ici, nous examinerons les propriétés de correction de NoCs telles que l'absence d'interblocage.

3.2 Vérification formelle des propriétés temporelles

Afin de s'assurer qu'un modèle M vérifie une propriété p ($M \models p$), il faudra que le modèle en question s'appuie sur une notation mathématique sinon il est impossible démontrer une quelconque propriété.

Nous nous intéressons à la propriété d'absence d'interblocage entre unités de

calcul. Il faudra s'assurer alors qu'un message puisse arriver toujours à sa destination. Le modèle PROMELA décrivant un NoC de taille 3×3 montre qu'il est impossible de vérifier une telle propriété dû à la description très détaillée nécessaire à la simulation engendrant une explosion de l'espace d'états. En effet, les différents algorithmes de parcours du graphe d'exécution, généré à partir de ce modèle, échouent en n'arrivant pas à parcourir l'espace d'états d'une manière exhaustive. Nous avons alors proposé une approche qui explore seulement les sous-espaces d'états critiques par simulation jusqu'à leur détection, ensuite la vérification formelle prend le relais [3] . Cette approche, malheureusement, ne garantit pas une équivalence entre les deux modèles de simulation et de vérification sur lesquelles elle repose. En revanche, le modèle DEv-PROMELA décrivant le même NoC permet l'extraction à la fois d'un modèle PROMELA pour vérifier la propriété d'absence d'interblocage et d'un modèle de simulation DEVS pour quantifier certaines variables de l'interblocage. Rappelons que les modèles extraits préservent les propriétés structurelles et comportementales du modèle initial DEv-PROMELA. Ces différents modèles sont consultables à l'adresse URL `https://github.com/khemiriabdelhak/NetworkOnChip`.

3.3 Comparaison des résultats

Nous proposons de modéliser le NoC 3×3 dont certaines unités de calcul sont défectueuses à l'aide de DEv-PROMELA puis d'utiliser les modèles DEVS et PROMELA ainsi obtenus afin de les combiner et de vérifier que le système ne contient pas d'interblocage. L'approche combinée est comparée à l'approche classique de model checking tel que proposé par l'outil de vérification SPIN. Les deux approches sont comparées au regard du nombre d'états visités, du nombre de transitions effectuées, de la quantité de mémoire utilisée et de la durée totale de la recherche. Dans le cas de l'approche combinée, le temps de simulation pour atteindre l'état critique est indiqué. Les résultats sont résumés dans Table 1.

La vérification classique du modèle a épuisé toutes les ressources sans trouver d'erreur et a atteint les limites mémoire avant d'arrêter la recherche avec les options *collapse compression*, *hash-compact* et *exhaustive*. Avec l'option *exhaustive*, la recherche a été arrêtée avant de conclure après 16.2 secondes et 5 Go de mémoire utilisée, ce qui correspond à $1\ 660\ 720$ états uniques explorés et $3\ 670\ 664$ transitions effectuées. Avec l'option *collapse compression*, la recherche a épuisé toute la mémoire disponible après l'exploration de 83 millions d'états uniques, et plus de 10^8 transitions effectuées le tout en 577 secondes. L'option *hash-compact*, bien que permettant l'exploration de plus d'états (plus de 10^8), n'a pas pu trouver d'erreur, et a interrompu la recherche après 491 secondes. La vérification classique du modèle montre des résultats décourageants par rapport à l'approche combinée qui a trouvé l'erreur d'interblocage. En effet, pour chacune des configurations considérées, l'approche combinée a utilisé moins de 140 Mo de mémoire, effectué 2052 transitions et exploré 1992 états uniques en moins d'une seconde.

	MC Classique	Approche combinée
Exhaustive		
Erreur trouvée	Aucune (interrompu)	Interblocage
Nombre d'états	1 660 720	1 992
Nombre de transitions	3 670 664	2 052
Consommation mémoire (Mo)	5 119.940	135.565
Temps de vérification (s)	16.2	0.02
Temps de simulation (s)	N/A	2.20
Temps total (s)	16.2	2.22
Collapse compression		
Erreur trouvée	Aucune (interrompu)	Interblocage
Nombre d'états	83 104 696	1 992
Nombre de transitions	186 830 090	2 052
Consommation mémoire (Mo)	5 119.831	130.097
Temps de vérification (s)	577	0.01
Temps de simulation (s)	N/A	2.20
Temps total (s)	577	2.21
Hash-compact		
Erreur trouvée	Aucune (interrompu)	Interblocage
Nombre d'états	117 803 750	1 992
Nombre de transitions	152 787 170	2 052
Consommation mémoire (Mo)	5 119.831	129.315
Temps de vérification (s)	491	0.01
Temps de simulation (s)	N/A	2.20
Temps total (s)	491	2.21

TABLE 1 – Comparaison des performances de l'approche combinée avec l'approche classique de Model Checking (MC) pour un NoC 3×3

4 Conclusion

La simulation fournit des mesures quantitatives et qualitatives qui reflètent le comportement du modèle et qui ne peuvent pas être obtenues par une requête du modèle de vérification. Par conséquent, une simulation peut être utilisée pour identifier — au moyen de conditions sur les mesures effectuées — un sous-ensemble d'états où une propriété donnée est plus susceptible d'être insatisfaite. Une simulation peut alors permettre à la procédure de vérification de se concentrer sur un sous-ensemble de l'espace total d'état, et de limiter le problème de l'explosion de cet espace d'états. Cependant, il est nécessaire de garantir une équivalence entre le modèle de simulation et de vérification. Malheureusement, cette équivalence reposait sur les capacités du modélisateur à créer deux modèles, dans deux formalismes, qui soient équivalents au regard de la propriété à vérifier. Pour cela, nous avons montré au travers d'une étude de cas, la pertinence de l'utilisation du formalisme DEv-PROMELA décrivant le même système, et qui permet l'extraction à la fois d'un modèle PROMELA et d'un modèle DEVS qui préservent les propriétés structurelles et comportementales du modèle initial DEv-PROMELA.

Dans nos travaux futurs, nous allons nous intéresser à la mise à l'échelle de cette étude de cas et mettre en lumière les avantages et inconvénients de notre approche combinée reposant sur le langage DEv-PROMELA.

Remercients

Nous remercions les trois relecteurs anonymes de notre article pour les commetaires très constructifs et les présidents du comité de programme d'AFADL de nous avoir donné l'opportunité de publier nos travaux de recherche.

Références

[1] É. Cota, A. de Morais Amory, and M. Soares Lubaszewski. *Reliability, Availability and Serviceability of Networks-on-Chip*. Springer US, 1st edition, 2012.

[2] G. Holzmann. *The SPIN Model Checker : Primer and Reference Manual*. Addison-Wesley Professional, 1st edition, 2011.

[3] A. Khemiri, M. Hamri, C. Frydman, and J. Pinaton. Limiting State Space Explosion of Model Checking Using Discrete Event Simulation : Combining DEVS and PROMELA. In *Proceedings of Computer Simulation Conference 2019*, Berlin, Germany, July 2019.

[4] A. Yacoub. *Une approche de vérification formelle et de simulation pour les systèmes à événements : application à PROMELA*. PhD thesis, 2016. Thèse de doctorat dirigée par Frydman, C. et Hamri, M. Informatique Aix-Marseille 2016.

[5] A. Yacoub, M. Hamri, and C. Frydman. Complementarity between simulation and formal verification transformation of PROMELA models into FDDEVS models : Application to a case study. In *4th International Conference On Simulation And Modeling Methodologies, Technologies And Applications, SIMULTECH 2014, Vienna, Austria, August 28-30, 2014*, pages 421–426. IEEE, 2014.

[6] A. Yacoub, M. Hamri, and C. Frydman. A method for improving the verification and validation of systems by the combined use of simulation and formal methods. In *18th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications, DS-RT 2014, Toulouse, France, October 1-3, 2014*, pages 155–162. IEEE Computer Society, 2014.

[7] A. Yacoub, M. Hamri, and C. Frydman. Using DEv-PROMELA for modelling and verification of software. In *Proceedings of the ACM Conference on SIGSIM Principles of Advanced Discrete Simulation, SIGSIM-PADS 2016, Banff, Alberta, Canada, May 15-18, 2016*, pages 245–253. ACM, 2016.

[8] A. Yacoub, M. Hamri, C. Frydman, C. Seo, and B. P. Zeigler. DEv-PROMELA : an extension of PROMELA for the modelling, simulation and verification of discrete-event systems. *IJSPM*, 12(3/4) :313–327, 2017.

[9] B. P. Zeigler, A. Muzy, and E. Kofman. *Theory of Modeling and Simulation : Discrete Event & Iterative System Computational Foundations*. Academic Press, Inc., USA, 3rd edition, 2018.