Vérification de réseaux paramétrés par analyse d'accessibilité

Tayssir Touili

Les systèmes informatiques sont de plus en plus présents dans le contrôle de tâches critiques et très complexes. Une erreur dans la conception de ces systèmes peut avoir des conséquences graves et irréversibles. C'est pourquoi il est crucial de disposer de méthodes rigoureuses pour les concevoir et de techniques automatiques pour les vérifier.

Le problème de la vérification consiste à s'assurer qu'un système satisfait bien ses spécifications. Ces dernières années, des méthodes de vérification automatiques ont été développées et sont largement utilisées. Seulement, ces méthodes concernent essentiellement les systèmes finis (à nombre fini d'états).

Nous considérons ici le problème de la vérification des réseaux paramétrés de processus, c'est-à-dire, des réseaux comprenant un nombre arbitraire de processus identiques. Il s'agit de vérifier un système quelque soit le nombre de ses composantes. Des exemples de réseaux paramétrés sont les algorithmes d'exclusion mutuelle, les protocoles de communication entre un nombre arbitraire de processus,...etc. La vérification de tels systèmes est hors de portée des techniques usuelles de vérification pour les systèmes finis.

Nous réduisons le problème de la vérification des systèmes paramétrés au calcul de l'ensemble des configurations accessibles. Cet ensemble étant infini (dû à la paramétrisation), nous adoptons une approche symbolique basée sur la représentation d'un ensemble infini de configurations par un langage de mots (resp. d'arbres) si la topologie du réseau est linéaire (resp. arborescente). Par exemple, dans le cas des réseaux linéaires, nous représentons l'état global d'un système ayant n processus par un mot de longueur n, en concaténant les états locaux des différents processus. Un ensemble de configurations peut donc être représenté par un langage de mots. Par exemple, l'ensemble des configurations d'un système qui vérifie la propriété d'exclusion mutuelle peut être représenté par n^*cn^* où c (resp. n) exprime que le processus est (resp. n'est pas) dans la section critique.

Une action du programme peut être alors modélisée par une règle de réécriture de mots (ou d'arbres). Ainsi, la règle $a \rightarrow b$ exprime qu'une composante du système passe de l'état a à l'état b.

Par exemple, si nous considérons le cas du "token passing protocol" où un système est formé par un vecteur de processus, l'action qui consiste à faire passer le jeton de la gauche vers la droite peut être modélisée par la règle (ou semi-commutation) $t\perp \rightarrow \perp t$ où t (resp. $\perp$) exprime que le processus a (resp. n'a pas) le jeton. Initialement, c'est le processus le plus à gauche qui a le jeton, l'ensemble des configurations initiales est donc représenté par $t\perp^*$.

Nous réduisons alors le problème de la vérification au calcul de la fermeture d'un langage régulier par un système de réécriture, c'est à dire au calcul de $R^*(L)$, où L est un langage régulier d'arbres ou de mots, et R est un système de réécriture. Ce problème étant indécidable, notre but est de:

- Proposer des sous-classes $\mathcal{L}$ de langages réguliers et $\mathcal{R}$ de systèmes de réécriture pour lesquelles le calcul de la fermeture de tout langage de la classe $\mathcal{L}$ par un système de réécriture de la classe $\mathcal{R}$ est effectif.
- Définir une approche symbolique générale (semi-algorithmique) pour le calcul de l'ensemble des accessibles.

Dans cet exposé, nous nous restreignons au cas des réseaux linéaires, c'est à dire aux langages réguliers de mots sur un alphabet fini Σ. Nous présentons dans ce qui suit deux résultats principaux:

1. Fermeture des APCs par semi-commutations

Dans un premier temps, nous considérons les semi-commutations, i.e., les règles de la forme $ab \to ba$. Ces règles apparaissent de manière naturelle dans la modélisation d'un grand nombre de protocoles, tel que le "token passing protocol" considéré précédemment. Notre but est alors de calculer $R^*(L)$ pour un langage régulier L et un ensemble de semi-commutations R. Seulement, ce type de règles ne préserve pas la régularité. En effet, pour $R = ab \leftrightarrow ba$, le langage $R^*((ab)^*)$ n'est pas régulier puisque c'est l'ensemble de tous les mots de $(a + b)^*$ qui contiennent le même nombre de "a" et de "b". Nous voulons alors une sous-classe des réguliers qui soit effectivement fermée par semi-commutations. Nous avons identifié la classe des *Alphabetic Pattern Constraints* (APC):

Définition 1 : Un langage **APC** est une union finie de langages de la forme $\Sigma_0^* a_1 \Sigma_1^* \cdots a_n \Sigma_n^*$, où les Σ_i sont des ensembles finis de lettres, et les a_i sont des lettres.

Cette classe de langages apparaît naturellement dans la modélisation des ensembles de configurations des réseaux paramétrés. Par exemple, le langage $\Sigma^* c \Sigma^* c \Sigma^*$ représente l'ensemble des configurations qui ne satisfont pas l'exclusion mutuelle.

Nous avons montré que cette classe est effectivement fermée par semi-commutations :

Théorème 1 [3] : Soit R un ensemble de semi-commutations, et L un langage APC, alors $R^*(L)$ est un langage APC et peut être effectivement calculé.

2. Calcul des accessibles par "*regular widening*"

De manière plus générale, nous adoptons une méthode semi-algorithmique basée sur l'accélération de la terminaison du calcul, qui permet de calculer une

sur-approximation de l'ensemble des accessibles. En effet, ceci s'avère suffisant pour vérifier certains systèmes.

Le principe de cette méthode nommée *regular widening* [2, 5] consiste à *deviner automatiquement* l'effet de l'itération de R un nombre arbitraire de fois sur un ensemble régulier L donné: si une telle situation

$$L = L_1.L_2 \text{ et } R(L) = L_1.\Delta.L_2$$

est détectée, nous devinons qu'à chaque fois l'effet de R est d'introduire un "Δ" au milieu, nous rajoutons donc $L_1.\Delta^*.L_2$ à l'ensemble des accessibles. Un principe plus général qui tient compte du cas où R introduit plusieurs croissances est défini dans [5]. De manière plus générale, si nous représentons R par un langage de $\Sigma \times \Sigma$, ce même mécanisme permet de deviner l'effet de R^*, la fermeture reflexive-transitive de R.

Ce principe peut être utilisé pour calculer l'ensemble d'accessibilité exact si nous avons un *test* qui permet de décider si l'ensemble deviné est *exactement égal à* $R^*(L)$. Pour définir ce test, nous introduisons quelques définitions:

Définition 2 : Un système de réécriture R est nœthérien s'il n'existe pas une séquence infinie de mots w_0,$w_1, \ldots$ tels que pour chaque $i \geq 0$, $w_{i+1} \in R(w_i)$.

Définition 3 : Si R est un système de réécriture qui comprend les règles $\{l_i \rightarrow r_i\}$, R^{-1} est le système de réécriture qui comprend les règles $\{r_i \rightarrow l_i\}$.

Nous avons alors le résultat suivant dont une partie est due à [4]:

Proposition 1 : Si R ou R^{-1} est nœthérien alors $L' = R^*(L)$ ssi $L' = R(L') \cup L$.

Ainsi, si R ou R^{-1} est nœthérien, nous pouvons utiliser notre *regular widening* pour deviner l'ensemble des accessibles, et appliquer le test précédent pour nous assurer que notre calcul est exact.

Notre méthode s'avère être assez puissante pour simuler plusieurs constructions existantes. En effet, elle peut simuler le résultat du Théorème 1:

Théorème 2 [5] : Soit R un ensemble de semi-commutations, et L un langage APC, alors $R^*(L)$ peut être effectivement calculé par *regular widening*.

Dans [1], Abdulla et al. ont défini la classe des "*règles de réécriture contextuelles*" et ont donné une construction de R^* pour toute règle R dans cette classe. Notre technique est capable de calculer cette fermeture:

Théorème 3 [5] : Soit R une *règle de réécriture contextuelle*, alors R^* peut être effectivement calculé par *regular widening*.

Références

[1] P. A. Abdulla, A. Bouajjani, B. Jonsson, and M. Nilsson. Handling global conditions in parametrized system verification. *Lecture Notes in Computer Science*, 1633:134–150, 1999.

[2] A. Bouajjani, B. Jonsson, M. Nilsson, and T. Touili. Regular model checking. In *12th Intern. Conf. on Computer Aided Verification (CAV'00)*. LNCS, Springer-Verlag, 2000.

[3] A. Bouajjani, A. Muscholl, and T. Touili. Permutation Rewriting and Algorithmic Verification. In *Proc. 17th Symp. on Logic in Computer Science (LICS'01)*. IEEE, 2001.

[4] L. Fribourg and H. Olsen. Reachability sets of parametrized rings as regular languages. In *Infinity'97*. volume 9 of Electronical Notes in Theoretical Computer Science. Elsevier Science, 1997.

[5] T. Touili. Widening Techniques for Regular Model Checking. In *1st vepas workshop*. Volume 50 of Electronic Notes in Theoretical Computer Science, 2001.

Tayssir Touili
Laboratoire LIAFA
Université Paris VII
2, place Jussieu, case 7014
F-75251 Paris Cedex 05
France
`touili@liafa.jussieu.fr`
http://verif.liafa.jussieu.fr/∼touili