Une approche parallèle et distribuée pour la complétion d'automates d'arbre

Adrian Caciula, Roméo Courbis, Violeta Felea, Pierre-Cyrille Heam, Rasvan Ionescu

Une approche parallèle et distribuée pour la complétion d'automates d'arbre

A. Caciula et R. Courbis et V. Féléa et P.-C. Héam et R. Ionescu

LIFC - INRIA - Université de Franche-Comté
http://lifc.univ-fcomte.fr

Résumé. La technique dite *de complétion* sur les automates d'arbre permet, à partir d'un automate d'arbre et d'un système de réécriture, de calculer une sur-approximation de l'ensemble des termes accessibles. Cette technique a été utilisée avec succès pour la vérification de protocoles de sécurité et, plus récemment, pour l'analyse des programmes Java. Comme dans toute approche de vérification par *model-cheking*, nous sommes confrontés à des problèmes d'explosion combinatoire lorsque les exemples deviennent plus conséquents. Dans cet article nous montrons comment faire face à cette situation en parallélisant et distribuant les calculs. Nous présentons aussi quelques résultats expérimentaux qui montrent que l'approche permet d'obtenir de meilleures performances en temps d'exécution.

1 Introduction

L'informatisation toujours croissante de la société nous impose de développer des logiciels dont la qualité est de plus en plus irréprochable. Dans ce contexte, de nombreuses techniques complémentaires de test et de vérification ont été développées ces dernières années. Le théorème de Rice Rice (1953) montre que tout problème de vérification non-syntaxique est indécidable. Les approches consistent alors soit à rechercher des sous-cas décidables (comme l'accessibilité dans les réseaux de Petri) ou de développer des techniques semi-algorithmiques. Ce papier se place dans ce second cas pour la vérification de propriétés de sûreté : on souhaite garantir que le système n'atteindra pas une mauvaise configuration.

Plus formellement, le problème de sûreté peut être codé ainsi : étant donnés un ensemble I de configurations initiales, une relation R codant l'évolution du système et un ensemble B de mauvaises configurations, a-t-on $R^*(I) \cap B = \emptyset$? Si cette intersection est vide, le système est sûr (vis-à-vis de la propriété encodée par B), sinon il ne l'est pas. Ce problème est indécidable en général Gilleron et Tison (1995),Sakarovitch (1992), et nous nous intéressons ici à la technique dite *de complétion* Feuillade et al. (2004), qui modélise les configurations du système par des termes. Les ensembles I et B sont des ensembles réguliers de termes codés par des automates d'arbre (voir Comon et al. (2002) pour plus de détails sur les automates d'arbre). La relation R est donnée par un système de réécriture classique, que nous noterons aussi R. Cette technique vise à calculer un sur-ensemble K de $R^*(I)$ tel que $K \cap B$ soit vide, ce qui prouve alors la sûreté du système. Cette technique a été utilisée avec succès pour la vérification de protocoles de sécurité Boichut et al. (2008). Elle est aussi utilisée dans Boichut et al. (2007)

pour faire de l'analyse de code java pour téléphones mobiles. Elle montre cependant des limites algorithmiques, la taille des automates et le nombre de règles de réécriture étant trop importants pour analyser des programmes de longue taille. Dans cet article nous explorons des pistes pour ce problème en montrant comment distribuer le processus de complétion. Cette approche exploite le fait que certains calculs de procédures peuvent se faire dans un ordre quelconque.

Dans la partie 2 nous expliquons brièvement l'algorithme de complétion. La partie 3 dévoile comment nous avons parallélisé l'algorithme. Enfin, la partie 4 donne les résultats expérimentaux.

2 Complétion

Nous ne donnons ici qu'un aperçu du fonctionnement de la procédure de complétion. Pour plus de détail voir Feuillade et al. (2004). Si $\mathcal{A}$ est un automate d'arbre et Δ un ensemble de transitions, on note $\mathcal{A} \oplus \Delta$ l'automate obtenu à partir de $\mathcal{A}$ en y ajoutant les transitions de Δ.

Avec les notations de l'introduction, la procédure prend en entrée un automate d'arbre $\mathcal{A}_0$ reconnaissant I, un système de réécriture R et fonctionne selon le schéma suivant.

Complétion($\mathcal{A}_0$,R)
Entrées : $\mathcal{A}_0$ automate d'arbre, R système de réécriture.
Variables : $\mathcal{A}, \mathcal{B}$ automates d'arbre, Δ ensemble de transitions.
Algorithme :
$\mathcal{A} := \mathcal{A}_0$
Faire
$\mathcal{B} := \mathcal{A}$
Pour chaque règle r de R
Calculer Δ tel que $L(\mathcal{A}) \cup r(L(\mathcal{A})) \subseteq L(\mathcal{A} \oplus \Delta)$ (*)
$\mathcal{A} := \mathcal{A} \oplus \Delta$
FinPour
TantQue $\mathcal{B} \neq \mathcal{A}$
Retourner $\mathcal{A}$

A chaque étape de la boucle *Pour*, sont ajoutées à l'automate des transitions permettant de reconnaître au moins les termes accessibles en utilisant une fois la règle r. A la fin de chaque itération de la boucle *Faire* on a $L(\mathcal{B}) \cup R(L(\mathcal{B})) \subseteq L(\mathcal{A})$. A la fin de l'algorithme on obtient alors un automate $\mathcal{A}$ reconnaissant une sur-approximation (pour l'inclusion) de $R^*(L(\mathcal{A}_0))$. La problématique générale est de s'assurer de la convergence (terminaison) de l'algorithme.

3 Approche distribuée et parallèle

L'idée est de découper le système de réécriture en plusieurs sous-systèmes et d'appliquer la technique de complétion sur chacun des sous-systèmes sur des machines différentes, puis de synchroniser les résultats (par une union d'automates). La correction de l'approche réside dans le fait qu'il est possible d'effectuer plusieurs fois la ligne (*) pour chaque règle : cela change la sur-approximation obtenue mais pas le fait d'obtenir une sur-approximation. En supposant que l'on dispose de n machines, l'algorithme suivant donne l'approche distribuée des calculs.

Complétion($\mathcal{A}$,R) distribuée
Entrées : $\mathcal{A}_0$ automate d'arbre, R système de réécriture.
Variables : $R_1, \ldots, R_n$ des systèmes de réécritures et $\mathcal{A}_1, \ldots, \mathcal{A}_n, \mathcal{A}, \mathcal{B}$ automates d'arbre.
Algorithme :

```
A := A_0
Pour tout i dans {1,...,n} A_i := A_0 FinPour
Partionner R en R_1 ∪ R_2 ... ∪ R_n
Faire
    B := A
    Pour tout i dans {1,...,n}
        A_i :=Complétion(A_i, R_i)                (#)
    FinPour
    A := A_1 ∪ A_2 ... ∪ A_n
TantQue B ≠ A
Retourner A
```

Chaque ligne (#) est effectuée sur une machine différente. L'union des $\mathcal{A}_i$ est effectuée à partir de $\mathcal{B}$ en ajoutant toutes les transitions calculées à la ligne (#).

4 Résultats expérimentaux

L'algorithme proposé dans la partie 3 a été exécuté avec différentes heuristiques de calcul assurant la convergence des appels à la procédure complétion. Pour cela, nous avons utilisé une implémentation séquentielle de la complétion programmée en Caml. Nous avons programmé une sur-couche en Java qui gère les communications entre machines. La communication entre l'outil Caml et l'outil Java est effectuée à l'aide de fichiers. Les expérimentations ont été menées sur 2, 4 ou 8 machines (de PC de salles étudiants) sur trois systèmes de réécritures : l'un codant une instance difficile du problème de correspondance de Post, un autre codant une spécification de deux processus communicants et un dernier modélisant un protocole cryptographique. Nous utilisons aussi deux heuristiques différentes pour la convergence de la complétion.

Les expérimentations démontrent une amélioration des temps d'exécution par rapport au temps en séquentiel (par exemple sur la spécification du protocole cryptographique, le temps en séquentiel est de 181 secondes et de 48 secondes pour 4 machines). A chaque fois nous avons mesuré l'accélération, qui est donnée par la formule $a = \frac{T_{\text{distri}}}{n * T_{\text{seq}}}$ où n est le nombre de machines, T_{distri} est le temps de calcul pour l'application distribuée et T_{seq} le temps pour le même calcul lancé en séquentiel. Compte tenu des temps de transfert, théoriquement $a \leq 1$ (plus l'accélération est proche de 1, meilleure est l'approche distribuée).

	Pb de Post			Deux Processus			Protocole Crypt.		
nb machines	2	4	8	2	4	8	2	4	8
heuristique 1	0.93	0.76	0.6	0.7	0.31	0.23	0.87	0.64	0.26
heuristique 2	0.96	0.79	0.6	0.7	0.38	0.26	0.95	0.96	0.36

Expérimentalement, l'accélération diminue sensiblement avec le nombre de machines ce qui est fréquent car le nombre de communications augmente d'autant. On voit que l'heuris-

tique 2 est meilleure que la première. Sans rentrer dans les détails cela était prévisible car la seconde demande plus de calculs locaux, donc le coût des communications devient en proportion plus faible. Les résultats obtenus sont appréciables et prometteurs (et confirmés par plus d'expérimentations que nous ne présentons pas ici).

5 Conclusion

On a présenté dans cet article des résultats expérimentaux sur une approche distribuée d'une technique de vérification. Ces résultats sont plutôt encourageants, notamment en utilisant 2 ou 4 machines. Afin d'obtenir de meilleurs gains, il serait nécessaire d'affiner la programmation : nous avons utilisé une application locale sur chaque machine qui calcule la complétion et implante une sur-couche qui gère les communications à l'aide de fichiers. Il serait nécessaire maintenant de reprogrammer le tout dans une architecture complètement distribuée, ce qui devrait augmenter significativement les performances.

Références

Boichut, Y., T. Genet, T. P. Jensen, et L. L. Roux (2007). Rewriting approximations for fast prototyping of static analyzers. In *RTA*, pp. 48–62.

Boichut, Y., P.-C. Héam, et O. Kouchnarenko (2008). Approximation based tree regular model-checking. *Nordic Journal of Computing 14*, 216–241.

Comon, H., M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, et M. Tommasi (2002). *Tree Automata Techniques and Applications*. Available at *http ://www.grappa.univ-lille3.fr/tata/*.

Feuillade, G., T. Genet, et V. V. T. Tong (2004). Reachability analysis over term rewriting systems. *J. Autom. Reasoning 33*(3-4), 341–383.

Gilleron, R. et S. Tison (1995). Regular tree languages and rewrite systems. *Fundam. Inform. 24*(1/2), 157–174.

Rice, H. G. (1953). Classes of recursively enumerable sets and their decision problems. *Transactions of the AMS 74*.

Sakarovitch, J. (1992). The "last" decision problem for rational trace languages. In *LATIN*, pp. 460–473.

Summary

The completion technique on tree automata allows, from a tree automaton and a rewriting system, to compute an over-approximation of reachable terms. This technique was successfully used to verify security protocols and recently to analyze Java programs. As in many model-checking approaches we have to tackle a combinatoric explosion problem when addressing larger examples. In this article we show how to address this question using distributed and parallel approaches. We also present a couple of experimental results.