Vérification d'algorithmes de model-checking

Frédéric Gava & Julien Tesson
frederic.gava@univ-paris-est.fr ou julien.tesson@lacl.fr

Laboratoire d'Algorithme, Complexité, Logique
Université de Paris-Est Créteil (UPEC)
61 avenue du Général de Gaulle
94010 Créteil cedex
Équipe : Vérification et Spécification des Systèmes
Directeur du laboratoire : Régine Laleau (laleau@u-pec.fr)

Thématiques : Why, Coq, Vérification, Model-checking.

1 Problématique

Pour trouver des **bugs** dans des systèmes (agents communicants, protocoles de sécurité [1, 7], critiques, *etc.*), une technique largement reconnue et utilisée est le **model-checking explicite** [2]. Le principe est de se limiter à des scénarios fixes et de calculer toutes les traces d'exécution possibles depuis un **état initial** tout en vérifiant que ces traces restent valides suivant une **formule logique** donnée par l'utilisateur (dans la logique **temporelle** intrinsèque du model-checker). L'**espace des états** est donc cet ensemble de traces. Par exemple, dans le cas d'un protocole de sécurité, il est possible de trouver une propriété de sécurité violée (attaque) par un agent ou prouver qu'il n'existe pas une telle trace. En se basant sur un automate de Büchi et une décomposition de la formule logique (LTL), un algorithme de model-checking est intrinséquement une recherche de **composantes fortement connexes**. Si une telle composante est trouvée, la formule est considérée comme invalidant le système. Trouver une telle composante se fait généralement avec soit avec un algorithme **NDFS** [5] soit avec une adaptation du classique algorithme de **Tarjan** [8].

Le model-checking comporte deux sérieux problèmes :

1. Calculer cet espace des états peut s'avérer être **trop long** en temps de calculs.
2. L'utilisation de différentes techniques/algorithmes de réduction peut rendre faux les résulats obtenus

Le second problème n'en n'est pas un dans le cas de la recherche d'erreurs (les traces entraînant une violation de la formule logique) car les traces peuvent être facilement et rapidement vérifiées par un outil externe (en Coq p. ex.). On parle alors de certificats. Mais quand il n'y a pas une erreur de détectée, un model-checker ne produit que la réponse laconique "YES". On peut alors se poser la question de la validité d'une telle réponse. Et dans le cadre d'un model-checking explicite, la taille du certificat est proportionnelle à la taille de l'espace des états, c-à-d. trop grande en pratique. Les algorithmes de model-checking sont bien évidemment non triviaux. Et il est légitime de se poser la question de leur correction/validité. En effet, pour obtenir un model-chercker certifié qui peut servir de **référence** aux autres model-checkers, les algorithmes doivent déjà être prouvés corrects avant de s'intéresser à une implémentation (C ou Java) elle même correcte. Un model-checker bugué (comme ils le sont souvent [9]) peut trouver des faux-négatifs ou pire rater une trace d'erreur (ce qui peut engendrer une perte de temps quand on recherche ensuite à prouver le système quel que soit le scénario).

On trouve très peu de travaux sur la vérification des outils de model-checking où seule la vitesse d'exécution semble compter. La principale raison est que ces outils servent essentiellement à trouver des erreurs et non pas à dire que le système est correct. Ainsi, de fausses erreurs peuvent être trouvées. Néanmoins, rien ne dit que les *model-checkers* ne manquent pas des erreurs et il n'est pas rare de voir un modèle estampillé "formellement prouvé" même si on peut douter de la véracité de la réponse [10]. A notre connaissance, le premier développement (machine-checked avec l'assistant de preuves **Coq**, http://coq.inria.fr/) d'un model-checker a été fait dans [11]. On trouve par la suite, des développements de logiques temporelles en Coq dans [3] et [13]. On trouve aussi l'utilisation de model-checker pour la vérification des traces (les erreurs) et de la préservation de certaines

propriétés (mais pas du résultat final) [12]. Des algorithmes séquentiels prouvés ont aussi été écrits en B dans [14]. [15] présente le développement d'un algorithme de vérification LTL en Maude (réécriture). Dans [4], les auteurs ont développé un model-checker séquentiel pour LTL dans le prouveur Isabelle. Dans tout ces cas, ce sont des algorithmes purement fonctionnels qui sont vérifiés et donc pas forcement efficaces. Dans [6], nous avons travaillé sur la correction d'algorithmes séquentiels et distribués du calcul de l'espace des états (calculer le graphe mais pas les composantes fortement connexes).

Les algorithmes de model-checking (espace des états + vérification de la formule $\equiv$ composantes fortement connexes) de la littérature sont simples à formuler mais leurs propriétés logiques (**invariants de boucles**) sont en fait bien plus compliquées qu'il n'y paraît. Comme les outils de model-checking sont censés donner une réponse qui est une "vérité", il nous paraît souhaitable que leurs bons fonctionnements soient eux-mêmes **formellement vérifiés**. Pour cela, nous souhaitons **prouver formellement** (dans le sens **certifié/machine-checked**, et non pas juste sur papier) ces algorithmes. Ce travail servira de base pour la création d'un model-checker distribué certifié. Un précédent stage sur le sujet a permis la vérification d'une version simple de l'algorithme standard NDFS et des premiers résultats pour l'algorithme de Tarjan. Nous souhaitons étendre ce travail sur des algorithmes plus complexes (algorithme de Couvreur notament). De plus, il serait intéressant de pouvoir introduire des techniques de réductions (Ordre-partiels, BDD, symétries, etc.) pour être plus réaliste quand à l'efficacité du code.

2 Méthodes utilisées

Nous utiliserons l'outil **WHY** [1] ; Outil qui traite des *pre- post- conditions* et des invariants de boucles dans un mini langage ML avec effets de bords ; Cet outil permet donc **la génération d'assertions de preuves** à partir **des annotations du code**. Si toutes les obligations sont prouvées par les théorèmes prouveurs (automatiques ou non) alors la post-condition finale est correcte. L'outil WHY génère les **assertions** dans différents prouveurs comme des **théorèmes prouveurs automatiques** (p. ex. Z3 [2]) et à des assistants de preuves comme Coq.

Bien que cela ne soit pas nécessaire pour le stage, pour obtenir un model-checker explicite complet, nous avons besoin d'une fonction qui calcule les états **successeurs** d'un état du système. Dans le cadre de ce stage, cette fonction des successeurs sera bien évidement **abstraite** (posée en **axiome** en Why ou via un **type classe** en Coq).

3 But et déroulement envisagé du stage

Le but de ce stage est de **prouver formellement des algorithmes** de model-checking (calcul de composants fortement connexes) à l'aide de l'outils WHY. Il faudra donc écrire les algorithmes dans le langage Why-ML puis trouver les bons invariants de boucles et les faire vérifier par les théorèmes prouveurs.

Deux familles d'algorithmes existent : NDFS et Tarjan-like. Chacun avec ses avantages et inconvénients. Les prouver corrects en Why nous permettra de comparer les 2 algorithmes (en terme de temps et de difficulté de développement formel, performances des algorithmes extraits *etc.*) avant d'attaquer des codes distribués.
Le stage (5 mois) sera organisé de la manière suivante :

1. Premier mois :
   (a) Découverte rapide de la programmation Why-ML, de l'outil Why et des algorithmes de model-checking.
   (b) Apprentissage de Coq (si besoin)
   (c) Étude des algorithmes séquentiels ;
2. Deuxième mois : Formalisation des algorithmes séquentiels en WHY-ML ; codage des algorithmes.
3. Troisième, quatrième et cinquième mois : vérification de ces algorithmes : "jeu" essais/erreurs pour trouver les bons invariants et les bonnes propriétés logiques pour prouver les algorithmes.
4. Derniers 15 jours : rédaction du mémoire et préparation de la soutenance.

Des connaissances sur les prouveurs formels (automatiques ou assistés) et les méthodes formelles seraient vraiment les bienvenues mais ne sont pas obligatoires (un goût pour le formalisme et la logique est, en revanche, fortement conseillé). Une publication de ces résultats dans une conférence internationale est envisagée. Possibilité de poursuivre par une thèse dans l'équipe "Vérification" au LACL si une une allocation doctorale est obtenue. Comme pour tout stage de M2, une **indemnité mensuelle** correspondant au 1/3 du SMIC (environ 400 euros) sera accordée.

1. `http://why.lri.fr/index.fr.html`
2. `http://research.microsoft.com/projects/z3/`

Références

[1] David Basin, Cas Cremers, and Catherine Meadows. *Model Checking Security Protocols*, chapter 24. Springer, 2011.

[2] Edmund M. Clarke, Orna Grumberg, and Doron Peled. *Model checking*. MIT Press, 2001.

[3] S. Coupet-Grimal. An Axiomatization of Linear Temporal Logic in the Calculus of Inductive Constructions. *Logic and Computation*, 13(6) :801–813, 2003.

[4] Javier Esparza, Peter Lammich, René Neumann, Tobias Nipkow, Alexander Schimpf, and Jan-Georg Smaus. A fully verified executable ltl model checker. In Natasha Sharygina and Helmut Veith, editors, *Computer Aided Verification (CAV)*, volume 8044 of *LNCS*, pages 463–478. Springer, 2013.

[5] Sami Evangelista, Alfons Laarman, Laure Petrucci, and Jaco van de Pol. Improved multi-core nested depth-first search. In Supratik Chakraborty and Madhavan Mukund, editors, *Automated Technology for Verification and Analysis (ATVA)*, volume 7561 of *LNCS*, pages 269–283. Springer, 2012.

[6] Frédéric Gava, Jean Fortin, and Michaël Guedj. Deductive verification of state-space algorithms. In Einar Broch Johnsen and Luigia Petre, editors, *Integrated Formal Methods (IFM)*, volume 7940 of *LNCS*, pages 124–138. Springer, 2013.

[7] Frédéric Gava, Michaël Guedj, and Franck Pommereau. A bsp algorithm for on-the-fly checking ctl* formulas on security protocols. In Hong Shen, Yingpeng Sang, Yidong Li, Depei Qian, and Albert Y. Zomaya, editors, *Parallel and Distributed Computing, Applications and Technologies (PDCAT)*, pages 79–84. IEEE, 2012.

[8] Jaco Geldenhuys and Antti Valmari. More efficient on-the-fly ltl verification with tarjan's algorithm. *Theor. Comput. Sci.*, 345(1) :60–82, 2005.

[9] Fabrice Kordon, Alban Linard, Didier Buchs, Maximilien Colange, Sami Evangelista, Lukasz Fronc, Lom-Messan Hillah, Niels Lohmann, Emmanuel Paviot-Adet, Franck Pommereau, Christian Rohr, Yann Thierry-Mieg, Harro Wimmel, and Karsten Wolf. Raw report on the model checking contest at petri nets. *CoRR*, abs/1209.2382, 2012.

[10] Kedar S. Namjoshi. Certifying Model Checkers. In Gérard Berry, Hubert Comon, and Alain Finkel, editors, *Computer Aided Verification (CAV)*, volume 2102 of *LNCS*, pages 2–13. Springer, 2001.

[11] Christopher Sprenger. A Verified Model Checker for the Modal μ-calculus in Coq. In *Tools and Algorithms for Construction and Analysis of Systems (TACAS)*, volume 1384 of *LNCS*, pages 167–183. Springer-Verlag, 1998.

[12] Jun Sun, Yang Liu, and Bin Cheng. Model Checking a Model Checker : A Code Contract Combined Approach. In Jin Song Dong and Huibiao Zhu, editors, *Formal Engineering Methods (ICFEM)*, volume 6447 of *LNCS*, pages 518–533. Springer, 2010.

[13] Ming-Hsien Tsai and Bow-Yaw Wang. Formalization of CTL* in the Calculus of Inductive Constructions. In Mitsu Okada and Ichiro Satoh, editors, *Asian computing science conference on Advances in computer science : secure software and related issues (ASIAN)*, volume 4435 of *LNCS*, pages 316–330. Springer-Verlag, 2007.

[14] E. Turner, M. Butler, and M. Leuschel. A Refinement-based Correctness Proof of Symmetry Reduced Model-checking. In *Abstract State Machines, Alloy, B and Z*, LNCS, pages 231–244. Springer, 2010.

[15] Bow-Yaw Wang. Automatic verification of a model checker by reflection. In Pascal Van Hentenryck, editor, *Practical Aspects of Declarative Languages (PADL)*, volume 3819 of *LNCS*, pages 45–59. Springer, 2006.