Vérification par la preuve de propriétés temporelles sur des programmes C

Julien Groslambert
(Julien.Groslambert@trusted-labs.com)
Trusted-Labs
5 rue du Baillage, F-78000 Versailles

Nicolas Stouls*
(Nicolas.Stouls@insa-lyon.fr)
Université de Lyon, INRIA
INSA-Lyon, CITI, F-69621, France

Résumé

Dans le cadre du développement de la plate-forme *Frama-C*, permettant de vérifier des programmes C annotés, nous nous sommes intéressés à la vérification par la preuve de propriétés temporelles, *i.e.*, décrivant l'évolution du système au cours du temps.

État de l'art et problématique Il a été montré dans [1, 3] qu'une propriété décrite sous la forme d'un automate de Büchi B, peut être traduite sous la forme d'un ensemble suffisant d'annotations de programmes. Cette traduction consiste à décrire : (1) les états de B par des variables e_i ; (2) le franchissement des transitions de B par des affectations de ces variables e_i ; (3) la condition de synchronisation de B par un invariant [1].

Cette approche est pertinente dans le cadre d'une vérification à l'exécution [1]. En revanche, son adaptation à la vérification statique par la preuve nécessite d'ajouter une description du contexte d'exécution des opérations. Les obligations de preuve engendrées ont la structure générique suivante :

H_1	*/* Le système est synchronisé avec au moins un état de l'automate */*
$H_{2 \dots n}$	*/* Mise à jour des états de l'automate */*
$H_{n+1 \dots m}$	*/* Mise à jour des variables du programme par la fonction C */*
But	*/* Le système est synchronisé avec au moins un état de l'automate */*

Cette structure rend difficile la preuve automatique. En effet : (i) les $H_{n+1 \dots m}$ sont difficilement exploitables, car aucune hypothèse ne fait le lien entre les variables du programme et les états de l'automate ; (ii) l'hypothèse H_1 est trop faible, car des états inatteignables par certains appels d'opérations peuvent engendrés des obligations de preuve insatisfaisables ; (iii) la condition de synchronisation étant une disjonction d'états, les annotations générées explosent en nombre et en complexité avec la taille de l'automate.

Pour pallier a ces problèmes, l'utilisateur doit ajouter manuellement les assertions supplémentaires nécessaires à la preuve et guider la preuve de façon interactive. Ces tâches lourdes nécessitent une forte expertise et ne permettent pas d'utiliser la méthode sur des programmes réalistes.

*Ce travail a été réalisé au sein de l'équipe ProVal de l'INRIA Saclay – Île-de-France et financé par le projet PFC (Plate-Forme de Confiance) du pôle de compétitivité parisien System@tic.

L'outil *Jack* [2] propose une solution en deux parties pour ce type de problèmes : (1) traduction d'un automate étiqueté en annotations ; (2) propagation des pré et post-conditions des opérations. Cependant, la séparation de ces deux parties et l'abstraction faite dans la seconde peuvent pénaliser les résultats obtenus.

Contribution Le but de nos travaux est de proposer différentes optimisations du processus de génération d'annotations, afin de faciliter la résolution automatique des annotations générées. De plus, nous introduisons un début de prise en charge de la vivacité, mais cette dernière n'est pour l'instant considérée que dans des traces d'exécution de programme de longueur finie.

Nous proposons un nouvel algorithme de génération d'annotations prenant en compte, dans la traduction, la structure de l'automate et celle du programme. Cet algorithme réalise les actions suivantes :

1. Il introduit une description axiomatisée de l'automate de Büchi sous forme de constantes et d'invariants. Cela permet, par le biais des conditions de franchissement des transitions, de générer des hypothèses faisant le lien entre les états du programme et ceux de l'automate, répondant ainsi au point (i).
2. Il décrit la condition de synchronisation de l'automate par des pré et post-conditions spécifiques à chaque opération. Cela peut permettre d'être plus fin qu'avec un invariant, répondant ainsi au point (ii).
3. Pour exploiter le point précédent, une phase d'interprétation abstraite est réalisée sur le flot de contrôle programme, en intégrant les contraintes calculables statiquement (exemple : la pré-condition de main doit être synchronisée avec un état initial). Cela permet de simplifier le nombre et la complexité des annotations générées, répondant ainsi au point (iii).

Cet algorithme a été implanté dans le plug-in *Aoraï* de l'outil *Frama-C* et a été validé sur divers exemples. A terme, notre objectif est d'une part d'étendre le traitement de la vivacité à des traces d'exécution infinies et d'autre part d'affiner les abstractions faites afin de faire tourner l'outil sur des exemples réels fournis par nos partenaires au sein du projet PFC.

Références

[1] J. Groslambert. *Vérification de propriétés temporelles par génération d'annotations.* PhD thesis, Université de Franche-Comté, 2007.

[2] M. Pavlova, G. Barthe, L. Burdy, M. Huisman, and J-L. Lanet. Enforcing high-level security properties for applets. In *CARDIS'04*, pages 1–16.

[3] K. Trentelman and M. Huisman. Extending JML Specifications with Temporal Logic. In *AMAST'02*, number 2422 in LNCS, pages 334–348.