La logique contre les fantômes: comparaison de deux approches pour la preuve d'un module de listes chaînées *

Allan Blanchard, Nikolai Kosmatov, Frédéric Loulergue

La logique contre les fantômes: comparaison de deux approches pour la preuve d'un module de listes chaînées*

Allan Blanchard[†1], Nikolai Kosmatov[‡2], and Frédéric Loulergue[§3]

[1]Inria Lille – Nord Europe, Villeneuve d'Ascq, France
[2]CEA, List, Software Reliability Lab, 91191 Gif-suf-Yvette, France
[3]School of Informatics, Computing, and Cyber Systems, Northern Arizona University, Flagstaff, USA

Contexte et motivation. Les projets de vérification récents continuent d'offrir de nouveaux défis pour la vérification formelle. L'un d'eux est le module de listes chaînées de Contiki, un système d'exploitation pour l'*internet-des-objets*. Son API est riche, elle permet l'ajout et la suppression à n'importe quelle position dans une liste donnée. Par ailleurs, elle assure l'unicité des éléments lors de l'insertion (en assurant la suppression préalable de l'élément ajouté s'il était déjà présent dans la liste). Finalement, contrairement aux listes chaînées classiques, le module de Contiki ne produit pas d'allocation dynamique car le système d'exploitation ne le permet pas. Le module de listes chaînées est utilisé dans de nombreux autres modules du système, il est donc critique d'un point de vue sûreté et sécurité.

Dans un travail précédent [2], nous avons vérifié ce module avec l'outil FRAMA-C et son greffon WP. Ce travail reposait sur l'usage de tableaux fantômes (*ghost*) comme représentation des listes chaînées, et nous avait permis de prouver toutes les fonctions sauf l'insertion. Cependant, cette approche a ses inconvénients. D'abord, elle nécessite un travail d'annotation conséquent : il faut spécifier de nombreuses propriétés de séparation et écrire beaucoup d'assertions pour guider les prouveurs automatiques. Ceux-ci voient alors leur efficacité réduite car ces assertions augmentent la taille du contexte de preuve, qui devient difficile à manipuler. Vu le nombre important d'assertions que nous avions dû écrire pour prouver des propriétés parfois assez évidentes, la fonction d'insertion nous paraissait difficile à vérifier par cette approche du fait de ses nombreux et complexes comportements.

*Cette soumission est un résume étendu de l'article [1] accepté à SAC-SVT 2019.

[†]allan.blanchard@inria.fr
[‡]nikolai.kosmatov@cea.fr
[§]frederic.loulergue@nau.edu

Finalement, d'un point de vue méthodologique, une vue plus abstraite des listes que celle fournie par un tableau peut sembler préférable.

Méthode de vérification. L'article [1] présente une nouvelle vérification de ce module, réalisée avec les mêmes outils mais reposant cette fois sur l'usage de *listes logiques* fournies par ACSL, le langage de spécification de FRAMA-C. La nouvelle vérification est réalisée par le maintien d'une propriété d'équivalence entre la liste chaînée C et la liste logique qui la représente. Cette liste logique contient les adresses des différents éléments de la liste chaînée. Cette relation d'équivalence est définie à l'aide d'un prédicat inductif. Comme dans le précédent travail, nous avons écrit et prouvé 12 fonctions utilisatrices (et 12 variantes incorrectes), pour montrer que la spécification est utilisable pour prouver du code client du module (resp., que l'on ne peut pas prouver du code incorrect).

Manipuler des propriétés de listes logiques nécessite de raisonner par induction, une tâche pour laquelle les prouveurs automatiques ne sont pas bons. Dans un tel cas, l'approche classique consiste à ajouter un ensemble de lemmes exprimant des propriétés utilisant ce prédicat inductif, et qui peuvent être directement manipulées par les prouveurs automatiques. La preuve de ces lemmes est faite par induction en utilisant des prouveurs interactifs. Dans le cas des listes, les lemmes permettent par exemple d'exprimer qu'une liste peut être découpée en deux sous-listes ou inversement que deux sous-listes peuvent être fusionnées si elles se suivent.

Nous utilisons le prédicat d'équivalence pour spécifier les différentes fonctions de l'API. Chaque fonction met en relation la représentation logique de la liste chaînée en pré-condition avec la représentation obtenue en post-condition. Cependant, comme le langage ACSL ne permet pas de quantifier universellement ou existentiellement une variable sur l'ensemble d'un contrat, nous construisons la liste logique à chaque état de la mémoire (pré et post-conditions) à l'aide d'une fonction logique définie axiomatiquement. Une partie des lemmes qui étaient exprimés à propos de l'équivalence doivent être dupliqués pour pouvoir exprimer des propriétés similaires sur la construction de la liste logique. Cependant, cela permet de simplifier l'écriture des spécifications et d'éviter la présence de variables quantifiées existentiellement qui rendraient l'usage des contrats plus difficile.

Comparaison de deux approches. Nous comparons la précédente [2] et la nouvelle technique [1] pour le module de listes pour déterminer les avantages et inconvénients de chacune. Plus précisément, nous comparons le nombre d'obligations de preuve, le travail d'annotation et le temps nécessaire pour exécuter les preuves sur l'ensemble de fonctions qui avaient été prouvées dans [2]. Notons que quelques modifications ont été faites sur le travail original pour considérer la même version de FRAMA-C et des prouveurs automatiques (ces modifications ont amélioré les résultats du travail original).

Tout d'abord la taille des contrats dans la version avec listes logiques est de 42% inférieur (de 500 à 290 lignes) et le nombre d'obligations de preuve associées est diminué de 45% (de 274 à 152). La raison principale est le fait que la version avec listes logiques ne nécessite d'exprimer que très peu de propriétés de séparation,

à la différence de la version avec tableaux fantômes. La taille des annotations utilisées pour guider les prouveurs a également diminué de 33% (de 680 à 460 lignes), de même pour le nombre d'obligations (de 399 à 264). Le temps nécessaire à l'exécution complète des preuves a été divisé par 4 dans l'ensemble (de 21min 20s à 5min 30s), et divisé par 2 si l'on s'intéresse au temps moyen par obligation (de 1.6s à 0.7s). Cela rend la preuve des fonctions plus efficace : l'ingénieur doit attendre moins de temps pour avoir des résultats entre chaque tentative. En revanche, la version avec listes logiques nécessite plus de lemmes prouvés interactivement (33 contre 24 dans la version avec tableaux fantômes), et les preuves de ces lemmes sont globalement plus complexes et plus longues.

Même si nous pouvons utiliser la spécification pour prouver du code client, un utilisateur pourrait vouloir se tourner vers la vérification à l'exécution, et avoir besoin de spécifications exécutables, que le plugin E-ACSL de FRAMA-C peut transformer en code C. Nous avons pu montrer que l'approche avec les tableaux fantômes est compatible avec la vérification à l'exécution [3]. Pour l'approche avec listes logiques, ce n'est pas aussi clair, car E-ACSL ne supporte pas ce type.

La fonction d'insertion a été prouvée à l'aide de l'approche par listes logiques. À elle seule, la vérification de cette fonction représente le tiers des spécifications et des obligations de preuve du module (626 lignes, pour 259 annotations). C'est également la seule fonction ayant nécessité l'usage de COQ pour deux assertions.

Perspectives. Dans le futur, nous prévoyons de comparer ces deux versions avec une troisième version reposant sur l'usage d'une fonction d'observation. Dans le cas des listes, cela donne une fonction qui a une liste chaînée et un indice renvoie l'élément à l'indice correspondant dans la liste. La spécification du comportement des fonctions est alors faite en reliant les anciens indices des éléments aux nouveaux.

Remerciements. Ce travail a été partiellement soutenu par le CPER DATA et le projet VESSEDIA, financé par le programme européen pour la recherche et l'innovation Horizon 2020 selon la convention de subvention No 731453. Les auteurs remercient également l'équipe FRAMA-C pour les outils et le support, ainsi que Patrick Baudin, François Bobot et Loïc Correnson pour nos discussions et leurs conseils.

Références

[1] A. Blanchard, N. Kosmatov, and F. Loulergue. Logic against Ghosts : Comparison of Two Proof Approaches for a List Module. In *34^{th} Annual ACM Symposium on Applied Computing - Software Verification Track (SAC-SVT)*, Limassol, Cyprus, April 2019. ACM. Best "Software Development" Paper Award.

[2] A. Blanchard, N. Kosmatov, and F. Loulergue. Ghosts for Lists :A Critical Module of Contiki verified in FRAMA-C. In *NASA Formal Methods Symposium (NFM)*, LNCS, Newport News, VA, USA, April 2018. Springer.

[3] F. Loulergue, A. Blanchard, and N. Kosmatov. Ghosts for Lists : from Axiomatic to Executable Specifications. In *12th International Conference on Tests & Proofs (TAP)*, LNCS, Toulouse, France, June 2018. Springer.