Vérification d'une bibliothèque mathématique d'un autopilote avec Frama-C

Baptiste Pollien
baptiste.pollien@isae-supaero.fr

AFADL 2021 - Session doctorants

Résumé

Lors du développement de système critiques, comme par exemple un autopilote de drone, il est essentiel de s'assurer que le programme est sûr, en utilisant par exemple des méthodes formelles. Pour faciliter la vérification, on se restreint généralement à une abstraction du système ou un sous-ensemble. Cet article présente la vérification d'une bibliothèque mathématique de l'autopilote Paparazzi, à l'aide de l'outil Frama-C, afin de garantir l'absence d'erreur à l'exécution et certaines propriétés fonctionnelles.
Mots clés : Preuve de programme, méthodes déductives, interprétation abstraite

1 Introduction

Les méthodes formelles sont des techniques de vérification basées sur des modèles mathématiques qui permettent de vérifier et de garantir certaines propriétés. Il existe de nombreux outils de vérification formelle qui possèdent des spécificités en termes des propriétés vérifiables et qui nécessitent des efforts plus ou moins importants de spécification ou de modélisation du système afin de les mettre en œuvre.

L'objectif de ma thèse est de procéder à une revue des différentes techniques de vérification formelle afin de définir un processus d'analyse d'une architecture d'autopilote de drone tirant parti de ces techniques. Ce processus d'analyse est appliqué au cas de l'autopilote Paparazzi développé à l'ENAC en langage C.

Frama-C [3] est un outil d'analyse de code C qui nécessite l'ajout d'annotations dans le code pour spécifier les propriétés attendues : définition de contrats pour les fonctions (*préconditions*, *postconditions* et pour spécifier l'ensemble des éléments mémoire (hors pile) qui seront modifiés lors de l'exécution de la fonction), définition des *invariants* et *variants* de boucle pour vérifier la terminaison et l'ajout d'assertion. Frama-C dispose de nombreux greffons dont trois nous intéressent pour vérifier formellement les annotations : WP (*Weakest Precondition* qui utilise un calcul de plus faible préconditions), RTE (*RunTime Errors*) pour l'ajout automatique

d'assertions afin de vérifier l'absence d'erreur à l'exécution et EVA (*Evolved Value Analysis*) utilisant des techniques d'analyse statique par interprétation abstraite.

Dans le cadre de cet article, la vérification a été effectuée avec la plateforme Frama-C en n'utilisant que des démonstrateurs automatiques et s'est limitée à une bibliothèque mathématique de Paparazzi présentée en section 2. La section 3 présente la première partie de l'analyse concernant la vérification de l'absence d'erreur à l'exécution. La seconde partie, présentée en section 4, présente la vérification de propriétés fonctionnelles pour certaines fonctions mathématiques en ne considérant pas les erreurs d'arrondies potentielles liées aux calculs sur les nombres à virgule flottante.

2 Paparazzi

Paparazzi [2] est un autopilote open-source (sous licence GPL) développé à l'ENAC depuis 2003. Il supporte differents types de drones et permet le contrôle simultané de plusieurs drones. Paparazzi possède également différents modes intégrés et offre la possibilité de créer des plans de vol personnalisés. La bibliothèque mathématique étudiée correspond à la conversion de rotation de vecteurs dans différentes représentations (matrices de rotation, angles d'Euler, quaternions). Elle définit également certaines opérations élémentaires sur ces représentations. Elle est composée d'environ 4000 lignes de code C et chaque fonction dispose d'une version travaillant sur les types `int`, `float` et `double`. Cette bibliothèque est principalement utilisée pour faire le lien entre les capteurs et la partie contrôle de l'autopilote qui n'utilisent pas nécessairement les mêmes représentations. La vérification de cette bibliothèque permet donc de garantir la cohérence des données traitées malgré l'utilisation de représentations différentes.

3 Absence d'erreurs à l'exécution

La bibliothèque définit des structures C pour représenter les données manipulées (matrices de rotation, quaternions, vecteurs...). Les fonctions de la bibliothèque travaillent uniquement par référence pour les entrées et pour les sorties. Afin d'éviter des erreurs liées à un déréférencement de pointeurs non valides, des préconditions garantissant la validité des références ont été ajoutées dans les contrats des fonctions. Il a aussi été nécessaire de spécifier les variants et invariants de boucle ainsi que les variables de sorties comme seuls espaces mémoire (hors pile) qui seront modifiées.

WP dispose de différents modèles arithmétiques qui prennent en compte de façon plus ou moins précise la sémantique de C. La vérification de la bibliothèque sur les entiers a été faite en utilisant le modèle réaliste de l'arithmétique machine des entiers. Avec le greffon RTE, il est alors nécessaire de vérifier qu'il n'y a pas de débordement de valeur (ou *overflow* en anglais) pour chaque opération arithmétique. Pour vérifier l'absence de dépassement, chaque fonction a été analysée dans l'objectif de déterminer les bornes maximales possibles des différentes variables. Lorsque

ces bornes ont pu être déterminées, elles ont été ajoutées en préconditions dans les contrats des fonctions.

Malheureusement, WP associé à des prouveurs automatiques n'est pas parvenu à vérifier ces nouveaux contrats. L'utilisation de références pour l'accès aux valeurs numériques surcharge les prouveurs et ce même en spécifiant en précondition que les structures en paramètres sont stockées à des emplacements mémoire séparés.

Pour pallier ce problème, nous avons décidé d'associer EVA à WP. EVA arrive à calculer des intervalles suffisament précis des valeurs possibles pour chaque variable. Ce résultat est ensuite transmis à WP par Frama-C ce qui permet de conclure plus facilement les preuves. Cette limitation de WP avait aussi été notée par Vassil Todorov durant sa thèse [5] et il avait également utilisé un outil d'analyse statique par interprétation abstraite, Astrée, pour résoudre ce problème. En conclusion, lorsque l'on associe EVA avec WP, il est possible de vérifier l'absence d'erreur à l'exécution des fonctions de la bibliothèque sur les entiers.

WP dispose également d'un modèle arithmétique `real` qui correspond à l'arithmétique réelle au sens mathématique. Pour la vérification des versions de la bibliothèque travaillant sur des valeurs numériques à virgule flottante (aussi bien `float` ou bien `double`), nous avons décidé d'utiliser ce modèle. Il nous a permis de vérifier l'absence de division par 0 et que les variables ne prennent pas la valeur `NaN` (*Not A Number*). Pour effectuer ces vérifications, il a été seulement nécessaire d'ajouter comme préconditions que chaque valeur numérique passée en paramètre ne prend pas la valeur *NaN* et qu'elle n'est pas infinie. Là encore, l'absence de ces deux erreurs à l'exécution et la terminaison des fonctions ont été prouvées pour les versions `float` et `double` de la bibliothèque en utilisant WP et EVA. Par contre, notre vérification n'offre aucune garantie sur le risque de dépassement ou sur les erreurs liées aux arrondis. Cependant, ce modèle nous a été particulièrement utile pour la vérification des propriétés fonctionnelles présentée en section 4.

4 Vérification fonctionnelle de propriétés

La vérification fonctionnelle permet de garantir les résultats attendus d'une fonction. Dans notre cas d'étude nous avons décidé de vérifier certaines propriétés fonctionnelles de la fonction float_rmat_of_quat car elle est représentative et indépendante des autres fonctions présentes dans la bibliothèque. Cette fonction prend en paramètre un quaternion normalisé et retourne la matrice de rotation correspondante.

Afin de spécifier les propriétés fonctionnelles, des types et des prédicats ont été définis dans la logique fournie par le language ACSL. On retrouve la définition des types pour les matrices et les quaternions ainsi que des opérations élémentaires. Des lemmes ont ensuite été spécifiés et vérifiés pour garantir que ces opérations sont correctes. Une fonction logique qui convertit un quaternion en une matrice de rotation a également été définie, indépendamment du code C. Cette fonction est basée sur la formule mathématique de conversion d'un quaternion vers une matrice

de rotation [1, 4].

À partir de ces définitions, le contrat de la fonction a pu être établi. En supposant que le quaternion passé en paramètre est normalisé, nous avons voulu vérifier 2 propriétés fonctionnelles. La première est que la matrice retournée correspond bien à la conversion du quaternion passé en paramètre : notre post-condition vérifie que la matrice de rotation générée par le code C est égale à la matrice de rotation générée par notre fonction logique. Comme dans la section précédente, nous utilisons le modèle `real` de WP pour la vérification de cette fonction, ce qui permet d'ignorer les différences de résultats entre la version C et la version mathématique qui auraient pu être liées à des erreurs d'arrondis. La seconde propriété vérifie que la matrice générée est bien une matrice de rotation, i.e. que la transposée de la matrice est son inverse.

Malgré l'utilisation du modèle `real` d'arithmétique, WP n'arrivait pas à vérifier le contrat. Il a donc été nécessaire d'analyser le code. Nous avons remarqué que le code C utilisait une constante `M_SQRT2` pour représenter $\sqrt{2}$ et qu'en simplifiant les calculs, la constante `M_SQRT2` était tout le temps multipliée à elle-même. Nous avons donc suggéré une modification du code en remplaçant ces opérations par une multiplication par 2. Cette modification ne change pas le nombre de multiplications mais permet de réduire les erreurs d'arrondis propagées par la fonction. Avec ces modifications de code et avec le modèle `real` pour l'arithmétique, WP vérifie le contrat de la fonction `float_rmat_of_quat`. Ce contrat garantit bien l'absence d'erreur à l'exécution et définit les propriétés fonctionnelles attendues. Ces propriétés permettent de vérifier uniquement le comportement idéaliste de la fonction sans considérer les erreurs potentielles de calcul.

5 Conclusion

Dans cet article, nous avons pu présenter mon travail de vérification d'une bibliothèque mathématique de Paparazzi qui est disponible sur GitLab. Ce travail s'est principalement concentré sur la vérification de l'absence d'erreurs à l'exécution et a nécessité l'ajout de près de 3,5k lignes d'annotations sur le code.

Dans la continuité de ce premier travail, il est envisagé de vérifier fonctionnellement d'autres fonctions de la bibliothèque mathématique de Paparazzi et d'étudier les erreurs d'arrondis en ne se limitant plus au modèle `real` mais en utilisant un modèle représentant plus fidélement les flottants.

Remerciements

Je remercie mon directeur de thèse Xavier Thirioux (ISAE-SUPAERO) et mes co-encadrants Christophe Garion (ISAE-SUPAERO), Gautier Hattenberger (ENAC) et Pierre Roux (ONERA). Je tiens également à remercier l'Agence pour l'Innovation de Défense (AID) du Ministère des Armées pour le financement de cette thèse (projet de recherche CONCORDE N° 2019 65 0090004707501).

Références

[1] Carl Grubin. Derivation of the quaternion scheme via the euler axis and angle. *Journal of Spacecraft and Rockets*, 7(10) :1261–1263, 1970.

[2] Gautier Hattenberger, Murat Bronz, and Michel Gorraz. Using the Paparazzi UAV System for Scientific Research. In *IMAV 2014, International Micro Air Vehicle Conference and Competition 2014*, pages pp 247–252, Delft, Netherlands, August 2014.

[3] Florent Kirchner, Nikolai Kosmatov, Virgile Prevosto, Julien Signoles, and Boris Yakobowski. Frama-C : A software analysis perspective. *Formal aspects of computing*, 27 :573–609, 2015.

[4] Allan R. Klumpp. Singularity-free extraction of a quaternion from a direction-cosine matrix. *Journal of Spacecraft and Rockets*, 13(12) :754–755, 1976.

[5] Vassil Todorov. *Automotive embedded software design using formal methods*. Phd thesis, Université Paris-Saclay, December 2020.