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

AFADL 2021

B. Pollien[1], C. Garion[1], G. Hattenberger[2], P. Roux[3], X. Thirioux[1]

18 juin 2021

[1]ISAE-SUPAERO, [2]ENAC et [3]ONERA

Contexte

Les méthodes formelles

- Techniques de vérification basées sur des modèles mathématiques
- Utilisables dans l'avionique avec les normes DO-178C et DO-333
- Exemple : interprétation abstraite, méthodes déductives, model-checking

Objectifs de ma thèse

- Définir des processus de vérification avec des méthodes formelles,
- Appliquer ces méthodes à un autopilote de drone : Paparazzi.

Premiers résultats

Analyse d'une bibliothèque mathématique de Paparazzi :

- Utilisation de Frama-C,
- Vérification de l'absence d'erreurs à l'exécution,
- Vérification de certaines propriétés fonctionnelles,
- Sans modifier le code.

Frama-C

Frama-C

Frama-C est un outil d'analyse de code C

- Développé par le CEA et l'INRIA,
- Modulaire qui supporte différentes méthodes d'analyse
  ex : analyse statique avec EVA ou dynamique avec E-ACSL.

Processus de vérification d'un programme C avec Frama-C :

1. Spécification du code avec **ACSL** (*ANSI C Specification Language*),
2. Génération de l'arbre de syntaxe abstraite du code analysé,
3. Analyse de l'arbre par les plugins
   $\Longrightarrow$ vérifier si la spécification est respectée.

Remarque : l'analyse de l'arbre peut être réalisée par plusieurs plugins.

Quelques plugins Frama-C

RTE (*RunTime Errors*) :

- Ajout d'assertions dans le code,
- Permet de vérifier l'absence d'erreurs à l'exécution
  ex : division par 0, overflows...

WP (*Weakest Precondition*)

- utilise un calcul de plus faibles préconditions,
- Interfacé avec Why3 pour vérifier les buts avec des prouveurs automatiques (Alt-Ergo, Z3, CVC4).

EVA (*Evolved Value Analysis*)

- Basé sur des méthodes d'analyse statique par interprétation abstraite,
- Calcule des domaines de valeurs pour chaque variable du programme.

Paparazzi

Présentation

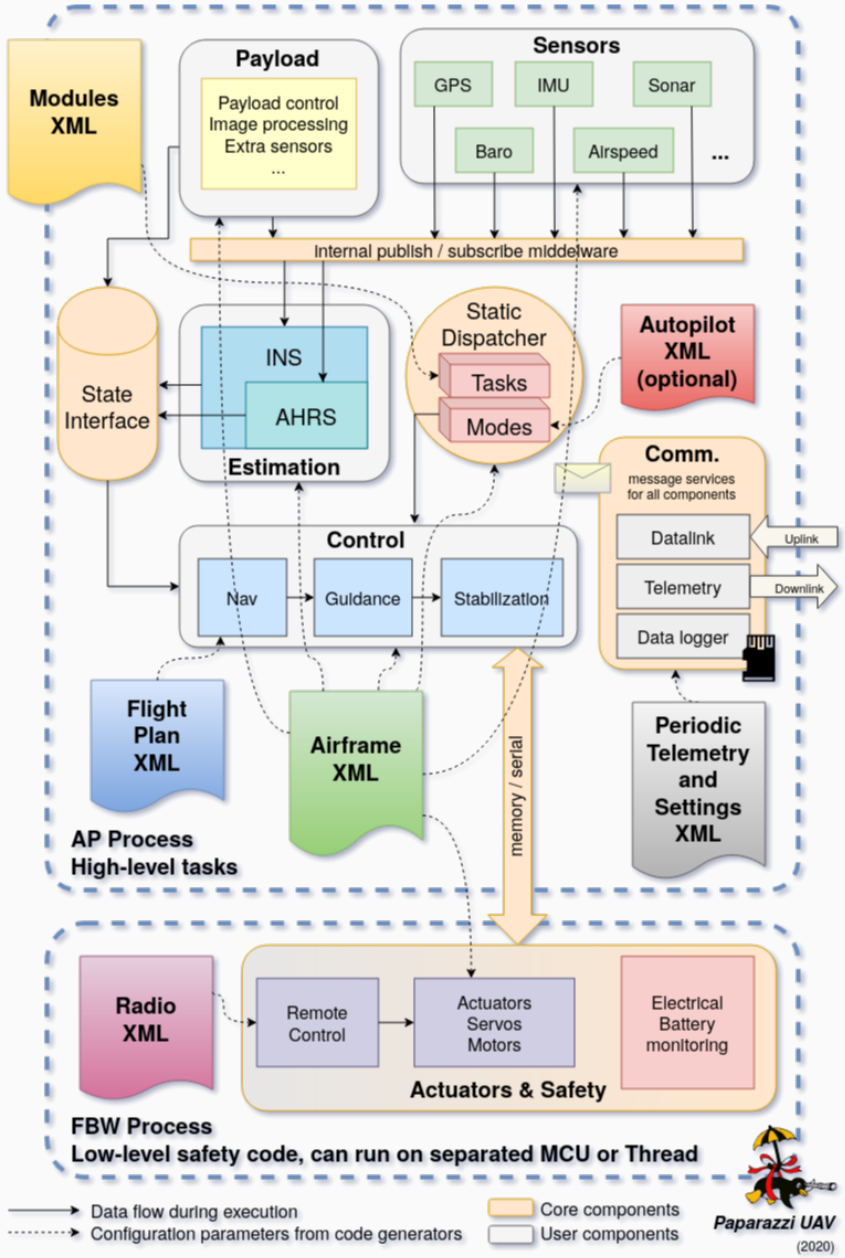
Paparazzi est un autopilote pour micro-drones

- Développé à l'ENAC depuis 2003,
- Open-Source sous licence GPL

Système complet de contrôle de drones :

- Propose la partie logiciel de contrôle,
- Propose également certains designs de composants matériels,
- Support des véhicules terrestres et aériens,
- Support du contrôle simultané de plusieurs drones.

Architecture du système de vol



Bibliothèque étudiée

`pprz_algebra` : bibliothèque mathématique d'algèbre ($\sim$ 4 000 loc)

La bibliothèque codée en C contient :

- La définition d'une représentation des `vecteurs`,
- Différentes représentations pour la rotation de vecteurs,
  matrices de rotation, angles d'Eulers, quaternions
- Des opérations élémentaires,
  ex : addition de vecteurs, calcul de la rotation d'un vecteur, normalisation d'un quaternion...
- Des fonctions de conversion entre ces différentes représentations.

Remarque : Chaque représentation/fonction dispose d'une version en virgule fixe (`int`) et en virgule flottante (pour `float` et `double`).

Absence d'erreurs à l'exécution

Absence d'erreurs à l'exécution

Il existe différents types d'erreurs à l'exécution :

- Déréférencement d'un pointeur non valide,
- Division par 0,
- *Overflows*,
- Valeur flottante non finie,
- ...

Objectif : Déterminer les contrats minimaux des fonctions de la bibliothèque afin de garantir l'absence d'erreurs à l'exécution.

Processus :

- Analyser le code avec les plugins RTE et WP de Frama-C.
- Déduire les informations manquantes dans le contrat.

Analyse avec Frama-C et le plugin RTE

Analyse de l'instruction :

```
c->x = a->x * b->x;
```

Frama-C trouve 2 erreurs potentielles !

- Les pointeurs peuvent ne pas être valides.
  - `/*@ assert rte: mem_access: \valid(&c->x); */`
  - `/*@ assert rte: mem_access: \valid_read(&a->x); */`
  - `/*@ assert rte: mem_access: \valid_read(&b->x); */`

  $\Longrightarrow$ Exiger en précondition la validité des pointeurs.
- Les valeurs ne sont pas bornées.
  - `/*@ assert rte: signed_overflow: -2147483648 ≤ a->x * b->x; */`
  - `/*@ assert rte: signed_overflow: a->x * b->x ≤ 2147483647; */`

  $\Longrightarrow$ Déterminer les valeurs qui garantissent l'absence d'*overflows*.

Calcul d'intervalle

Calcul de l'intervalle des valeurs possibles pour `a->x` et `b->x` pour garantir l'absence de RTE dans l'instruction :

```
c->x = a->x * b->x;
```

Les intervalles doivent être égaux.

On cherche à déterminer la borne maximale M telle que :

Si `a->x`$\in [-M; M]$

`b->x`$\in [-M; M]$

Alors, `c->x` $\in$ $[\texttt{INT\_MIN}; \texttt{INT\_MAX}] \subseteq [-\texttt{INT\_MAX}; \texttt{INT\_MAX}]$

On peut facilement déduire que $M := \sqrt{\texttt{INT\_MAX}}$

Exemple de contrat final pour la fonction `int32_quat_comp`

```
#define SQRT_INT_MAX4 23170 // 23170 = SQRT(INT_MAX/4)

/*@
  requires \valid(a2c);
  requires \valid_read(a2b);
  requires \valid_read(b2c);
  requires bound_Int32Quat(a2b, SQRT_INT_MAX4);
  requires bound_Int32Quat(b2c, SQRT_INT_MAX4);
  requires \separated(a2c, a2b) && \separated(a2c, b2c);
  assigns *a2c;
*/
void int32_quat_comp(struct Int32Quat *a2c,
                     struct Int32Quat *a2b,
                     struct Int32Quat *b2c)
```

Remarques

EVA et WP ont dû être associés pour vérifier l'absence de RTE.

- WP est surchargé lors des accès aux valeurs par référence,
- EVA n'arrive pas vérifier les variants/invariants de boucles.

$\Longrightarrow$ Problème identique soulevé dans la thèse de V. Todorov.

Utilisation du **modèle arithmétique** `real` (réels au sens mathématique) pour la vérification des versions des fonctions à virgule flottante.

Le modèle `real` permet de garantir :

- L'absence de division par 0,
- L'absence de déréférencement de pointeurs non valides.

Mais l'absence d'overflows n'est pas vérifiée.

Vérification fonctionnelle

Vérification fonctionnelle

Offrir des garanties sur le comportement ou le résultat d'une fonction.

Exemple : *Propriétés fonctionnelles pour la fonction racine carrée*

```
/*@
  requires x >= 0;
  ensures \result >= 0;
  ensures \result * \result == \old(x);
  assigns \nothing;
*/
float sqrt(float x);
```

Remarque : La vérification de ces propriétés est seulement possible avec le modèle *real*.

La vérification fonctionnelle a seulement été réalisée sur certaines fonctions flottantes.

Spécification des propriétés fonctionnelles de la bibliothèque

Les propriétés fonctionnelles doivent être exprimées dans la logique d'ACSL.

Dans un premier temps, il est nécessaire de définir :

- **Les types**,
  ex : `RealVect3`, `RealRMat`, `RealQuat`.
- **Des fonctions élémentaires**,
  ex : addition de vecteurs, rotation d'un vecteur...
- **Des fonctions de conversion** entre certaines représentations,
  ex : Conversion d'un quaternion vers une matrice de rotation.
- **Des lemmes**.
  ex : La fonction de conversion produit la même rotation, ...

Enfin, les propriétés fonctionnelles sont exprimées sous forme de prédicats :

- M est une matrice de rotation : $M.M^t = I$
- ...

Exemple

Spécification de la fonction `float_rmat_of_quat`.

```
/*@
    requires \valid(rm);
    requires \valid_read(q) && finite_FloatQuat(q);
    requires unitary_quaternion(q);
    requires \separated(rm, q);
    ensures rotation_matrix(l_RMat_of_FloatRMat(rm));
    ensures special_orthogonal(l_RMat_of_FloatRMat(rm));
    ensures l_RMat_of_FloatRMat(rm) ==  l_RMat_of_FloatQuat(q);
    assigns *rm;
*/
void float_rmat_of_quat(struct FloatRMat *rm,
                        struct FloatQuat *q)
```

Erreurs de calcul dans la fonction `float_rmat_of_quat`.

Problème :

WP et EVA n'arrivaient pas à vérifier certaines propriétés.

Après analyse du code :

- Les fonctions définies et le code C ne produisent pas toujours le même résultat, malgré l'utilisation du modèle *real*.
- Utilisation d'une constante `M_SQRT2` pour représenter $\sqrt{2}$,
- Lorsque les calculs sont dépliés, on obtient toujours :

$$\texttt{M\_SQRT2} * \texttt{M\_SQRT2} \neq 2$$

$\Longrightarrow$ **Le code produit donc toujours des erreurs de calcul.**

Modification de code proposée

Code original :

```
#define M_SQRT2          1.41421356237309504880
const float _a = M_SQRT2 * q->qi, _b = M_SQRT2 * q->qx;
(...)
const float a2_1 = _a * _a - 1;
(...)
RMAT_ELMT(*rm, 0, 0) = a2_1 + _b * _b;
(...)
```

Code modifié :

```
const float _a  = q->qi , _b  = q->qx ;
const float _2a = 2 * _a, _2b = 2 * _b;
(...)
const float a2_1 = _2a * _a - 1;
(...)
RMAT_ELMT(*rm, 0, 0) = a2_1 + _2b * _b;
(...)
```

Résumé de la vérification fonctionnelle

La vérification fonctionnelle offre des garanties sur le comportement d'une fonction.

La modification de code proposée pour la fonction `rmat_of_quat` :

- Réduit les erreurs de calcul à l'exécution,
- Ne modifie pas le nombre d'opérations,
- Facilite la vérification du contrat.

Utilisation du modèle `real` :

- Permet de vérifier que le code est correct au sens mathématique,
- N'offre aucune garantie fonctionnelle lors de l'exécution.

Conclusion

Conclusion

Résumé :

- Vérification de l'absence d'erreurs à l'exécution de la bibliothèque,
- Vérification de propriétés fonctionnelles sur certaines fonctions flottantes.

$\Longrightarrow$ Environ 3 500 lignes d'annotation.

`gitlab.isae-supaero.fr/b.pollien/paparazzi-frama-c`

Perspectives :

- Vérification des appels aux fonctions de la bibliothèque,
- Vérification de la bibliothèque flottante sans le modèle `real`,
- Vérification du générateur de plan de vol de Paparazzi.

Projet de recherche CONCORDE N° 2019 65 0090004707501 financé par l'Agence pour l'Innovation de Défense (AID) du Ministère des Armées.

Merci de votre attention

References i


S. Sarabandi and F. Thomas.
Accurate computation of quaternions from rotation matrices.
In J. Lenarcic and V. Parenti-Castelli, editors, *Advances in Robot Kinematics 2018*, pages 39–46, Cham, 2019. Springer International Publishing.


S. W. Shepperd.
Quaternion from rotation matrix.
Journal of Guidance and Control, 1(3) :223–224, 1978.


V. Todorov.
Automotive embedded software design using formal methods.
Phd thesis, Université Paris-Saclay, Dec. 2020.