

Apport des méthodes formelles

David Monniaux

CNRS / VERIMAG

Persyval / Minalogic, 6 février 2018

David Monniaux (CNRS / VERIMAG)

Apport des méthodes formelles



Minalogic 2018-02-06

1 / 29

analyse statique

Plan

Analyse statique

Preuve assistée

Recherche automatique de bugs

Compilation certifiée

David Monniaux (CNRS / VERIMAG)

Apport des méthodes formelles



Minalogic 2018-02-06

2 / 29

Analyse statique

Obtenir des informations sur le programme **sans l'exécuter**.

Se distingue du **test** ou analyse **dynamique**.

À distinguer

Critères syntaxiques

- ▶ Liste de cas suspects à vérifier
- ▶ Constructions interdites
- ▶ Critères numériques (taux de commentaires, nombre cyclomatique...)

Critères « sémantiques »

- ▶ Construit un modèle mathématique du programme
- ▶ Prouve des propriétés dessus

Cas intermédiaires

Pas vraiment de preuve mais moins restreint que pleinement syntaxique.

Misra-C

Norme pour les logiciels de l'industrie automobile.

Critères syntaxiques

certaines constructions interdites (réutilisations d'identificateurs)
 ⇒ vérification facile, outils disponibles

Critères « sémantiques »

- ▶ « comportements indéfinis interdits »
 - ▶ dépassement arithmétique
 - ▶ accès hors des bornes
 - ▶ pointeurs incorrects...
- ▶ violations d'assertions

Mélangés dans MISRA-2004, distingués dans MISRA-2012

Raison de la difficulté

Alan Turing



Pas de méthode d'analyse statique

- ▶ **automatique**
- ▶ qui répond **toujours** et **sans mentir**
- ▶ détecte toutes les violations sur le résultat final du programme
- ▶ sans imposer de bornes arbitraires de profondeur d'analyse

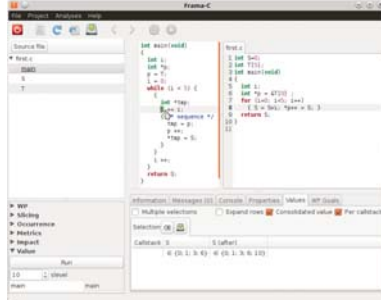
e.g. pas de méthode qui dit si le programme peut planter avec une erreur

Outils d'analyse statique « sémantique »

ex: **Astrée** (ENS/CNRS), **Frama-C Value** (CEA), **Polyspace** (Mathworks, Montbonnot) mais aussi **Infer** (Facebook)

Construisent un ensemble de valeurs d'où le programme ne peut pas s'échapper (« invariant inductif »).

p.ex. intervalle de variation pour chaque variable



<http://www.astree.ens.fr/>

<https://www.absint.com/astree/index.htm>

<https://frama-c.com/value.html>

David Monniaux (CNRS / VERIMAG)

Apport des méthodes formelles



Minalogic 2018-02-06

7 / 29

Analyses automatiques

Conditions d'emploi

- ▶ Cherchent des arguments de preuves (invariants inductifs) dans une certaine classe.
- ▶ Adaptées à certaines classes de programmes (p.ex. Astrée pour codes de contrôle / commande, automatique).
- ▶ Échoueront sur codes trop éloignés de leur cible.

Sortie

Une liste de violations possibles (violations d'assertions, erreurs à l'exécution) avec indication

Rouge chaque passage ici provoque une erreur

Orange possibilité d'erreur ici (« ne sait pas »)

Vert jamais d'erreur ici

Gris code garanti mort



Exemple d'erreur

```

int t[n];
...
if (t[i] >= 5 && i < n) {
    ...
}

```

Exemple d'erreur

```

int t[n];
...
if (t[i] >= 5 && i < n) {
    ...
}

```

Si $i \geq n$ erreur possible.


- ▶ Détection statique par recherche de motif
- ▶ Analyse statique sûre qui trouvera
 - ▶ soit du code mort dans le test ($i \geq n$ jamais pris)
 - ▶ soit un « orange »

WCET

Analyse statique pour borner le plus grand temps d'exécution.
Difficile sur des processeurs modernes

- ▶ caches
- ▶ pipelines

Outils aIT (<https://www.absint.com/ait/index.htm>),
Ottawa (<http://ottawa.fr/>)

Travaux à VERIMAG pour augmenter la précision des analyses!
e.g. collaboration avec Continental 

Model checking

Analyse statique pour **systèmes à état fini**

- ▶ protocoles
- ▶ matériel

Énumération **exhaustive** des cas.

e.g. outils NuSMV, NuXMV, CADP

Expertise à VERIMAG / PACSS et INRIA / CONVECS

Message à retenir

Analyse statique automatique adaptée à la **classe de programmes** et la **classe de propriétés**.

Ne pas confondre propriétés syntaxiques (faciles) et propriétés sémantiques (difficiles).

Grande variété de techniques.

Choix :

- ▶ correction (pas de faux négatifs) au prix de l'imprécision (faux positifs) et de l'inefficacité ?
- ▶ ou faux négatifs mais moins de faux positifs / plus efficace ?

Temps indicatifs : Astrée

Développement des outils

- ▶ Prototype pour propriété syntaxique sur C : très rapide, 1 mois ?
- ▶ Prototype pour analyse sémantique simple : 6 mois (2 développeurs)
- ▶ Analyse sémantique complexe, développements scientifiques : 2 ans (5 développeurs) puis encore des années (améliorations, extensions)
- ▶ Interface, générateur de rapports, etc. : transférer !

Temps de calcul

- ▶ Commande de vol Airbus complète à analyser (300 kLOC) : une nuit de calcul, 0 ou quasi 0 faux positifs



- ▶ Mais programme simple !

Plan

Analyse statique

Preuve assistée

Recherche automatique de bugs

Compilation certifiée

David Monniaux (CNRS / VERIMAG)

Apport des méthodes formelles



Motivation

Analyse statique automatique : limitée par ce qu'elle sait trouver seule
comme invariants

Preuve assistée : **l'utilisateur fournit ses invariants**
L'outil **vérifie les invariants**

Nécessite des utilisateurs plus experts
Formation par des enseignants-chercheurs du domaine

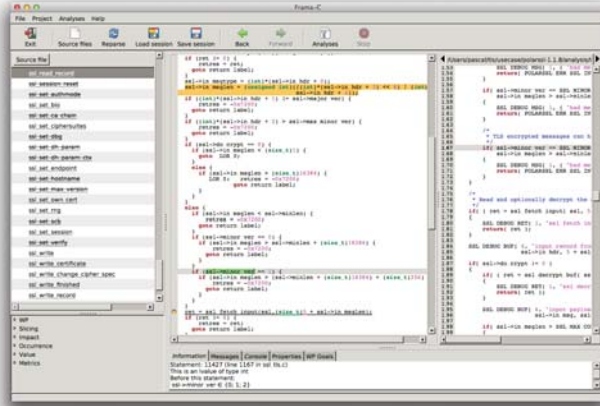
David Monniaux (CNRS / VERIMAG)

Apport des méthodes formelles



Frama-C WP

<https://frama-c.com/>



David Monniaux (CNRS / VERIMAG)

Apport des méthodes formelles

Exemple

```
for(int i=0; i<n; i++) t[i]=0;
```

L'utilisateur dit que l'invariant c'est

- ▶ $0 \leq i \leq n$
- ▶ les cases d'indice $< i$ sont à 0

L'outil vérifie que cela passe bien à la récurrence.

David Monniaux (CNRS / VERIMAG)

Apport des méthodes formelles

Plan

Analyse statique

Preuve assistée

Recherche automatique de bugs

Compilation certifiée

Recherche de bugs

Amélioration du test à l'aide de techniques du formel.

- ▶ *Bounded model checking* : recherche **exhaustive** de bugs à une certaine profondeur (p.ex. CBMC, <http://www.cprover.org/cbmc/>)
- ▶ **Exécution symbolique**, ou « concolique » : on simule les chemins de programme (p.ex. Klee, <https://klee.github.io/>)
- ▶ **Fuzzing** avancé : exécution concolique spéciale à la recherche de trous de sécurité (p.ex. Microsoft SAGE)

Ciblage de la recherche

```
void f(int x, int y) {
    int z = 2*y;
    if (x >= 40000 && x <= 40042) {
        assert (x >= z);
    }
}
```

Difficile à trouver par test aléatoire

Bounded model checking et exécution symbolique résolvent

$$z = 2y \wedge x \geq 40000 \wedge x \leq 40042 \wedge x < z \quad (1)$$

Satisfiabilité modulo théorie

Analyse statique, preuve assistée et exécution symbolique ramènent beaucoup de problème à résoudre des formules.

Efficacité sur de grosses formules ?

Expertise sur le sujet à VERIMAG (PACSS) et LIG (CAPP)

- ▶ Transformer un problème de recherche de bug, d'optimisation etc. en une formule
- ▶ Résoudre la formule efficacement

Plan

Analyse statique

Preuve assistée

Recherche automatique de bugs

Compilation certifiée

David Monniaux (CNRS / VERIMAG)

Apport des méthodes formelles



Compilation

Simulink / Scade \longrightarrow C \longrightarrow assembleur

Comment être sûr que les compilateurs préservent le sens du programme ?

Qu'ils n'introduisent pas de bugs ?

David Monniaux (CNRS / VERIMAG)

Apport des méthodes formelles



Probabilité accrue de bugs

(Regehr, *Is That a Compiler Bug?*)

Ce qui est moins testé par moins de gens est probablement plus buggué.

- ▶ Options d'optimisation etc. inhabituelles.
- ▶ Code inhabituel (génééré automatiquement, etc.).
- ▶ Processeur cible peu courant (trop nouveau, trop ancien, etc.)

Cas courants pour des machines embarquées !

(Moins pour des applications x86 pour Linux ou Windows !)

Que faire ?

- ▶ Utiliser plate-forme courante (x86), compilateur courant (gcc / MSVC), options par défaut ?
- ▶ Désactiver les optimisations ?
- ▶ Relire code source vs code assembleur non optimisé ?

⇒ Pas toujours possible !

⇒ Coûteux en efficacité !

CompCert

Compilateur C vers diverses architectures.

- ▶ Chaque code intermédiaire a un sens mathématiquement défini.
- ▶ Preuves que les traductions respectent ce sens.
- ▶ Preuves vérifiées avec Coq.

⇒ le code objet a le même sens que le code source.

<http://compcert.inria.fr/>

<https://www.absint.com/compcert/index.htm>

CompCert chez Airbus

(J. Souyris, *Industrial Use of CompCert on a Safety-Critical Software Product*)



WCET

Réduction de la borne de WCET (pire temps d'exécution) par rapport à un compilateur non optimisant.

Préservation du sens

Par rapport à des comparaisons objet/source.

CompCert à VERIMAG

Ajout de nouvelles architectures à CompCert

Kalray



Risc-V

Pour un dérivé du Risc-V

Questions ?

<https://www-verimag.imag.fr/~monniaux/>

Venez discuter de **vos** problèmes !