“Peut-on prouver un code ordinaire?”

Hadrien Grasland & Xavier Grave

LAL & CSNSM – Orsay

Prouver quoi?

- On ne peut pas tout prouver
  - Des obstacles fondamentaux: Problème de l'arrêt, E/S...
  - Tout repose sur des infos données par le programmeur
- On peut prouver l'absence d'erreurs courantes :
  - Paramètres/résultats non conformes à une spec
  - Non-respect des invariants d'une classe/boucle
  - Exceptions à l'exécution
  - Erreurs mémoire (uninitialized, overflow, use-after-free...)
  - *Data races* (un thread écrit alors qu'un autre accède)
  - Et bien d'autres...

Preuve et test

- Le test est plus général et flexible, plus bas niveau
  - Les outils de preuve ne rendent pas le test obsolète !
- Un test exhaustif est impossible
  - Fonction prenant un entier en paramètres = 2^{32} possibilités
- Les résultats du test dépendent beaucoup du testeur
  - Boîte noire: Tâtonner au hasard en espérant bien tomber
  - Boîte blanche: Grande influence des biais psychologiques
- Une couverture de tests de 100% ne suffit pas !
  - Exceptions, vtables & généricité, overflow, sécurité, threads...

Préparation (dépend de l'outil)

- Clarifier la spécification du programme
  - Contrats d'entrée/sortie des fonctions
  - Invariants de classe/type/boucle
  - Attentes sur les E/S (vérifiables à l'exécution)
- Réduire la combinatoire
  - Typage fort & early binding
  - Pas d'allocation mémoire dynamique
  - Pas de pointeurs arbitraires (null, aliasing, arithmétique...)
  - Programmes bien structurés (pas de goto, try/catch…)
- Bien séparer le code prouvable du non-prouvable !

Rôle du langage

- Deux possibilités pour se placer dans le cadre précédent
  - Modifier un langage existant (extensions + restrictions)
  - Utiliser un langage pensé pour la preuve
- Nos exemples
  - SPARK 2014 : Prouver des contrats d'interface en Ada
  - Rust : Prouver l'absence d'erreurs mémoire