
# Logiques de programmes: quand la machine raisonne sur ses logiciels

Introduction

Xavier Leroy

2021-03-04

Collège de France, chaire de sciences du logiciel
xavier.leroy@college-de-france.fr

# Comment s’assurer qu’un logiciel fait ce qu’il est censé faire ?

# Les méthodes classiques

## Test

- Exécuter le programme sur des entrées bien choisies.
- Comparer les comportements observés aux comportements attendus.

## Revues

- Relire attentivement le code, les tests, les documents de conception, …

## Analyses

- Étudier mathématiquement certains aspects du programme : précision numérique, complexité en temps ou en espace, etc.
- Sur le papier ou assisté par des outils d'analyse statique.

# Les limites du test

*Testing shows the presence, not the absence of bugs.*

(E. W. Dijkstra, 1969)

One ne teste qu'un petit nombre des comportements possibles du programme. Certains bugs se déclenchent très rarement !

**Exemple (propagation de retenue dans les codes crypto)**

Ajoute `2 * ta * tb` à `c2:c1:c0` en «optimisant» les retenues.

```
BN_UMULT_LOHI(t0,t1,ta,tb);
t2 = t1+t1; c2 += (t2<t1)?1:0;
t1 = t0+t0; t2 += (t1<t0)?1:0;
c0 += t1; t2 += (c0<t1)?1:0;
c1 += t2; c2 += (c1<t2)?1:0;
```

# Les limites des revues de code

*Given enough eyeballs, all bugs are shallow.*

(Eric Raymond, 1999)

Fatigue, étourderie, distraction des relecteurs.

Certains codes sont moins relus que d'autres (p.ex. les *hot fixes*).

**Exemple (le bug `goto fail`)**

```
if ((err=SSLHashSHA1.update(&hashCtx,&signedParams)) != 0)
    goto fail;
    goto fail;
if ...
...
fail: return err;
```

# Les limites des analyses de code

*Beware of bugs in the above code;*
*I have only proved it correct, not tried it.*
(Donald E. Knuth, 1977)

Risques d'erreurs dans l'analyse faite à la main et d'*unsoundness* dans les analyseurs statiques.

Décalage possible entre l'analyse et le ≪vrai≫ programme ou ses conditions d'exécution.

**Exemple (Ariane 501)**

Débordement dans conversion flottant 64 bits $\rightarrow$ entier 16 bits.

Une analyse menée dans le contexte d'Ariane 4 avait montré que cette quantité BH ≪tenait≫ dans 16 bits. Analyse invalide dans le contexte d'Ariane 5.

# La vérification déductive (aussi appelée *program proof* )

Établir, par le raisonnement logique, des propriétés vraies de *toutes* les exécutions possibles du programme.

Au contraire d'autres ≪méthodes formelles≫, les propriétés établies vont jusqu'à la correction complète vis-à-vis de la spécification.

Intérêts pratiques :

- Obtenir des garanties plus fortes
  que celles atteignables par test et revues.
- Trouver des bugs introuvables autrement.

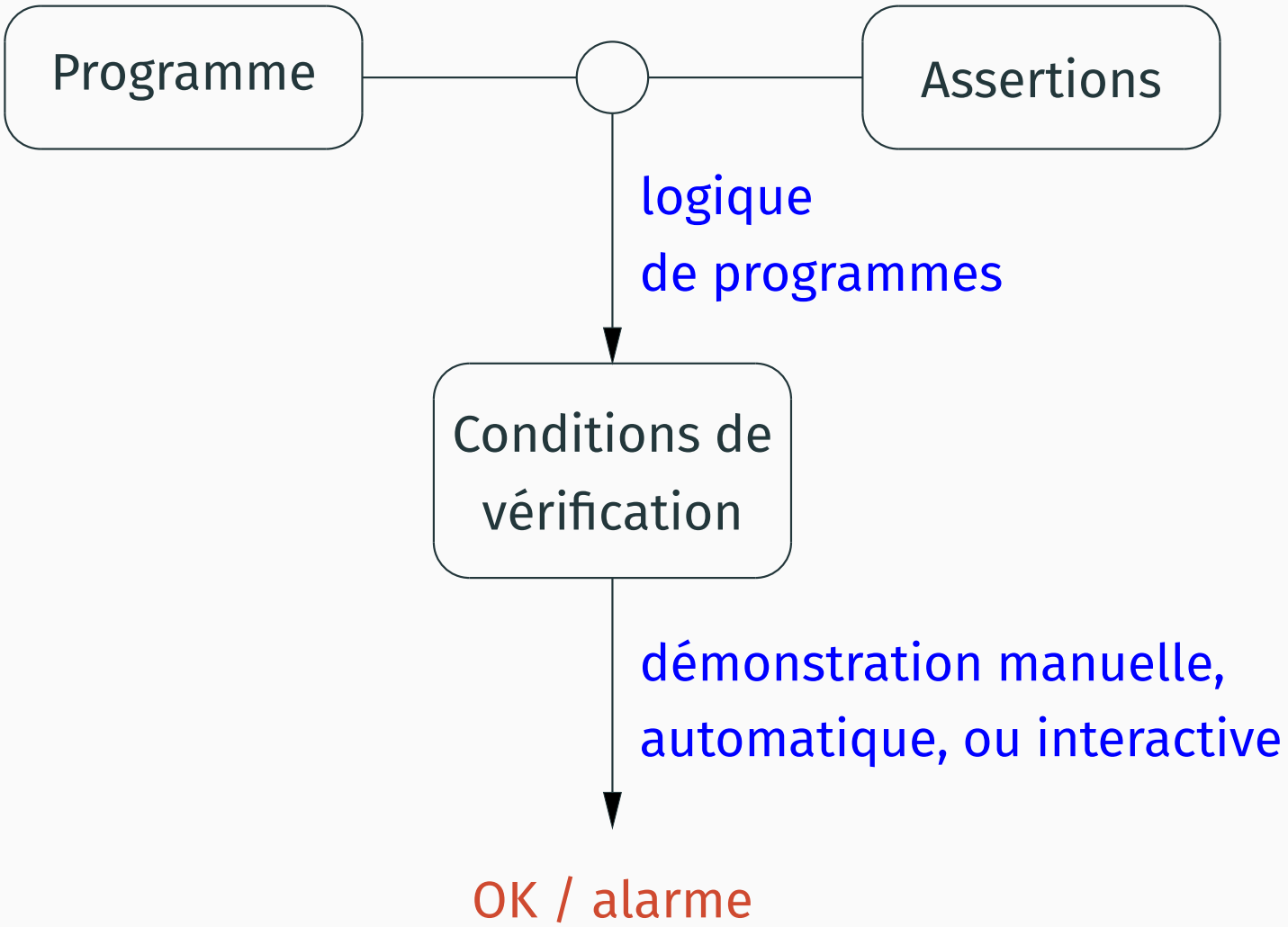
# Les logiques de programmes

Une logique de programmes fournit un langage de spécification et des principes de raisonnement sur les comportements du programme.

Les spécifications se présentent généralement comme des assertions logiques portant sur le programme :

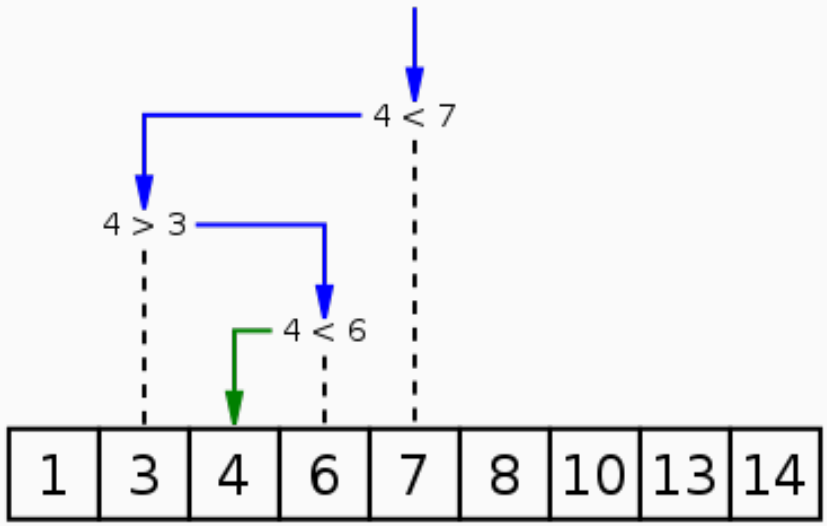
- préconditions : hypothèses sur les entrées
  (paramètres de fonction ; valeurs initiales des variables)
- postconditions : garanties sur les sorties
  (résultats de fonction ; valeurs finales des variables)
- invariants : garanties sur l'état en un point du code
  (invariants de boucles, de structures de données, …)

# Logiques de programmes et vérification déductive



# À la chasse au bug :
# La recherche dichotomique

# La recherche dichotomique




```
l = 0; h = a.length - 1;
while (l <= h) {
  m = (l + h) / 2;
  if (a[m] == v) return m;
  if (a[m] < v) h = m - 1; else l = m + 1;
}
return -1;
```

# Une longue histoire

```
l = 0; h = a.length - 1;
while (l <= h) {
  m = (l + h) / 2;
  if (a[m] == v) return m;
  if (a[m] < v) h = m - 1; else l = m + 1;
}
return -1;
```

1946 John Mauchly, *Moore School Lectures*

1960 Derrick H. Lehmer publie l'algorithme moderne

1986 Jon Bentley, *Programming pearls*, chapitre 4

2004 Rapport de bug : `java.util.Arrays.binarySearch()` *will throw an* `ArrayIndexOutOfBoundsException` *if the array is large.*

2006 Joshua Bloch, *Nearly All Binary Searches and Mergesorts are Broken*.

# La source du bug : un débordement arithmétique

```
m = (l + h) / 2;
```

On a $0 \leq$ `l` $\leq$ `h` $<$ `a.length`.

`l + h` peut déborder si `a.length` est suffisamment grand.

En Java, `l + h` est alors négatif, donc `m` aussi, et `a[m]` lève une exception «accès hors bornes».

En C, on a un «comportement indéfini» ou on continue avec `m` incorrect.

Un correctif simple : `m = l + (h - l) / 2;`

# Pourquoi ce bug est difficile à trouver ?

**Test :**

- On teste rarement sur de très grosses données.
- Il faut une machine 64 bits et plusieurs Go de RAM pour faire apparaître ce bug.

**Revues :**

- La formule $(l + h)/2$ est familière.
- Risque d' ≪optimiser≫ $l + (h - l)/2$ en $(l + h)/2$.

**Analyses :**

- Une analyse d'intervalles de variation peut signaler l'erreur.

# Démo

Vérification déductive de la recherche dichotomique avec l'outil Frama-C.

# Le cours et le séminaire

# Objectifs du cours

Comprendre les principes des logiques de programmes et les développements récents dans ce domaine.

*Leitmotiv* : quelles logiques pour quels traits des langages de programmation ?

(variables, pointeurs, parallélisme, ordre supérieur, etc)

# Objectifs du séminaire

Illustrer l'utilisation de logiques de programmes dans des outils de vérification de qualité industrielle.

Montrer de nouveaux problèmes de vérification et de nouvelles idées pour les aborder.

# La suite du cours

| | |
|---|---|
| 04/03 | La naissance des logiques de programmes |
| 11/03 | Variables et boucles : la logique de Hoare |
| 18/03 | Pointeurs et structures de données : la logique de séparation |
| 25/03 | Parallélisme à mémoire partagée : la logique de séparation concurrente |
| 01/04 | Extensions de la logique de séparation : permissions fractionnaires, état fantôme, algèbres de ressources, … |
| 08/04 | Logiques pour la mémoire partagée faiblement cohérente |
| 15/04 | Logiques pour les langages fonctionnels et l'ordre supérieur |

# Le séminaire

11/03 Loïc Correnson (CEA).
Les logiques de programmes à l'épreuve du réel : tours et détours avec Frama-C/WP.

18/03 Yannick Moy (Adacore).
Preuve auto-active de programmes en SPARK

25/03 Bart Jacobs (K. U. Leuven).
*VeriFast : Semi-automated modular verification of concurrent C and Java programs using separation logic*

01/04 François Pottier (Inria).
Raisonner à propos du temps en logique de séparation

08/04 Jacques-Henri Jourdan (CNRS).
Protocoles personnalisés en logique de séparation : ressources fantômes et invariants dans la logique Iris

15/04 Philippa Gardner (Imperial College London).
*Gillian : Compositional Symbolic Testing and Verification*