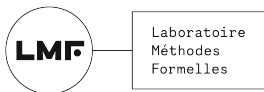


La vérification déductive avec l'outil Why3

Jean-Christophe Filiâtre
CNRS

Journée Algorithmes d'aide à la décision publique
4 octobre 2022



une **approche mathématique** de la correction du logiciel :

- ① écrire une **spécification** dans un langage mathématique
- ② construire une **preuve** que le programme la satisfait

un exemple (extrême)

```
a[52514],b,c=52514,d,e,f=1e4,g,h;main(){for(;b=c-=14;h=printf("%04d",  
e+d/f))for(e=d%=f;g--b*2;d/=g)d=d*b+f*(h?a[b]:f/5),a[b]=d%--g;}
```

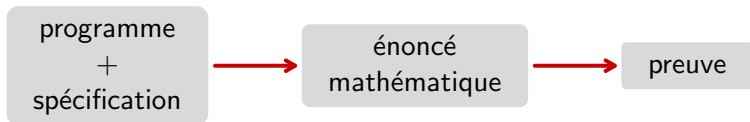
```
a[52514], b, c=52514, d, e, f=1e4, g, h; main() {for(; b=c-=14; h=printf("%04d", e+d/f)) for(e=d%=f; g=--b*2; d/=g) d=d*b+f*(h?a[b]:f/5), a[b]=d%--g; }
```

spécification affiche 15 000 décimales de π

preuve beaucoup de maths, dont

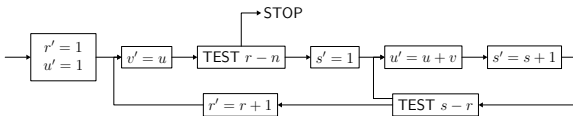
$$\pi = \sum_{i=0}^{\infty} \frac{(i!)^2 2^{i+1}}{(2i+1)!}$$

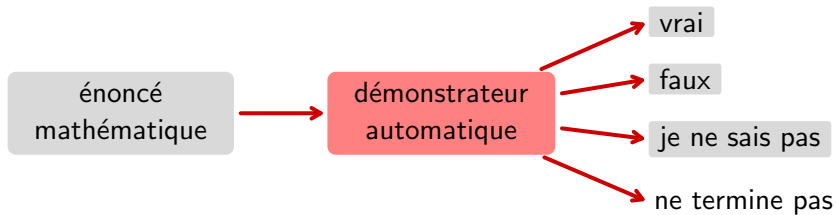
- ① la vérification déductive
- ② l'outil Why3
- ③ les enjeux de la vérification
- ④ leçons d'une expérience avec Parcoursup

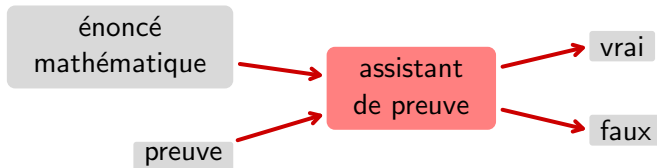
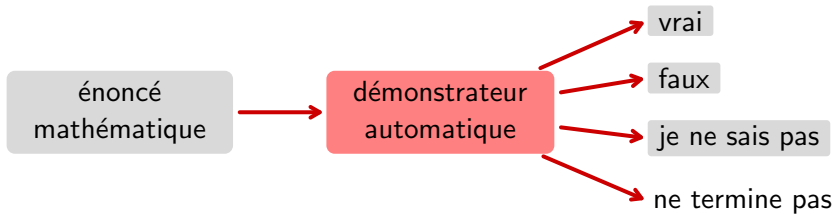




A. M. Turing. Checking a large routine. 1949.

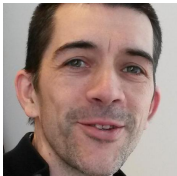
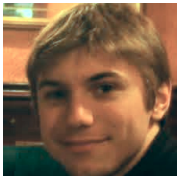






une plateforme pour la vérification déductive

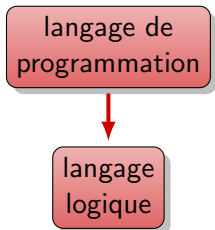
- 1999 : prototype intégré à Coq
- 2001 : outil indépendant
- 2009 : refonte complète



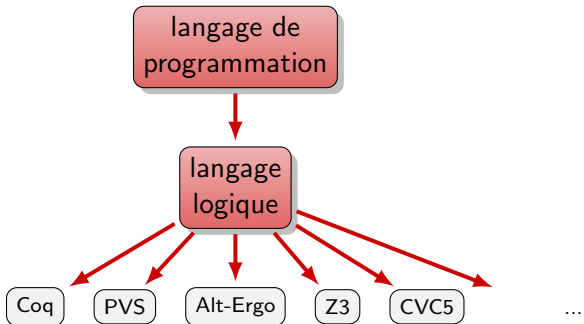
Why3, un couteau suisse

langage de
programmation

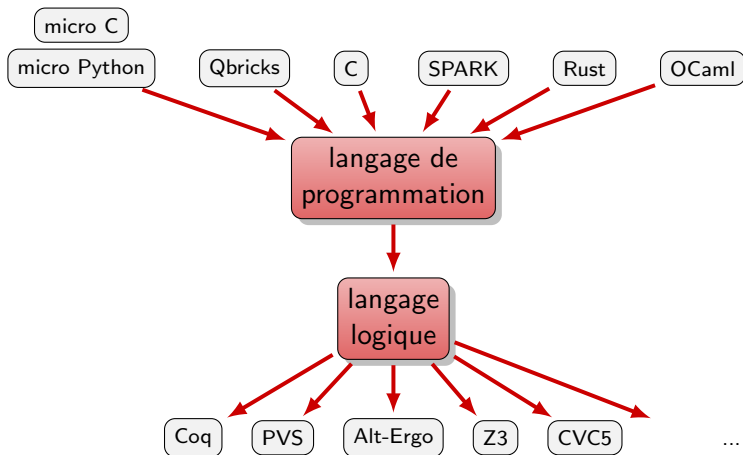
langage
logique



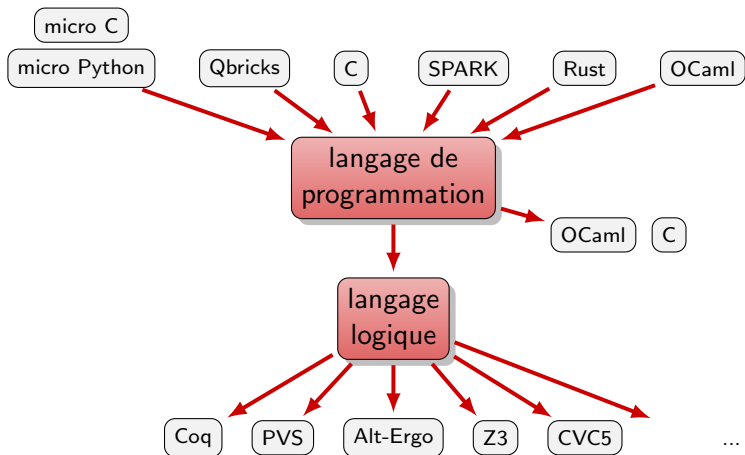
Why3, un couteau suisse



Why3, un couteau suisse





Why3, un couteau suisse



les enjeux de la vérification déductive

écrire une spécification est un processus

- complexe
- coûteux
- qui ne s'automatise pas
- qui comporte des pièges
 -  spécification incomplète
 -  spécification incorrecte

```
void yet_another_sort(a: array int)
```

entrée un tableau d'entiers

sortie le tableau est trié dans l'ordre croissant

```
void yet_another_sort(a: array int)
```

entrée un tableau d'entiers

effets le tableau est modifié en place

sortie le tableau est trié dans l'ordre croissant

```
void yet_another_sort(a: array int)
```

entrée un tableau d'entiers

effets le tableau est modifié en place

sortie le tableau est trié dans l'ordre croissant

1	2	2
---	---	---

→

0	0	0
---	---	---

```
void yet_another_sort(a: array int)
```

entrée un tableau d'entiers

effets le tableau est modifié en place

sortie le tableau est trié dans l'ordre croissant

tout élément présent en entrée est présent en sortie

```
void yet_another_sort(a: array int)
```

entrée un tableau d'entiers

effets le tableau est modifié en place

sortie le tableau est trié dans l'ordre croissant

tout élément présent en entrée est présent en sortie

1	2	2
---	---	---

→

1	2	3
---	---	---

```
void yet_another_sort(a: array int)
```

entrée un tableau d'entiers

effets le tableau est modifié en place

sortie le tableau est trié dans l'ordre croissant

tout élément présent en entrée est présent en sortie

tout élément présent en sortie était présent en entrée


```
void yet_another_sort(a: array int)
```

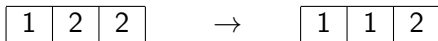
entrée un tableau d'entiers

effets le tableau est modifié en place

sortie le tableau est trié dans l'ordre croissant

tout élément présent en entrée est présent en sortie

tout élément présent en sortie était présent en entrée



```
void yet_another_sort(a: array int)
```

entrée un tableau d'entiers

effets le tableau est modifié en place

sortie le tableau est trié dans l'ordre croissant

le tableau est une permutation du tableau initial

```
void yet_another_sort(a: array int)
```

entrée un tableau d'entiers

effets le tableau est modifié en place

terminaison garantie

sortie le tableau est trié dans l'ordre croissant

le tableau est une permutation du tableau initial

la vérification implique la recherche d'**invariants**, un processus

- complexe
- coûteux
- qui ne s'automatise pas

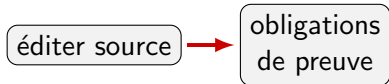
la vérification implique la recherche d'**invariants**, un processus

- complexe
- coûteux
- qui ne s'automatise pas

[éditer source](#)

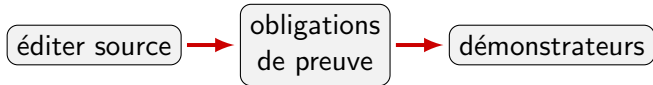
la vérification implique la recherche d'**invariants**, un processus

- complexe
- coûteux
- qui ne s'automatise pas



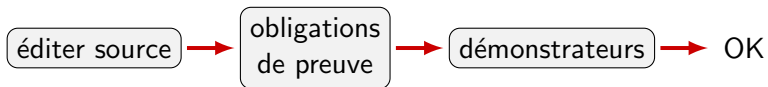
la vérification implique la recherche d'**invariants**, un processus

- complexe
- coûteux
- qui ne s'automatise pas



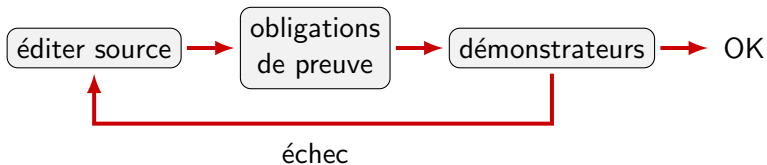
la vérification implique la recherche d'**invariants**, un processus

- complexe
- coûteux
- qui ne s'automatise pas



la vérification implique la recherche d'**invariants**, un processus

- complexe
- coûteux
- qui ne s'automatise pas



File Edit Tools View Help

Status Theories/Goals

- random_access_list.mlw
 - RandomAccessList
 - length_flatten'vc [VC for length_flatten]
 - size'vc [VC for size]
 - cons'vc [VC for cons]
 - nth_flatten'vc [VC for nth_flatten]
 - lookup'vc [VC for lookup]
 - split_vc
 - 0 [unreachable point]
 - 1 [variant decrease]
 - 2 [precondition]
 - 3 [precondition]
 - 4 [variant decrease]
 - 5 [precondition]
 - 6 [precondition]
 - 7 [postcondition]

Task random_access_list.mlw

```

79
80 let rec lookup (i: int) (l: ral 'a) : 'a
81   requires { i <= 1 < length (elements l) }
82   variant { i, l }
83   ensures { nth i (elements l) = Some result }
84   =
85   match l with
86   | Empty   -> absurd
87   | One x l1 -> if i = 0 then x else lookup (i - 1) (Zero l1)
88   | Zero l1  -> let (x0, x1) = lookup (div i 2) l1 in
89                 if mod i 2 = 0 then x0 else x1
89   end
90
91
92 let rec tail (l: ral 'a) : ral 'a
93   requires { elements l << Nil }
94   variant { l }
95   ensures { match elements l with
96             | Nil -> false
97             | Cons _ l -> elements result = l
98             end }
99   =
100  match l with
101  | Empty   -> absurd
102  | One _ l1 -> Zero l1
103  | Zero l1  -> let (_, x1) = lookup 0 l1 in One x1 (tail l1)
104  end
105

```

0/0/0

Messages Log Edited proof Prover output Counterexample

une obligation de preuve non vérifiée peut signifier

- un problème dans la spécification
- un problème dans le code
- un échec du démonstrateur

faire la part des choses demande **une grande expertise**

leçons d'une expérience avec Parcoursup

travail entamé au LMF en 2019 et aujourd'hui poursuivi au LaBRI

outils

- papier/crayon
- Coq
- JML
- Why3

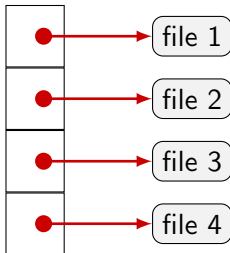
cf Vérification de l'algorithme de calcul des ordres d'appel dans Parcoursup (Castéran, Gimbert, Mathieu, Point, 2022)

issue de *Document de présentation des algorithmes de Parcoursup*

semi-formelle = français + maths

traduite en Why3 / Coq / JML


```
EnumMap<TypeCandidat, Queue<VoeuClasse>> filesAttente
```



- code déjà écrit



- code déjà écrit



- code produit au final



- code déjà écrit



- code produit au final



- code écrit au fur et à mesure de la preuve
 - possiblement dans un fragment bien identifié (cf $\text{SPARK} \subseteq \text{Ada}$)

- la spécification est fondamentale
 - ne s'automatise pas
- la vérification est possible mais coûteuse
 - ne s'automatise pas
- les chercheurs travaillent à réduire l'effort demandé
 - entre science et ingénierie
- on a besoin de développeurs **éduqués à la vérification**