Prouver un programme, ça veut dire quoi?

David Delahaye

Première journée mondiale de la logique
Université de Montpellier, Campus Triolet, Polytech
14 janvier 2019

LIRMM, Université de Montpellier, CNRS
Département Informatique, équipe MaREL

Sécurité logicielle

Sûreté de fonctionnement

- Fiabilité : fait de fonctionner pendant un intervalle de temps donné ;
- Disponibilité : fait d'être prêt à l'utilisation à un instant donné ;
- Maintenabilité : aptitude du système à être réparer ou à évoluer ;
- Sécurité : aptitude du système à éviter de faire apparaître des événements critiques ou catastrophiques.

Objectif

- Éviter les défaillances (faute $\Rightarrow$ erreur $\Rightarrow$ défaillance).

Plusieurs réglementations avec plusieurs niveaux de sûreté

- Critères communs (EAL 1 à EAL 7) ;
- IEC 61508 (SIL 1 à SIL 4).

Contexte

Développement de systèmes critiques

Systèmes critiques : système dont la panne peut avoir des conséquences dramatiques (morts, dégâts matériels importants, conséquences graves pour l'environnement).

Domaines d'application critiques :

- Transports : avions, trains, automobiles ;
- Production d'énergie : contrôle des centrales nucléaires ;
- Santé : chaînes de production de médicaments, appareil médicaux ;
- Système financier : paiement électronique ;
- Domaine militaire.

Est-ce rare d'avoir des bugs dans les systèmes critiques ?

Bug ? Vous avez dit bug ?

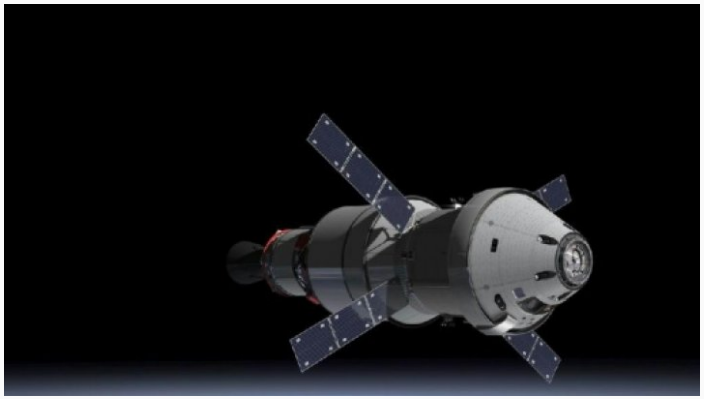
Catastrophes dues à des erreurs informatiques

- 2003 : arrêt en cascade et simultanées de 256 centrales électriques en Amérique du Nord.
  50 millions de foyers privés d'électricité, 11 morts,
  6 milliards de dollars de dégâts.
- 1985-1987 : dysfonctionnement du logiciel de la machine de radiothérapie Therac 25.
  Surdosage de radiations (jusqu'à 100 fois la dose de radiations).
  Mort directe de 6 patients.
- 2012 : problème dans le logiciel algorithmique de passage d'ordre de l'entreprise Knight capital group.
  Fortes fluctuations de l'action, perte de 440 millions de dollars.
  L'entreprise a frôlé la faillite, recapitalisation en urgence.

Vous avez besoin d'un exemple plus récent ?

Bug ? Vous avez dit bug ?

Vaisseau américain Orion



- Premier vol habité d'Orion retardé à 2023 (17/09/2015) ;
- « Trop de paramètres et d'incertitudes sont en jeu pour permettre de déterminer si le vaisseau peut être prêt en 2021, a renchéri l'administrateur associé de la Nasa, Robert Lightfoot, lors d'une conférence de presse téléphonique.
  "C'est très loin d'être certain car des choses peuvent arriver, comme l'expérience nous l'a montré", a-t-il ajouté, citant notamment des "inconnus" comme des problèmes potentiels dans la mise au point des logiciels. »

Méthodes formelles

Différentes méthodes

1. Le typage des langages de programmation est historiquement une des premières méthodes formelles.
2. La vérification déductive consiste à donner une représentation purement logique et sémantique à un programme.
3. Le « model-checking » analyse exhaustivement l'évolution du système lors de ses exécutions possibles.
4. L'analyse statique par interprétation abstraite calcule symboliquement un sur-ensemble des états accessibles du système.

Nous allons nous intéresser aux méthodes déductives.

Preuves formelles

Plusieurs pré-requis

- Avoir une bonne connaissance de la sémantique de son langage ;
- Être capable d'exprimer cette sémantique formellement ;
- Savoir spécifier précisément le comportement de son programme ;
- Faire en sorte que la spécification soit totale.

Plusieurs langages en jeu

- Le langage de programmation ;
- Le langage de spécification ;
- Le langage de preuve.

Si ces trois langages sont réunis au sein du même environnement, c'est beaucoup plus pratique pour le développeur !

Spécification

Qu'est-ce que c'est ?

- C'est le « quoi » du programme, ce qu'il doit faire ;
- Peut-être exprimé dans le langage naturel (mais ambigu) :
  - Exemple : « ce programme calcule la racine carrée ».
- Plus formellement : spécification = type d'un programme.

Plusieurs degrés de spécifications

- Spécifications partielles :
  - Exemple : `sqrt : float` $\rightarrow$ `float` ;
  - Donne de l'information mais pas assez ;
  - Beaucoup de fonctions ont ce type (pas seulement racine carrée).
- Spécifications totales :
  - Exemple : $\forall x \in \mathbb{R}^+ . f(x) \geq 0 \wedge f(x) \times f(x) = x$;
  - Seule racine carrée vérifie cette proposition ;
  - Nécessite un langage basé sur la logique.

Preuves

Objectifs

- Mettre en adéquation un programme et sa spécification ;
- Apporter une garantie sur l'exécution du programme.

Remarques

- Plus simple de faire des preuves sur des programmes fonctionnels ;
- Outils basés sur du fonctionnel : Coq, HOL, PVS, etc. ;
- Outils basés sur de l'impératif : Atelier B.

Un exemple de preuve

Spécification

- On cherche à écrire une fonction f telle que :
  $\forall x \in \mathbb{N}. f(x) = x \times x$

Programme

- On considère le programme (fonction) suivant :
  $g(x) = x \times x$

Preuve d'adéquation

- On doit prouver que le programme g vérifie la spécification :
  $\forall x \in \mathbb{N}. g(x) = x \times x$
- On « déplie » la définition de g :
  $\forall x \in \mathbb{N}. x \times x = x \times x$
- Ce qui est trivial.

Une preuve moins triviale

Spécification

- La même que précédemment, c'est-à-dire :
  $\forall x \in \mathbb{N}. f(x) = x \times x$

Programme

- On considère le programme (fonction) suivant :
  $$h(x, i) = \begin{cases} x, \text{ si } i = 0, 1 \\ x + h(x, i - 1), \text{ sinon} \end{cases}$$
  $$g(x) = h(x, x)$$

Preuve ?

- Par récurrence (exercice).

Outils d'aide à la preuve

Quelles logiques ? Quelles théories ? Quels outils ?

- Logique classique, logique intuitionniste ;
- Logique du premier ordre, logique d'ordre supérieur ;
- Théorie des types, théorie des ensembles ;
- Polymorphisme, types dépendants, types inductifs, etc.

Le cas de l'outil d'aide à la preuve Coq

L'induction à la base de la formalisation

- On peut tout formaliser à base de types inductifs ;
- Types inductifs pour les types de données ;
- Relations inductives pour spécifier des comportements ;
- Fonctions récursives pour les programmes ;
- Preuves par induction pour l'adéquation prog./spéc. ;
- Moyen idiomatique de formalisation de beaucoup d'outils.

Support pour l'induction

- Générer les schémas d'induction automatiquement ;
- Pouvoir en générer de nouveaux au besoin ;
- Gérer les lemmes d'inversion automatiquement.

L'induction dans les autres systèmes

Systèmes formels et outils

- Induction présente en théorie des ensembles ;
- Théories des types dédiées :
  - Système T de Gödel, théorie des types de Martin-Löf, calcul des constructions inductives de Coquand-Huet-Paulin.
- Outils dédiés : Coq, Lego, Alfa, etc.

Historiquement

- Formulation explicite de l'induction au 17ème siècle ;
- Auparavant : utilisation de l'induction mathématique ;
- Pascal : « Traité du triangle arithmétique » ;
- Fermat : descente infinie.

Preuve du théorème de Fermat pour $n = 4$

Théorème

Il n'existe pas d'entiers non nuls x, y, et z, tels que :

$$x^4 + y^4 = z^4$$

Le théorème se déduit aisément de la preuve du 20ème problème de Diophante : est-ce qu'un triangle rectangle dont les côtés sont mesurés par des entiers peut avoir une surface mesurée par un carré ?

Fermat a résolu la question par la négative et il a démontré qu'il n'existe pas d'entiers naturels non nuls tels que :

$$x^2 + y^2 = z^2 \text{ et } xy = 2t^2$$

Le cas de l'outil d'aide à la preuve Coq

Logique classique

- Une formule est toujours vraie ou fausse ;
- Que je puisse en démontrer la validité ou non ;
- Logique du « tiers exclu » : $A \vee \neg A$.

Logique intuitionniste

- Une formule est vraie, fausse, ou « on ne sait pas » ;
- Si on ne sait en démontrer la validité, alors « on ne sait pas » ;
- Le « tiers exclu » n'est pas admis dans cette logique.

Logiques classique/intuitionniste

Sémantique du « il existe »

- En logique classique : $\exists x.P(x) \equiv$ il existe n termes $t_1, t_2, \ldots, t_n$ tels que $P(t_1) \vee P(t_2) \vee \ldots \vee P(t_n)$ est vraie (théorème de Herbrand) ;
- En logique intuitionniste : $\exists x.P(x) \equiv$ il existe un terme t tel que $P(t)$ est vraie.
  On doit construire un témoin t qui vérifie P et en avoir l'intuition.
  D'où le nom de logique « intuitionniste » ou « constructive ».

Logique classique

- La logique classique est une logique assez « exotique » ;
- On peut démontrer une formule $\exists x.P(x)$ sans jamais montrer un seul témoin qui fonctionne ! Exemple : $\exists x.P(x) \Rightarrow P(a) \wedge P(b)$.
- De ce fait, c'est plus facile de faire des preuves en logique classique qu'en logique intuitionniste.

Interprétation de Brouwer-Heyting-Kolmogorov

Interprétation BHK

- Interprétation de la logique intuitionniste (sans le tiers exclu) ;
- Proposée par Brouwer et Heyting, et aussi par Kolmogorov ;
- Appelée aussi « interprétation par réalisabilité » (Kleene) ;
- Idée : donner une interprétation fonctionnelle aux preuves.

Par induction sur les formules

- Une preuve de $A \Rightarrow B$ est une fonction qui associe à une preuve de A une preuve de B ;
- Une preuve de $A \wedge B$ est un couple (π_1, π_2), où π_1 est une preuve de A et π_2 une preuve de B, etc. ;
- Une preuve de $\exists x.A(x)$ est un couple (t, π), où t est un objet et π est une preuve de $A(t)$.

Isomorphisme ou correspondance de Curry-Howard

Principe et historique

- Basé sur une double correspondance :
  - Correspondance preuves/programmes ;
  - Correspondance formules/types.
- Curry : analogie entre les preuves dans les systèmes à la Hilbert et la logique combinatoire ;
- Howard : analogie entre les preuves en déduction naturelle intuitionniste et les termes du λ-calcul typé.

Prouver = programmer

Un exemple

- Théorème :
  $\forall l : (\text{list nat}).\exists l' : (\text{list nat}).\text{est\_permut}(l, l') \wedge \text{est\_trie}(l')$;
- La preuve de ce théorème exhibe un algorithme de tri ;
- Le programme et la preuve sont réalisés au même moment !

Faits marquants

Ligne Meteor (ligne 14)

- Ouverte à Paris il y a 20 ans ;
- Par Matra (Siemens maintenant) ;
- Utilisation de la méthode B (J.-R. Abrial, 1996) ;
- Outil dédié : l'Atelier B (Alstom, puis Clearsy) ;
- 27 800 obligations de preuve !

Autres faits marquants

JavaCard (Gemalto)

- Formalisation de l'architecture JavaCard ;
- Utilisation de Coq (Inria, πr^2).

Compilateur certifié CompCert (Inria, Gallium)

- Produit du code pour PowerPC, ARM, et x86 ;
- Développé en Coq et certifié correct.

Projet L4.verified (NICTA)

- Formalisation du micro-noyau seL4 ;
- Utilisation d'Isabelle/HOL (Cambridge/Munich).

Formalisation des Mathématiques

Manifeste QED

- Proposition à la conférence CADE 12 (1994) ;
- Construire une base de données de toute notre connaissance mathématique, entièrement formalisée et avec les preuves vérifiables automatiquement par un système.

Formalisation des Mathématiques

« Grundlagen der Analysis » de Landau

- Traduction du livre entier par L. S. van Benthem Jutting ;
- Utilisation d'Automath (N. G. de Bruijn).

Projet Mizar

- Meilleure approximation du manifeste QED ;
- Développé en Mizar (A. Trybulec).

Projet « Mathematical Components »

- Théorème des 4 couleurs, théorème de Feit-Thompson ;
- Utilisation de Coq.

Distinctions

Outil d'aide à la preuve Coq

- Outil développé depuis plus de 30 ans par Inria ;
- « SIGPLAN Programming Languages Software Award » (01/2014) ;
- « ACM Software System Award » (06/2014).

Tendances actuelles

Preuve interactive

- Beaucoup d'outils (surtout pour le fonctionnel) ;
- Très expressif mais peu automatisable.

Preuve automatique

- Outils de preuve au premier ordre (tableaux, résolution, etc.) ;
- Solveurs SMT (solveurs SAT + théories).

Plates-formes de preuve

- Ne pas construire un unique outil d'aide à la preuve ;
- Plate-forme regroupant des collections d'outils de preuve ;
- Exemples : Why3, Frama-C, BWare, Sledgehammer, etc.