Sécurité logicielle

David Delahaye

Faculté des Sciences
David.Delahaye@lirmm.fr

Master M2 2018-2019

Sécurité logicielle

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 ?

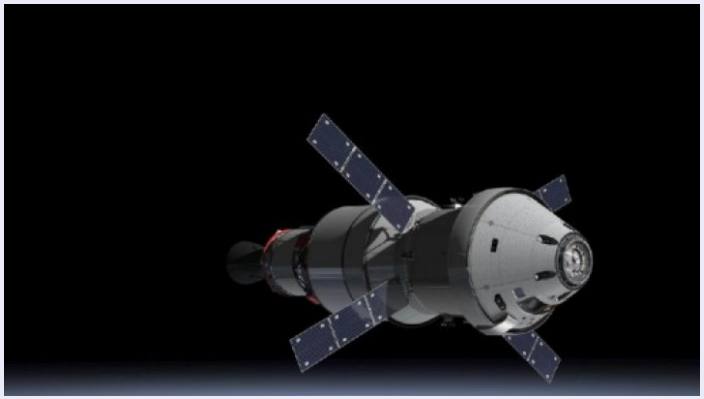
« catastrophes » dues à des erreurs informatiques

- 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. »

D'où viennent les bugs ?

Sources des bugs

- Fautes de cahier des charges ;
- Fautes de spécification ;
- Fautes de conception ;
- Fautes de réalisation ;
- Fautes de production.

Plusieurs difficultés

- Choix des architectures et des outils utilisés ;
- Tests non appropriés ;
- Transmission de l'information entre plusieurs corps de métiers ;
- Pas de vue globale de la sûreté de fonctionnement du système.

Un exemple

Description

Extrait de ssshandbook.pdf mis à disposition par le Joint Software System Safety Committee.

Un avion de combat avait été développé très rigoureusement, testé exhaustivement avant sa mise en service. De nombreux exemplaires étaient en exploitation.
Demande d'évolution du système de défense embarqué.
Le contrôle de l'un des composants demandait une transmission au sol très précise de la vitesse atteinte.
Cela permit aux contrôleurs du système de constater que tous les ordinateurs embarqués faisaient un « shut-down », puis une réinitialisation à la vitesse Mach 1.
Le pilote ne s'était aperçu de rien car l'avion était dans un état suffisamment stable pour que la défaillance reste temporaire.

Un exemple

Analyse

Une équation décrivant l'aérodynamique du système fait l'hypothèse qu'une certaine grandeur tend vers l'infini à Mach 1, fait bien connu des ingénieurs en aérodynamique, qui savent faire la transition vers les vitesses supersoniques.

L'ingénieur logiciel n'avait aucune compréhension du modèle physique de la transition vers les vitesses supersoniques à Mach 1.

L'ingénieur faisant l'interface entre les aérodynamiciens et les informaticiens ne connaissait pas ce problème de transition et envoya donc, telle quelle, l'équation aux ingénieurs soft pour qu'elle soit codée. Elle le fut telle quelle sans évaluation préalable des résultats.

Un exemple

Recensement des fautes

Faute de cahier des charges / spécification : pas d'indication des limites de la pertinence physique de l'équation.

Faute de spécification / conception : pas de considération de la précision des calculs sur ordinateur dans l'utilisation de l'équation.

Faute d'évaluation : les tests n'ont pas porté sur une vitesse de Mach 1.

Un exemple

Leçons tirées

Specified equations describing physical world phenomenon must be thoroughly defined, with assumptions as to accuracy, ranges, use, environment, and limitations of the computation.

When dealing with requirements that interface between disciplines, it must be assumed that each discipline knows little or nothing about the other and, therefore, must include basic assumptions.

Boundary assumptions should be used to generate test cases as the more subtle failures caused by assumptions are not usually covered by ordinary test cases (division by zero, boundary crossing, singularities, etc.).

Méthodes formelles pour la sécurité logicielle

Sûreté de fonctionnement

- Méthodes formelles : partie (assez fine) d'un domaine plus large qu'est la sûreté de fonctionnement ;
- Facettes de la sûreté de fonctionnement :
  - Fiabilité (« Reliability ») : fait de fournir un service pendant un intervalle de temps donnée ;
  - Disponibilité (« Availability ») : fait d'être prêt à l'utilisation à un instant donné ;
  - Maintenabilité (« Maintainability ») : aptitude du système à être réparer ou à évoluer ;
  - Sécurité (« Safety ») : aptitude du système à éviter de faire apparaître des événements critiques ou catastrophiques.
- Acronyme : FDMS en français ou RAMS en anglais.

Défaillances, erreurs, et fautes

Vocabulaire

- Défaillance : altération temporaire ou permanente du service délivré ;
- Défaillance causée par une (ou des) erreurs ;
- Erreur : état inattendu du système ;
- Erreur causée par une (ou des) fautes ;
- Faute : modification non voulue par le concepteur ;
- Défaillances internes ou externes au système.

Chaîne de causalité

- Faute $\Rightarrow$ erreur $\Rightarrow$ défaillance ;
- Potentiellement cyclique : défaillance $\Rightarrow$ faute $\Rightarrow$ etc.

Exemple d'enchaînement faute, erreur, défaillance

Nœud réseau

- Système : le code dans un noeud (commutateur, routeur, ...) d'un réseau de télécommunications. Dans ce code, le programmeur a écrit « i = 0 ; » au lieu de « i = 1 ; », qui était l'instruction correcte. On a une faute.
- À un moment particulier, l'ordinateur exécute cette instruction, et suite à un certain calcul, un tampon est dimensionné à 10 au lieu d'être dimensionné à 100. On a une erreur.
- Les conditions d'utilisation du réseau font qu'exceptionnellement, ce jour-là, la charge est telle que le tampon reçoit un trafic trop important. Il y a alors trop de pertes (plus que les niveaux maximaux spécifiés). On a une défaillance.

Difficulté de diagnostic des fautes

Histoire véridique

- Un patient en dialyse est atteint d'un malaise grave après un certain temps de dialyse.
- Le dialyseur (rein artificiel) a été complètement testé au début de la séance. Il est testé à nouveau après l'accident. Aucune défaillance ne peut être mise en évidence.
- Le dialyseur ne provoque aucun malaise chez les autres patients.
- L'accident se reproduit plusieurs fois avec le même patient, avec des degrés de gravité variés.

Que se passe-t-il ?

- Une idée de la raison ?

Difficulté de diagnostic des fautes

Identification du problème

- Après un certain nombre de recherches, l'équipe médicale se rend compte que les accidents surviennent alors que le patient est en train de téléphoner sur son portable.
- La gravité des malaises était proportionnelle à la longueur du coup de téléphone. Les ondes électro-magnétiques perturbaient le gestionnaire de la composition du dialysat (liquide qui fait les échanges avec le sang au travers d'une membrane). celui-ci était parfait avant et après. On a renforcé le blindage et interdit les portables.

Conclusion

- La faute apparaît à l'usage.
- Elle est due à une interaction, impossible à imaginer au moment de la conception du produit.
- Le diagnostic est difficile : panne byzantine.

Classification des défaillances

Suivant

- La manière dont elles sont perçues ;
- La nature de la perturbation ;
- La durée de la défaillance ;
- La gravité/sévérité des conséquences de la défaillance ;
- La fréquence de l'occurrence des défaillances.

Mode de défaillances

- Un mode de défaillances est défini par ces caractéristiques.
- Le mot « mode » est consacré : il signifie « genre, sorte, catégorie ».

Classification des défaillances

Suivant la manière dont elles sont perçues

- Défaillance cohérente : tous les utilisateurs en ont la même perception ;
- Défaillance incohérente ou byzantine : elle n'est constatée que par certains utilisateurs.

Suivant la nature de la perturbation

- Défaillance de valeur : délivrance d'une valeur erronée.
- Défaillance temporelle : la réaction n'arrive pas au moment attendu (elle arrive soit trop tôt, soit trop tard).
- Défaillance par arrêt : plus de données, plus de réaction. On parle d'arrêt sur défaillance.

Classification des défaillances

Suivant la durée de la défaillance

- Défaillance durable : altère le service rendu pendant un temps représentant une fraction importante de la durée de la mission ;
- Défaillance temporaire : altère le service rendu momentanément, puis fournit à nouveau un service correct.

Suivant la gravité/sévérité des conséquences de la défaillance

- Défaillance bénigne : pas de conséquence sérieuse, la mission se poursuit normalement ;
- Défaillance significative : la mission est affectée, perte d'efficacité du service rendu, à chiffrer ;
- Défaillance critique : réduction dangereuse des marges de sécurité ;
- Défaillance catastrophique : mission arrêtée, destruction du produit ou du procédé, blessures, pertes de vies humaines.

Classification des défaillances

Suivant la durée de la défaillance

- Défaillance durable : altère le service rendu pendant un temps représentant une fraction importante de la durée de la mission ;
- Défaillance temporaire : altère le service rendu momentanément, puis fournit à nouveau un service correct.

Niveaux de gravité

- Dans l'aéronautique civile : mineur, majeur, dangereux, catastrophique ;
- Dans l'automobile : mineur, majeur, sérieux, fatal ;
- Dans le nucléaire : significatif, majeur, grave, catastrophique ;
- Dans l'industrie des lanceurs spatiaux : incident mineur, incident, incident grave ;
- Ces niveaux sont définis dans les normes des métiers.

Classification des défaillances

Suivant la fréquence de l'occurrence des défaillances

- Niveaux de probabilité : extrêmement improbable, extrêmement rare, rare, probable, fréquent.

Classification des systèmes suivant les défaillances

- Criticité d'un système : plus forte gravité de ses modes de défaillance ;
- Système sûr en présence de défaillances : la probabilité de défaillances catastrophiques est acceptable ;
- Système à défaillance contrôlée : soit sûr en présence de défaillances, soit à arrêt sur défaillances.

Erreurs et fautes

De l'erreur à la défaillance

- Une défaillance (non accomplissement de la fonction) est la manifestation d'une erreur dans l'état du système.
- Une erreur peut être effacée avant de provoquer une défaillance.
- Une erreur peut être éliminée par exemple par redondance.

De la faute à l'erreur

- Une erreur est produite par une faute.

Fautes

Causes phénoménologiques des fautes

- Acteurs humains : exemple ? Oubli d'un chiffon dans une tubulure.
- Mauvaise adaptation de la technologie du produit : exemple ? Mauvaise résistance des matériaux à l'échauffement.
- Outils employés : exemple ? Bug d'un compilateur d'un langage, compilateur « dans le silicium ».
- Environnement : exemple ? Coupure de courant.

Caractérisation des fautes

- Passive ou dormante : existe mais ne perturbe pas le fonctionnement du système ;
- Active : son activation produit une erreur de l'un des composants du système.

Fautes

Classification des fautes

- Par nature : accidentelles/intentionnelles sans volonté de nuire ou avec volonté de nuire ;
- Par le moment de leur création :
  - Fautes de développement : fautes de spécification/de conception ;
  - Fautes opérationnelles : fautes de production/d'exécution.
- Par leur rapport avec le produit :
  - Fautes internes : exemple ? Instruction erronée, court-circuit.
  - Fautes externes : exemple ? Perturbations.
- Par rapport à leur persistance : permanentes/temporaires.
  - Fautes temporaires externes dites fautes transitoires ;
  - Fautes temporaires internes dites intermittentes (cas du dialyseur).

Un exemple

Ariane V

Système de pilotage de la fusée : système multi-processeur, multi-tâches.

Un composant (C) utilise une fonction de conversion d'un entier en un flottant, sous l'hypothèse (H) que cet entier est compris entre certaines bornes.

(H) vérifiée par Ariane IV.

Composant réutilisé pour Ariane V.
(H) cesse d'être vérifiée.

$\Rightarrow$ La fonction de conversion lève une exception.

Un exemple

Ariane V

Le mécanisme de tolérance aux fautes réagit à cette levée :

- Fait appel au second calculateur, en recopiant les données ;
- Même cause, même effet : levée d'exception.

Hypothèse de la « panne simple » : transmission du code d'erreur en tant que résultat à l'appelant, qui l'interprète comme une valeur fonctionnelle... et braque les tuyères...

Un exemple

Quelle est la faute ?

- Fonction de conversion ?
- Appel avec une valeur trop grande ?
- Accélération trop grande ?
- Moteur trop puissant ?
- Entiers machine trop petits ?

À qui attribuer la faute ?

- La notion de faute est arbitraire.
- Faute : cause adjugée ou supposée d'une erreur.
- La faute est définie par la cause que l'on veut prévenir ou tolérer.
- Permet la distinction entre faute humaine et faute physique.
- Toute faute est-elle une faute de développement ?

Fautes

Latence d'une faute

- Intervalle de temps entre le moment de la création de la faute et la première activation conduisant à une erreur.
- Dépend du composant affecté, de la manière et du moment où il est utilisé et du choix d'observation effectué.
- La latence est définie par une étude statistique et elle s'adresse plutôt aux systèmes physiques. Difficile à déterminer.
- Exemple : rappels de véhicules par les constructeurs automobiles plusieurs années après la mise en vente.
- Latence faible lorsque toutes les configurations possibles sont prises dans un laps de temps court.

Exercice

Compteur binaire

- Cahier des charges : soit un compteur binaire asynchrone par 16 comptant des impulsions arrivant sur une entrée E et affichant le résultat en sortie S en code binaire naturel sur quatre bits (notés (a, b, c, d), poids fort à gauche).
- Une panne de collage fixe le bit de poids fort à 0, dès l'état initial.
- Déterminer la latence moyenne de cette faute, sachant que l'intervalle de temps moyen entre deux entrées est de 2 ms.

Exercice

Solution

- Quel top d'horloge doit faire basculer le bit de poids fort à 1 ?
- Le basculement devrait se faire au 8ème top d'horloge.
- Sa latence est donc $8 \times 2 = 16$ms.

Caractérisation de la faute

- La panne (i.e. faute matérielle) est une faute permanente qui provoque une faute « valeur compteur erronée ».
- Cette faute « valeur compteur erronée » dure 8 tops d'horloge, puis disparaît pendant les 8 tops suivants. Il s'agit d'une faute transitoire.
- Elle provoque une erreur dans la représentation du temps écoulé.
- Cette erreur peut conduire à une défaillance temporaire périodique, si les fautes ou l'erreur ne sont pas détectées et corrigées, ou tolérées.

Méthodes formelles

Différents 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.

Dans la suite, nous allons voir (2).

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

- C'est plus simple de faire des preuves sur des programmes fonctionnels.
- Le fonctionnel sert aussi à encoder les preuves pour certains outils.
- Outils basé sur du fonctionnel : Coq, HOL, PVS, etc.
- Outils basé sur de l'impératif : Atelier B.

Peut-on automatiser ce processus ?

- Pas totalement (problème semi-décidable) ;
- Certains fragments sont décidables :
  - Logique propositionnelle, arithmétique linéaire, réels, géométrie, etc.

Une preuve triviale

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 plus difficile

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 pour la semaine prochaine).

Preuve de programmes fonctionnels

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.

Preuve de programmes fonctionnels

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 :

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

Preuve du théorème de Fermat pour $n = 4$

Principe de la descente infinie

- Preuve par l'absurde : démontrer que résoudre le problème au rang n revient à le résoudre au rang $n-1$, ce qui n'est pas possible car on est borné par 0 ;
- La descente infinie résout des propositions $\nexists x.P(x)$;
- Mais équivalent au principe habituel (contraposée).

Preuve de Fermat chargée d'histoire

- Première utilisation de l'induction ;
- Résolution d'un problème de Diophante (250 apr. J.-C.) ;
- Utilisation des triplets pythagoriciens (Euclide, 300 av. J.-C. ; Babyloniens, 1900-1600 av. J.-C.).

Voir preuve en Coq de Delahaye-Mayero

Un exemple

Spécification

On définit la relation inductive *is_sum* de type $\mathbb{N} \times \mathbb{N} \to \mathrm{Prop}$ de la façon suivante :

1. On a : $is\_sum(0, 0)$;
2. Pour $n, s \in \mathbb{N}$, si $is\_sum(n, s)$, alors on a : $is\_sum(S(n), s + S(n))$.

Fonction

On définit la fonction suivante de type $\mathbb{N} \to \mathbb{N}$:

$$f_{is\_sum}(n) = \begin{cases} 0, \text{ si } n = 0 \\ f_{is\_sum}(p) + S(p), \text{ si } n = S(p), \text{ avec } p \in \mathbb{N} \end{cases}$$

Un exemple

Théorème de correction

L'adéquation entre la fonction et sa spécification se vérifie avec le théorème suivant :

$$\forall n, s \in \mathbb{N}. f_{is\_sum}(n) = s \Rightarrow is\_sum(n, s)$$

Preuve

La preuve se fait par induction sur n.
On utilise le schéma d'induction structurelle sur $\mathbb{N}$:

$$\forall P \in \mathbb{N} \rightarrow \mathrm{Prop}. P(0) \Rightarrow (\forall n \in \mathbb{N}. P(n) \Rightarrow P(S(n))) \Rightarrow \forall n \in \mathbb{N}. P(n)$$

Dans notre cas :

$$P(n) = \forall s \in \mathbb{N}. f_{is\_sum}(n) = s \Rightarrow is\_sum(n, s)$$

Un exemple

Preuve

On applique le schéma d'induction et on doit démontrer :

1. Cas de base :

$$\forall x \in \mathbb{N}. f_{is\_sum}(0) = s \Rightarrow is\_sum(0, s)$$

On calcule $f_{is\_sum}(0)$, ce qui donne :

$$\forall x \in \mathbb{N}. 0 = s \Rightarrow is\_sum(0, s)$$

On remplace s par 0, et on doit démontrer $is\_sum(0, 0)$, qui est le cas de base de la spécification inductive de la relation is_sum.

Un exemple

Preuve

On applique le schéma d'induction et on doit démontrer :

2. Cas inductif : pour $n \in \mathbb{N}$,

$$\forall s \in \mathbb{N}. f_{is\_sum}(S(n)) = s \Rightarrow is\_sum(S(n), s)$$

sous l'hypothèse d'induction :

$$\forall s \in \mathbb{N}. f_{is\_sum}(n) = s \Rightarrow is\_sum(n, s)$$

On calcule $f_{is\_sum}(S(n))$, ce qui donne :

$$\forall s \in \mathbb{N}. f_{is\_sum}(n) + S(n) = s \Rightarrow is\_sum(S(n), s)$$

On remplace s par $f_{is\_sum}(n) + S(n)$, et on doit démontrer :

$$is\_sum(S(n), f_{is\_sum}(n) + S(n))$$

Un exemple

Preuve

On applique le schéma d'induction et on doit démontrer :

2. Cas inductif :
   On applique le cas inductif de la spécification de is_sum, et on doit démontrer :

$$is\_sum(n, f_{is\_sum}(n))$$

   On applique l'hypothèse d'induction avec $s = f_{is\_sum}(n)$, et il nous reste à démontrer que $f_{is\_sum}(n) = f_{is\_sum}(n)$, ce qui est trivial.

Preuve de programmes impératifs

Langage

- Expressions entières et booléennes ;
- Instructions d'affectation, de conditionnelle, et de boucle.

Expressions et instructions

- $e \ ::= \ n \mid x \mid e_1 + e_2 \mid e_1 - e_2 \mid e_1 \times e_2 \mid e_1/e_2$
  $\mid \text{true} \mid \text{false} \mid \text{not}(e) \mid e \text{ and } e \mid e \text{ or } e$
  $\mid e = e \mid e \ != e \mid e < e \mid e \leq e \mid e \geq e \mid e > e$
  où $n \in \mathbb{Z}$ et $x \in \mathbb{V}$ (ensemble de noms de variables) ;
- $i \ ::= \ \text{skip} \mid x := e \mid i; i \mid \text{if } e \text{ then } i \text{ else } i \mid \text{while } e \text{ do } i.$

Logique de Hoare

Triplet de Hoare

- Triplet noté : $\{P\}\ i\ \{Q\}$, où P et Q sont des assertions logiques, et i une instruction ;
- Assertions logiques : exprimées en logique du premier ordre, où les atomes sont les expressions de notre langage ;
- Un triplet de Hoare $\{P\}\ i\ \{Q\}$ est valide si pour tous états E_1 et E_2 tels que si P est vraie dans E_1 et $E_1 \vdash i \rightsquigarrow E_2$ (i termine), alors Q est vraie dans E_2.

Exemples de triplets de Hoare valides

- $\{x = 1\}\ x := x + 2\ \{x = 3\}$;
- $\{x = y\}\ x := x + y\ \{x = 2 \times y\}$.

Logique de Hoare

Règles

$$\frac{}{\{P\}\ \text{skip}\ \{P\}}\ \text{skip} \qquad \frac{}{\{P(e)\}\ x := e\ \{P(x)\}}\ :=$$

$$\frac{\{P\}\ i_1\ \{Q\} \qquad \{Q\}\ i_2\ \{R\}}{\{P\}\ i_1; i_2\ \{R\}}\ ;$$

$$\frac{\{P \wedge e\}\ i_1\ \{Q\} \qquad \{P \wedge \neg e\}\ i_2\ \{Q\}}{\{P\}\ \text{if}\ e\ \text{then}\ i_1\ \text{else}\ i_2\ \{Q\}}\ \text{if}$$

$$\frac{\{I \wedge e\}\ i\ \{I\}}{\{I\}\ \text{while}\ e\ \text{do}\ i\ \{I \wedge \neg e\}}\ \text{while}$$

$$\frac{\{P'\}\ i\ \{Q'\} \qquad P \Rightarrow P' \qquad Q' \Rightarrow Q}{\{P\}\ i\ \{Q\}}\ \text{Aff}$$

Logique de Hoare

Correction totale (avec terminaison)

- La sémantique précédente est partielle : elle suppose que le programme termine ;
- La sémantique peut être totale en imposant que le programme termine (par la pré-condition) ;
- Correction totale :
  Un triplet de Hoare $\{P\}\ i\ \{Q\}$ est valide si pour tous états E_1 et E_2 tels que si P est vraie dans E_1, alors $E_1 \vdash i \rightsquigarrow E_2$ (i termine), et Q est vraie dans E_2 ;
- Nouvelle règle pour le while :

$$\frac{\{I \wedge e \wedge v = n\}\ i\ \{I \wedge v \geq 0 \wedge v < n\}}{\{I\}\ \text{while}\ e\ \text{do}\ i\ \{I \wedge \neg e\}}\ \text{while}$$

où v est le variant (expression) et n une variable entière n'apparaissant pas dans i.

Un exemple

Avec boucle while

$$\dfrac{\dfrac{x \geq 0 \wedge x < 10 \Rightarrow x+1 \geq 0 \qquad \dfrac{}{\{x+1 \geq 0\}\ x := x+1\ \{x \geq 0\}}\ :=}{\{x \geq 0 \wedge x < 10\}\ x := x+1\ \{x \geq 0\}}\ \text{Aff}}{\{x \geq 0\}\ \text{while } x < 10 \text{ do } x := x+1\ \{x \geq 0 \wedge \neg(x < 10)\}}\ \text{while}$$

Mécanisation des preuves

Outils d'aide à la preuve

- Beaucoup d'outils existants ;
- Développés par des équipes de recherche ;
- Mais pas que : Atelier B (Alstom, ClearSy) ;
- Mécanisation ne signifie pas automatisation :
  - Outils de preuve interactive (Coq, HOL, etc.) ;
  - Outils de preuve automatique (Vampire, Zenon, etc.).

Transfert industriel ?

- Difficile au début ;
- Investit les milieux R&D progressivement ;
- Plus facile si l'outil vient d'une initiative industrielle (Atelier B) ;
- Plusieurs succès académiques récents changent la donne.

Quelques succès marquants

Ligne de métro 14 (Meteor)

- Ouverte en 1998 ;
- Développée par Siemens (Matra) ;
- Utilisation de la méthode B ;
- Théorie des ensembles en première page des journaux ;
- Siemens continue à développer des métros sans conducteur.

Quelques succès marquants

JavaCard (Gemalto)

- Formalisation de l'architecture JavaCard ;
- Utilisation de Coq.

Le compilateur certifié CompCert (Inria, Gallium)

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

Projet L4.verified (NICTA)

- Formalisation du micro-noyau seL4 ;
- Utilisation d'Isabelle/HOL.

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.

Outil d'aide à la preuve Coq

Caractéristiques

- Développement par l'équipe Inria πr^2 ;
- Preuve de programmes fonctionnels ;
- Théorie des types (calcul des constructions inductives) ;
- Isomorphisme de Curry-Howard (objets preuves).

Implantation

- Premières versions milieu des années 80 ;
- Implantation actuelle en OCaml ;
- Preuve interactive (peu d'automatisation) ;
- En ligne de commande ou avec l'interface graphique CoqIDE.

Outil d'aide à la preuve Coq

Distinctions

- « SIGPLAN Programming Languages Software Award » (01/2014) ;
- « ACM Software System Award » (06/2014) ;
- A servi à détecter une faille dans le logiciel OpenSSL en 2014 : http://ccsinjection.lepidum.co.jp/blog/2014-06-05/CCS-Injection-en/index.html