Des erreurs, des preuves, des caractères et un Coq

Épiphymath – Besançon, 12/9/2013

Jérôme Germoni (IREM de Lyon - UCBL)

Un événement scientifique

Site de l'équipe jointe Microsoft Research – INRIA :

The formalization of the Odd Order theorem has been completed on September 20th, 2012.

"This is really the End."

After a six year effort, members of the Mathematical Components team have completed an axiom-free formalization of the proof of the Odd Order theorem (due to Walter Feit and John Griggs Thompson), using the Coq proof assistant.

This team work was lead by Georges Gonthier [...].

Erreurs

Revue de presse de Bruno Duvic du 22 octobre 2012

≪ En témoigne l'erratum, (la correction) qui a été publié après un article [du New York Times] du 18 octobre et repéré par Rue89. L'article concernait Ahmed Abu Khattala, suspecté dans l'attaque contre le consulat américain de Benghazi. Le premier paragraphe racontait que, alors que les autorités américaines ont promis de le traduire devant la justice, on l'avait vu récemment siroter un jus de fruit dans un hôtel de luxe.
Erratum ! Ah bon ? On aurait accusé cet homme à tort ? Voici le texte publié par le New York Times.

Erreurs

Revue de presse de Bruno Duvic du 22 octobre 2012

≪ En témoigne l'erratum, (la correction) qui a été publié après un article [du New York Times] du 18 octobre et repéré par Rue89. L'article concernait Ahmed Abu Khattala, suspecté dans l'attaque contre le consulat américain de Benghazi. Le premier paragraphe racontait que, alors que les autorités américaines ont promis de le traduire devant la justice, on l'avait vu récemment siroter un jus de fruit dans un hôtel de luxe.
Erratum ! Ah bon ? On aurait accusé cet homme à tort ? Voici le texte publié par le New York Times. "Une version précédente de cet article décrivait de façon incorrecte la boisson que Ahmed Abu Khattala buvait dans un hôtel de Benghazi, en Libye. Il s'agissait d'une boisson frappée aux fraises et non d'un jus de mangue." ≫

Pas d'erreurs en mathématiques !

Notion de preuve, méthode hypothético-déductive, axiomes évidents et étapes logiques élémentaires : autant de garde-fous.

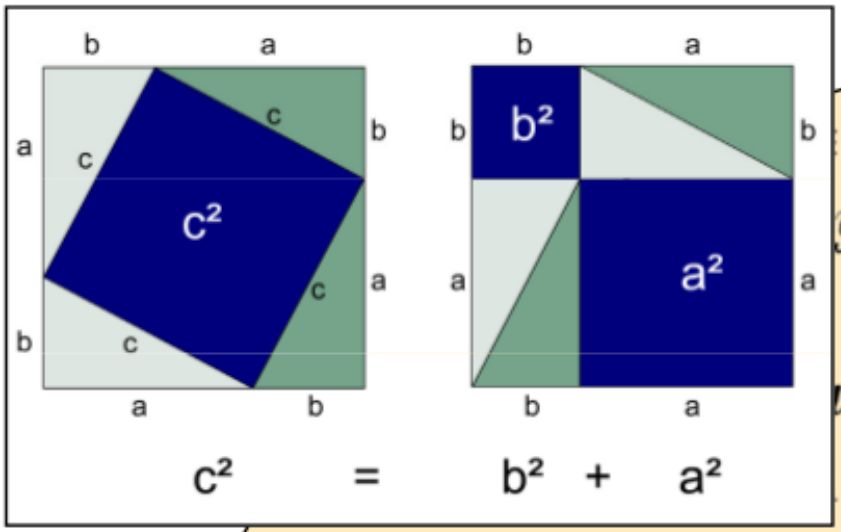
Pas d'erreurs en mathématiques ?

Notion de preuve, méthode hypothético-déductive, axiomes évidents et étapes logiques élémentaires : autant de garde-fous.

Et pourtant...

- erreur de Poincaré pour le prix du roi Oscar de Norvège (1890)
- erreur de Wiles dans la preuve du théorème de Fermat, corrigée par Wiles-Taylor (1993-1994),
- Amnon Neeman : "A counterexample to a 1961 'theorem' in homological algebra" (2002)

Preuve et niveaux de formalisation



$\alpha \cup \beta \epsilon 2$

$y . \supset : \alpha \cup \beta \epsilon 2 . \equiv . x \neq y .$

$\iota' x . \beta = \iota' y . \supset : \alpha \cup \beta \epsilon 2 .$

$\beta \epsilon 1 . \supset : \alpha \cap \beta = \Lambda . \equiv . \alpha \cup$

$6 . \supset \vdash :. \alpha = \iota' x . \beta = \iota' y . \supset :$

$] \vdash . (1) . *11{\cdot}11{\cdot}35 . \supset$

$\vdash :. (\exists x, y) . \alpha = \iota' x . \beta$

$(2) . *11{\cdot}54 . *52{\cdot}1 . \supset$

Supposons qu'il existe deux entiers positifs tels que le quotient du premier par le second soit égal à racine de deux : le carré du premier est égal au double du carré du second. L'exposant du nombre premier deux dans la décomposition en facteurs premiers du carré du

Fondements des mathématiques

- « Crise des fondements » des mathématiques (tournant du XX^{e}) ; axiomes d'Euclide-Hilbert, paradoxe de Russell
- Tentative de Poincaré (1902)
- Tentative de Russell – Whitehead
- « Rêve » de Hilbert (1928) : algorithme universel pour décider si une assertion est vraie ou fausse et en fournir une preuve
- Logique du premier ordre ; notion de *démontrabilité*, théorèmes de complétude et d'incomplétude de Gödel (1931)

Fondements des mathématiques

- « Crise des fondements » des mathématiques (tournant du XX^{e}) ; axiomes d'Euclide-Hilbert, paradoxe de Russell
- Tentative de Poincaré (1902)
- Tentative de Russell – Whitehead
- « Rêve » de Hilbert (1928) : algorithme universel pour décider si une assertion est vraie ou fausse et en fournir une preuve
- Logique du premier ordre ; notion de *démontrabilité*, théorèmes de complétude et d'incomplétude de Gödel (1931)
- Tentative de Bourbaki (1940) : en principe...

> « *L'usage exclusif des assemblages conduirait à des difficultés typographiques et mentales insurmontables. C'est pourquoi les textes courants utilisent des symboles abréviateurs (notamment des mots de la langue ordinaire).* »

Preuve ?

Une preuve est une suite d'arguments acceptés par un interlocuteur ou une communauté.
C'est un phénomène social qui dépend au moins :

- de l'époque (Cauchy et la continuité, Euclide et Hilbert...),
- du contexte (élève ou professionnel, oral ou écrit, article),
- du pays (Bourbaki - France/Perelman - Russie),
- de la discipline (géométrie/algèbre) !

Objectivité ? Validation d'une preuve ? Réfutation ?

Motivation : échecs célèbres

La confiance dans un *système logiciel* est traditionnellement fondée sur des tests.
Deux événements ayant stimulé la certification :

- « *Bug* de la division du Pentium » (1994) : nié par Intel puis minoré puis rappel de tous les processeurs
- explosion de la première fusée Ariane 5 (1996) : *bug* « mineur » d'un calculateur qui était présent sur Ariane 4 mais ne pouvait pas produire d'erreur
  (différence entre tests et certification !)

Preuve d'un algorithme : terminaison

```
Algorithme PGCD
Entree : a, b entiers
Sortie : un entier
Variables locales : x, y, r
 x := a ; y := b ;
 tant que y != 0 faire
      r := reste de la division de x par y
      x := y
      y := r  //
 renvoyer x
```

Preuve d'un algorithme : terminaison

```
Algorithme PGCD
Entree : a, b entiers
Sortie : un entier
Variables locales : x, y, r
 x := a ; y := b ;
 tant que y != 0 faire
      r := reste de la division de x par y
      x := y
      y := r   //nouvelle valeur de y < ancienne valeur
 renvoyer x
```

Preuve d'un algorithme : validité

Note $D(a, b)$ l'ensemble des diviseurs communs à a et b

```
Algorithme PGCD
Entree : a, b entiers
Sortie : un entier
Variables locales : x, y, r
  x := a ; y := b ; //
  tant que y != 0 faire
      r := reste de la division de x par y
      x := y                    //
      y := r                    //
  renvoyer x                    //
```

Preuve d'un algorithme : validité

Note $D(a, b)$ l'ensemble des diviseurs communs à a et b

```
Algorithme PGCD
Entree : a, b entiers
Sortie : un entier
Variables locales : x, y, r
  x := a ; y := b ; // D(a,b) = D(x,y)
  tant que y != 0 faire
      r := reste de la division de x par y
      x := y                    //
      y := r                    //
  renvoyer x                    //
```

Preuve d'un algorithme : validité

Note $D(a, b)$ l'ensemble des diviseurs communs à a et b

Algorithme PGCD
Entree : a, b entiers
Sortie : un entier
Variables locales : x, y, r
  x := a ; y := b ; // $D(a, b) = D(x, y)$
  tant que y != 0 **faire**
    r := reste de la division de x par y
    x := y // $x = qy + r, \quad 0 \leq r < y$
    y := r // $D(a, b) = D(x, y) = D(y, x - qy)$
  renvoyer x //

Preuve d'un algorithme : validité

Note $D(a, b)$ l'ensemble des diviseurs communs à a et b

Algorithme PGCD
Entree : a, b entiers
Sortie : un entier
Variables locales : x, y, r
  x := a ; y := b ; // $D(a, b) = D(x, y)$
  tant que y != 0 **faire**
    r := reste de la division de x par y
    x := y // $x = qy + r, \quad 0 \leq r < y$
    y := r // $D(a, b) = D(x, y) = D(y, x - qy)$
  renvoyer x // $D(a, b) = D(x, 0) = x$

Conséquence : démonstration de l'*existence* du PGCD de a et b.

Certification de preuve

But : faire certifier une preuve par un ≪ assistant de preuve ≫ – le logiciel Coq :

1. l'humain produit une démonstration et la code dans un langage compréhensible par Coq,
   $\rightsquigarrow$ preuve formelle (analogue de l'algorithme)
2. le logiciel vérifie chaque étape de la démonstration et vérifie que la preuve prouve bien ce qu'elle affirme
   $\rightsquigarrow$ certificat (analogue de la preuve de l'algorithme)

Vérifier la preuve

Mise en garde : il s'agit bien de certifier une preuve écrite par un être humain dans un langage symbolique et pas trouver la preuve.

L'expression ≪ preuve automatique ≫ est hors de propos.

En effet :

- 1931 : théorèmes d'incomplétude de Gödel
- 1936 : Church et Turing prouvent indépendamment ≪ qu'un tel algorithme ne pouvait exister. Avant même la construction des ordinateurs, le théorème d'indécidabilité de la démontrabilité fixait donc l'existence d'une limite intrinsèque à leur utilité en mathématiques ≫ (Dowek).

L'assistant de preuve Coq

Origines du nom :

- CoC : *calculus of constructions*...
- introduit par Thierry Coquand

- Cours de Coq dans certaines universités américaines pour apprendre la *preuve* aux étudiants en *informatique*
- Usages industriels : JavaCard (cartes à puce, Schlumberger), Esterel Technologies (transports, Défense, G. Berry)

L'assistant de preuve Coq

Origines du nom :

- CoC : *calculus of constructions*...
- introduit par Thierry Coquand

- Cours de Coq dans certaines universités américaines pour apprendre la *preuve* aux étudiants en *informatique*
- Usages industriels : JavaCard (cartes à puce, Schlumberger), Esterel Technologies (transports, Défense, G. Berry)

Écrit en OCaml (langage *fonctionnel*, ≪ implémente ≫ le λ-calcul)
Fondement : théorie des types : ≪ revanche de Russell ≫ (Gonthier)

L'assistant de preuve Coq

Origines du nom :

- CoC : *calculus of constructions*...
- introduit par Thierry Coquand

- Cours de Coq dans certaines universités américaines pour apprendre la *preuve* aux étudiants en *informatique*
- Usages industriels : JavaCard (cartes à puce, Schlumberger), Esterel Technologies (transports, Défense, G. Berry)

Écrit en OCaml (langage *fonctionnel*, ≪ implémente ≫ le λ-calcul)
Fondement : théorie des types : ≪ revanche de Russell ≫ (Gonthier)

Un exemple en direct !

Implication et abstraction

Énoncé : langage, connecteurs (et, ou, non), mutificateur ($\forall$).

1. On exprime les connecteurs avec l'implication $\Rightarrow$ (et la constante $\bot$) :

$$\begin{cases} \text{non}(A) & : & A \Rightarrow \bot, \\ A \text{ ou } B & : & \big((A \Rightarrow \bot) \Rightarrow B\big) \\ A \text{ et } B & : & \big(A \Rightarrow (B \Rightarrow \bot)\big) \Rightarrow \bot \end{cases}$$

2. Substitut à $\forall$: l'abstraction

$$\lambda x \;\; f(x) \qquad \big(\text{ou mieux : } \lambda(x : A) \cdot (f(x) : B)\big).$$

C'est, *a minima*, un moyen de mutifier les variables. Plus ?

Correspondance preuve-programme

Moralement : on identifie un énoncé mathématique avec l'ensemble de toutes ses preuves – les preuves d'un même énoncé sont toutes de même *type*, qui est l'énoncé en question.

Correspondance de Curry-Howard / preuve-programme

logique (λ-calcul)	programmation (fonctionnelle)
énoncé	type de données
preuve	programme

Correspondance preuve-programme (2)

Informatiquement : définition inductive des types.
Part de types « atomiques ».
Construction clé : si A et B sont deux types, $A \to B$ en est un.

Un terme t de type $A \to B$ est une *fonction* qui,
à tout terme a de type A, associe $t(a)$, un terme de type B.

Correspondance preuve-programme (2)

Informatiquement : définition inductive des types.
Part de types « atomiques ».
Construction clé : si A et B sont deux types, $A \to B$ en est un.

Un terme t de type $A \to B$ est une *fonction* qui,
à tout terme a de type A, associe $t(a)$, un terme de type B.
Donc, à une preuve a de A, elle associe une preuve $t(a)$ de B.

Correspondance preuve-programme (2)

Informatiquement : définition inductive des types.
Part de types « atomiques ».
Construction clé : si A et B sont deux types, $A \to B$ en est un.

Un terme t de type $A \to B$ est une *fonction* qui,
à tout terme a de type A, associe $t(a)$, un terme de type B.
Donc, à une preuve a de A, elle associe une preuve $t(a)$ de B.
Ainsi, t est une preuve de $A \to B$ (*i.e.* $A \Rightarrow B$).

Correspondance preuve-programme (2)

Informatiquement : définition inductive des types.
Part de types « atomiques ».
Construction clé : si A et B sont deux types, $A \to B$ en est un.

Un terme t de type $A \to B$ est une *fonction* qui,
à tout terme a de type A, associe $t(a)$, un terme de type B.
Donc, à une preuve a de A, elle associe une preuve $t(a)$ de B.
Ainsi, t est une preuve de $A \to B$ (*i.e.* $A \Rightarrow B$).

Morale : La ressemblance entre $\to$ et $\Rightarrow$ n'est pas fortuite !

Question (de recherche) : Que fait le programme qui correspond à la preuve du théorème de... ?

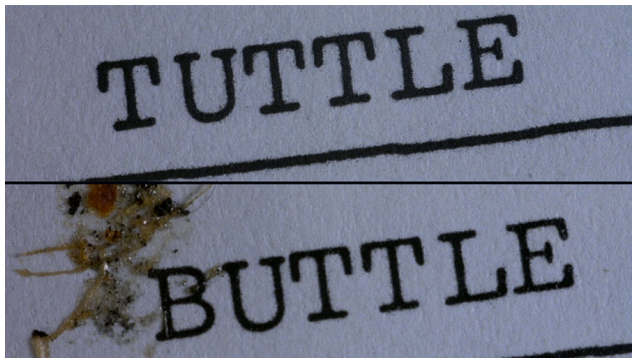
Et en pratique, quels résultats ?

État de l'art grossier :

- la plupart des théorèmes « classiques » ont été certifiés (par Coq ou autre), par exemple : Bézout, incomplétude de Gödel, courbe fermée de Jordan, nombres premiers...
- grand succès : quatre couleurs (Gonthier-Werner, 2004)
- grand succès : Feit-Thompson (Gonthier *et al.*, 2012)
- grand chantier : conjecture de Kepler (Hales, 90% ?)

Bugs

Pour Thomas Hales, tout programme contient au moins un *bug* toutes les 500 lignes de code.



Comment y remédier ?

- Coq est construit autour d'un cœur d'environ 500 lignes, scruté et testé dans de nombreuses situations ;
- le cœur de Coq peut certifier le reste de Coq.

Ce que l'on perd

- longueur d'une preuve formelle (bien nettoyée) : environ 1,5 fois la longueur d'une preuve rédigée
- coût en temps pour faire une preuve formelle !
  ex. : Feit-Thompson = 6 ans pour 5-10 personnes
- intuition...

Ce que l'on gagne

Apports plus ou moins évidents :

- confiance dans la preuve, parfois sans alternative (quatre couleurs, Kepler, Feit-Thompson)
- détection d'erreurs (de détail dans Feit-Thompson)
- « mise en évidence de ce qui fait marcher la preuve »
- mise au clair (nouveaux concepts, « simplifications »)
- travail sur la preuve comme objet mathématique : (ex. : arbre des dépendances)

Il semble y avoir plus...

Renouveau des mathématiques constructives

Année spéciale à “Univalent Foundations of Mathematics” à l’Institute of Advanced Studies (Princeton)

Thierry Coquand
(informatique)

Vladimir Voevodsky
(homotopie)

Steve Awodey
(catégories)

Apports de la « preuve formelle »

- une logique fondée sur l'implication

Apports de la « preuve formelle »

- une logique fondée sur l'implication
- prise en charge des registres de langage
  - « factorisation unique d'un entier » :

$$1100736 = 2^6 \times 3^3 \times 7^2 \times 13.$$

  - « forme canonique d'un polynôme du second degré »$(2a \neq 0)$:

$$ax^2 + bx + c = a\left(x + \frac{b}{2a}\right)^2 + \frac{4ac - b^2}{4a}$$

$\rightsquigarrow$ types riches qui prennent en compte plusieurs registres
(« la revanche de Russell »)

Apports de la « preuve formelle »

- une logique fondée sur l'implication
- prise en charge des registres de langage
  - « factorisation unique d'un entier » :

$$1100736 = 2^6 \times 3^3 \times 7^2 \times 13.$$

  - « forme canonique d'un polynôme du second degré »$(2a \neq 0)$:

$$ax^2 + bx + c = a\left(x + \frac{b}{2a}\right)^2 + \frac{4ac - b^2}{4a}$$

  $\rightsquigarrow$ types riches qui prennent en compte plusieurs registres (« la revanche de Russell »)
- traitement de l'égalité

Apports de la « preuve formelle »

- traitement de l'égalité : égalité stricte/à quelque chose près
  - corps fini de cardinal donné

    Rappel : corps $= (\mathbb{K}, +, \cdot, 0, 1)$, règles de calcul habituelles, inverses
    - Corps à 2 éléments : $\{0, 1\}$, $1 + 1 = 0$ forcé

Apports de la « preuve formelle »

- traitement de l'égalité : égalité stricte/à quelque chose près
  - corps fini de cardinal donné

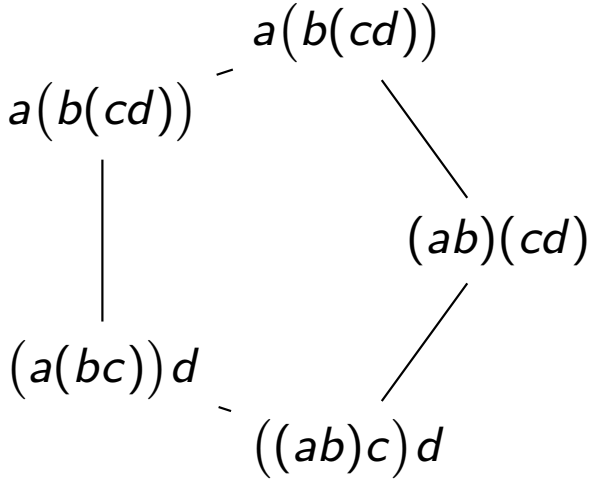
    Rappel : corps $= (\mathbb{K}, +, \cdot, 0, 1)$, règles de calcul habituelles, inverses
    - Corps à 2 éléments : $\{0, 1\}$, $1 + 1 = 0$ forcé
    - Corps à 4 éléments : $\mathbb{F}_4 = \{0, 1, \alpha, \beta\}$, $1 + 1 = 0$ forcé
      $\alpha + 1 \notin \{0, 1, \alpha\}$ donc $\alpha + 1 = \beta$
      X^2, $X^2 + X$ et $X^2 + 1$ ont deux racines : $\alpha^2 \notin \{1, \alpha\}$:
      $\alpha^2 = \alpha + 1$; de même : $\beta^2 = \beta + 1$
      $\alpha(\alpha + 1) = \alpha^2 + \alpha = 1$: $\beta = 1/\alpha$
      $F : \mathbb{F}_4 \to \mathbb{F}_4$, $x \mapsto x^2$ est un automorphisme
    - pour tout p premier et tout e entier, il existe un corps $\mathbb{F}_q$ de cardinal $q = p^e$, unique à isomorphisme près

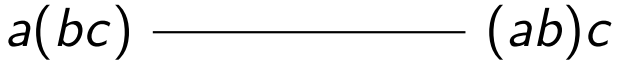
Apports de la « preuve formelle »

- traitement de l'égalité : égalité stricte/à quelque chose près
  - corps fini de cardinal donné

Apports de la « preuve formelle »

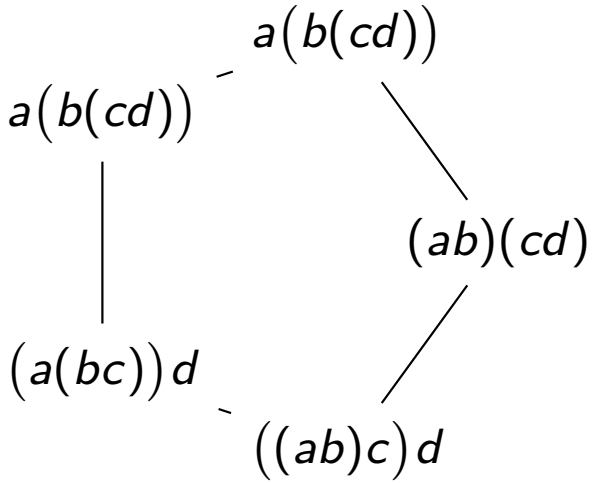
- traitement de l'égalité : égalité stricte/à quelque chose près
  - corps fini de cardinal donné
  - associativité (dans les catégories tensorielles...)

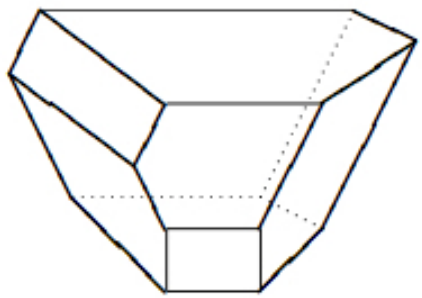


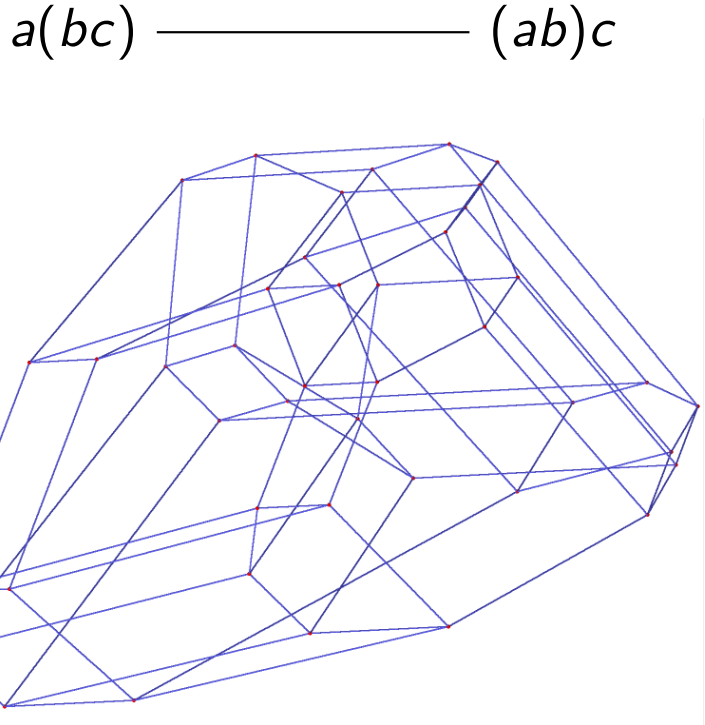


Apports de la « preuve formelle »

- traitement de l'égalité : égalité stricte/à quelque chose près
  - corps fini de cardinal donné
  - associativité (dans les catégories tensorielles...)







Résumé des épisodes précédents

Une preuve formelle est

- une traduction purement symbolique en langage informatique d'une preuve écrite par un être humain *et*
- un certificat de validité de cette preuve fournie par un logiciel spécialisé appelé assistant de preuve (Coq).

L'assistant vérifie et certifie *toutes* les étapes de la démonstration.
La confiance dans l'assistant de preuve vient :

- de l'examen et de tests approfondis du noyau
- de la validation du reste de l'assistant par le noyau.

Bilan : confiance nouvelle dans la preuve, très forte diminution du risque d'erreurs, pas de certitude philosophique.

Énoncé

Théorème de Feit-Thompson (1963)

Tout groupe fini dont le nombre d'éléments est impair est résoluble.

Walter Feit

John Griggs Thompson

Groupe

« Transformation » d'un ensemble X : application bijective de X dans X.

Définition d'un groupe

Un groupe est un ensemble G de transformations d'un ensemble X donné qui est stable par les opérations de composition et de passage à l'inverse.

Exemples :

- groupe $\mathfrak{S}_X$ de toutes les transformations de X,
- groupe $\mathrm{GL}_n(\mathbb{K})$ des matrices carrés inversibles sur $\mathbb{K}$.

Sous-groupe, sous-groupe distingué

Sous-groupe

Un sous-groupe H de G est une partie de G stable par composition et passage à l'inverse.

Sous-groupe distingué

Un sous-groupe H est distingué dans G si, pour tout $h \in H$ et tout $g \in G$, on a : $ghg^{-1} \in H$.

Notion analogue à la divisibilité des entiers : permet de construire un groupe quotient G/H.

Groupe simple

Un groupe est *simple* s'il n'admet pas de sous-groupe distingué.

Notion analogue à la notion de nombre premier chez les entiers.

Factorisation d'un entier

Soit $n \in \mathbb{N}^*$. Il existe des entiers $n_0 = 1$, $n_1, \ldots, n_r = n$ tels que pour tout $i \geq 1$:

- n_{i-1} divise n_i,
- $p_i = n_i / n_{i-1}$ est un nombre premier.

De plus, r est unique et la suite des p_i est unique à l'ordre près.

Exemple : $n = 6$: deux suites possibles ($r = 2$) :

- $n_0 = 1$, $n_1 = 2$, $n_2 = 6$: alors $p_1 = 2$, $p_2 = 3$;
- $n_0 = 1$, $n_1 = 3$, $n_2 = 6$: alors $p_1 = 3$, $p_2 = 2$.

Dévissage d'un groupe fini (Jordan-Hölder)

Soit G un groupe fini. Il existe des sous-groupes $G_0 = \{e\}$, $G_1, \ldots,$ $G_r = G$ tels que $S_i = G_i/G_{i-1}$ pour tout $i \geq 1$:

- G_{i-1} est distingué dans G_i,
- $S_i = G_i/G_{i-1}$ n'a pas de sous-groupe distingué.

De plus, r est unique et la suite des S_i est unique à l'ordre près.

Groupe résoluble

Un groupe est résoluble si tous les S_i sont *commutatifs* ($ss' = s's$ pour tous s et s' de S_i).

Représentations et caractères

Idée-clé de la preuve : étudier les représentations de G.
Sous-jacent : linéariser !

Représentation

Une représentation d'un groupe G est la donnée d'un morphisme $\rho : G \to \mathrm{GL}_n(\mathbb{K})$.
Le caractère de ρ est la fonction $G \to \mathbb{K}$, $g \mapsto \mathrm{tr}\rho(g)$.

Point-clé : si $\mathbb{K} = \mathbb{C}$, le caractère caractérise la représentation.

Théorème de Burnside

Si G a $p^\alpha q^\beta$ éléments, où p, q premiers, alors G est résoluble.

État de la preuve

Pas de doutes véritables dans la communauté :

- preuve originale de Feit-Thompson : 245 pages en 1963, dur !
- deux volumes parus chez Cambridge University Press
  1. *Local Analysis for the Odd Order Theorem*, H. Bender and G. Glauberman (préliminaires) (188 p., 1995)
  2. *Character Theory for The Odd Order Theorem*, T. Peterfalvi (preuve proprement dite) (162 p., 2000)

  qui confirment la validité de la preuve initiale (toujours dur !).

Motivations de Georges Gonthier :

- pas du tout lever un doute !

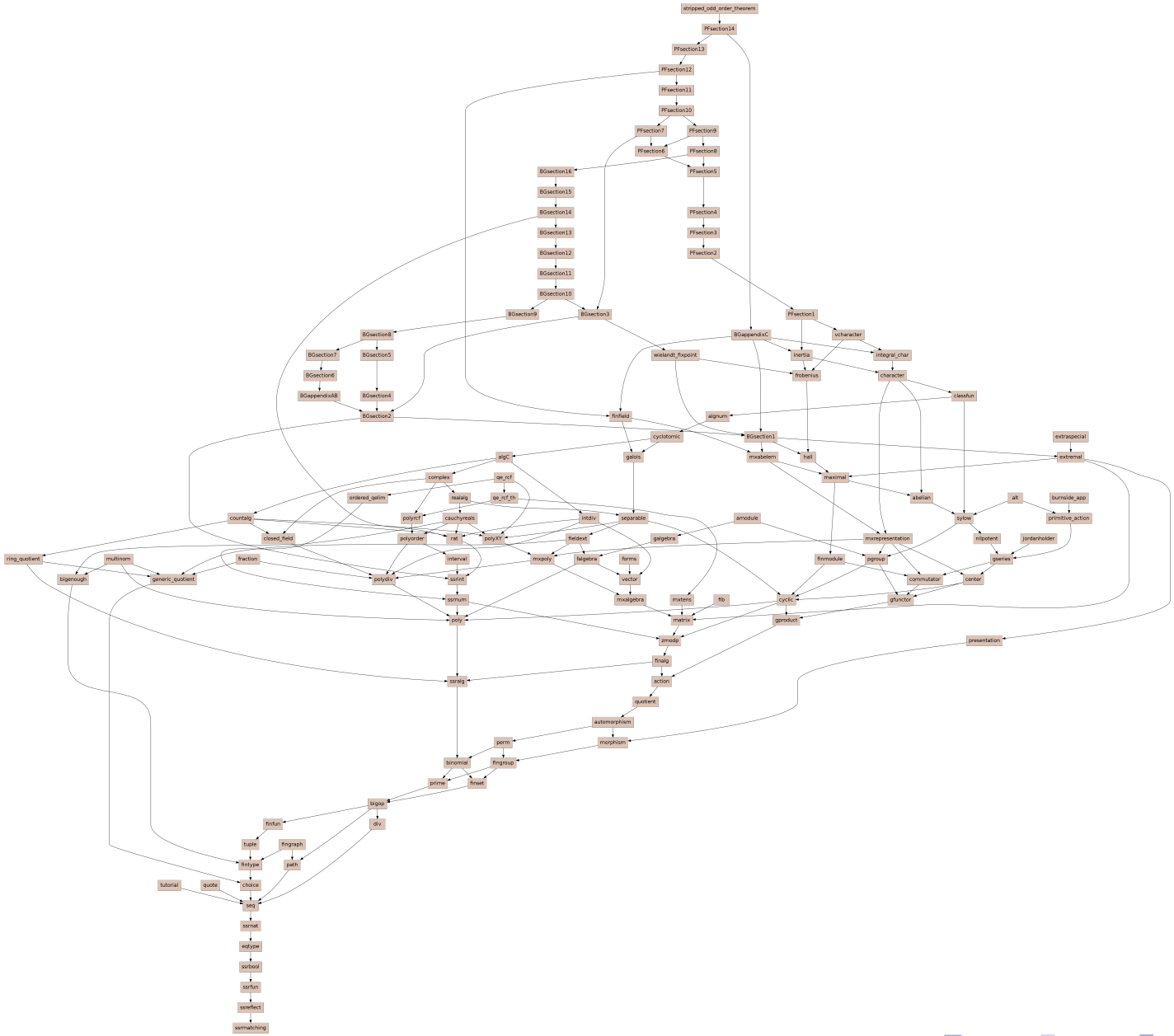
Motivations, quelques chiffres

Motivations de Georges Gonthier :

- faire un théorème *difficile* d'algèbre
  $\rightsquigarrow$ pas les mêmes difficultés que la certification logicielle
- qui touche à de nombreux domaines, utilise de nombreux outils
  $\rightsquigarrow$ produire des choses réutilisables

Quelques chiffres :

- 6 ans de travail, mais résultat atteint dans les temps annoncés !
- 170.000 lignes de code
- environ 2 h de calcul pour compiler la preuve
- 70 % du code produit : l'environnement (anneaux, corps, groupes, représentations, caractères, sous-groupes de Sylow...)


stripped_odd_order_theorem
PFsection14
PFsection13
PFsection12
PFsection11
PFsection10
PFsection7
PFsection9
PFsection6
PFsection8
BGsection16
PFsection5
BGsection15
BGsection14
PFsection4
BGsection13
PFsection3
BGsection12
PFsection2
BGsection11
BGsection10
BGsection9
BGsection3
PFsection1
BGsection8
BGappendixC
vcharacter
BGsection7
BGsection5
wielandt_fixpoint
inertia
integral_char
BGsection6
frobenius
character
BGappendixAB
BGsection4
classfun
BGsection2
finfield
algnum
cyclotomic
BGsection1
extraspecial
algC
galois
mxabelem
hall
extremal
complex
qe_rcf
maximal
ordered_qelim
realalg
qe_rcf_th
abelian
alt
burnside_app
countalg
polyrcf
cauchyreals
intdiv
separable
sylow
primitive_action
closed_field
polyorder
rat
polyXY
amodule
fieldext
galgebra
mxrepresentation
nilpotent
jordanholder
ring_quotient
multinom
fraction
interval
mxpoly
falgebra
forms
finmodule
pgroup
gseries
bigenough
generic_quotient
polyorn
sorint
vector
commutator
center
ssrnum
mxalgebra
mxtens
fib
cyclic
gfunctor
poly
matrix
gproduct
zmodp
presentation
finalg
ssralg
action
quotient
automorphism
morphism
perm
binomial
fingroup
prime
finset
bigop
finfun
div
tuple
fingraph
fintype
path
tutorial
quote
choice
seq
ssrnat
eqtype
ssrbool
ssrfun
ssreflect
ssrmatching

En guise de conclusion

Large éventail de niveaux de formalisation :

- peut-être un peu plus large que prévu côté informel,
- beaucoup plus large que prévu côté formel : on peut désormais pousser *en réalité* au niveau de détail ultime qu'on imaginait possible *en principe* (mais ça a un coût)

En guise de conclusion

Large éventail de niveaux de formalisation :

- peut-être un peu plus large que prévu côté informel,
- beaucoup plus large que prévu côté formel : on peut désormais pousser *en réalité* au niveau de détail ultime qu'on imaginait possible *en principe* (mais ça a un coût)

Compromis entre un langage évocateur (transmettre l'idée) et une formalisation complète (éviter l'erreur)

La façon qu'on a de faire/transmettre des mathématiques résulte d'un choix dont il vaut mieux être conscient

Quelques documents en ligne :

- Annonce officielle sur le site d'Inria (septembre 2012)
- J.-P. Delahaye, Du rêve à la réalité des preuves (juin 2012)
- G. Dowek, Une preuve formelle du théorème des quatre couleurs (2004)
- J. Germoni, Coq et caractères, Images des math. (novembre 2012)
- É. Ghys, Impossible, Images des math. (mai 2011)
- J.-L. Krivine : Fonctions, programmes et démonstrations (1994)
- B. Rittaud, La conjecture de Kepler démontrée à 99 % (2003)
- Wikipedia : Assistant de preuve, Coq, Feit-Thompson

Documents papier :

- *Les dossiers de La Recherche* **46** (novembre 2011)