La Vérité et la Machine

Benjamin Werner

La Vérité et la Machine

Benjamin WERNER*

Longtemps reservée aux informaticiens et logiciens, la vérification formelle de démonstration commence à être utilisée par une fraction grandissante de la communauté mathématique.

En décembre 2004, Georges Gonthier a annoncé l'achèvement de la formalisation complète de la preuve du théorème des quatre couleurs avec le logiciel Coq [Go]. Cette nouvelle a depuis été assez largement relayée par la presse scientifique de grand public et même généraliste ; c'est d'autant plus remarquable que la nature exacte de ce résultat est finalement relativement difficile à expliquer. Par ailleurs, plusieurs résultats mathématiques célèbres ont été revérifiés par des systèmes de preuve au cours de l'année écoulée, en particulier le théorème des nombres premiers ou le théorème de Jordan. On peut donc s'interroger s'il s'agit là d'une coïncidence chronologique ou du début d'un mouvement plus vaste.

Preuves formelles : historique

La quête de la correction

On considère en général que la logique mathématique moderne est née à la fin du XIX^{e} siècle, avec les travaux de logiciens tels que Frege, Peano, Zermelo ou Russell, qui ont contribué à la définition précise de formalismes tels que l'arithmétique, la théorie des ensembles ou les premières formes de la théorie des types. A partir de ce moment-là, on peut considérer une preuve mathématique comme étant elle-même un objet mathématique, dont la correction repose sur des règles syntaxiques, bien comprises et non ambiguës. Dans la plupart des formalismes, les preuves possèdent un structure d'arbre. Par exemple, étant donnés une preuve σ_A d'une proposition A et une preuve σ_B d'une autre proposition B, on peut les combiner pour construire une preuve de la proposition A *et* B ; on écrit la règle correspondante ainsi :

$$\frac{\dfrac{\sigma_A}{\vdash A} \qquad \dfrac{\sigma_B}{\vdash B}}{\vdash A \wedge B}$$

Le point de vue logique est donc qu'une proposition peut être déclarée vraie dans un certain formalisme si elle admet une preuve formelle vérifiant les règles de ce formalisme. Un texte mathématique traditionnel peut alors être vu comme une description informelle de cette preuve formelle ; le but est de convaincre le lecteur mathématicien de l'existence de cette preuve. C'est ainsi qu'au cours du siècle dernier, le consensus s'est fait autour de l'idée que la vérité mathématique était une notion exacte et objective.

Toutefois, dans la pratique mathématique, cet objet-preuve est longtemps resté virtuel. S'il est, *en principe*, possible d'écrire, ou « dessiner », une preuve entièrement formalisée, sa taille, c'est-à-dire le nombre d'étapes élémentaires de déduction, rend cette entreprise à peu près impossible *en pratique*, si la preuve n'est pas mathématiquement triviale. Qui plus est, il est vain de croire qu'aligner des symboles peu intuitifs sur la papier réduira de quelque façon

* *LIX, Ecole Polytechnique, 91 128 PALAISEAU cedex.*
Benjamin.Werner@inria.fr

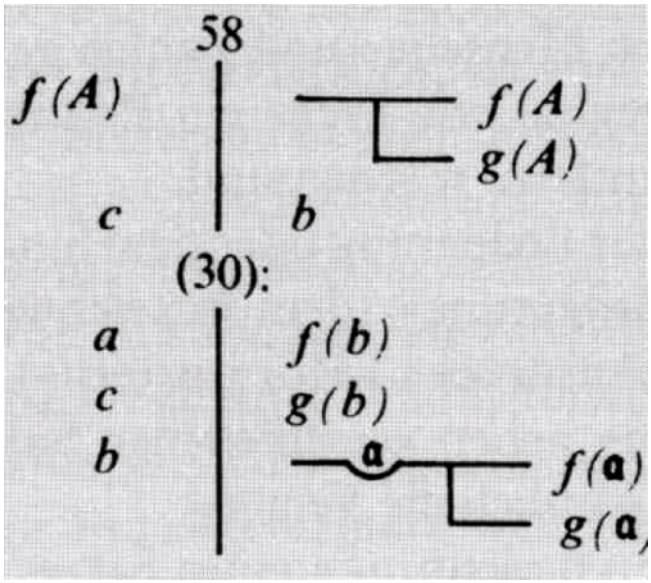

Figure 1 – *L'écriture mathématique de Frege (1872).*

que ce soit le risque d'erreur. En d'autres termes, la logique mathématique est, au départ, une discipline fondamentale et peu applicable destinée à être appliquée.

L'arrivée de l'ordinateur

Si un étudiant d'informatique d'aujourd'hui regarde les écrits de Frege, il ne peut être que frappé par ce qui lui apparaîtra comme la nature essentiellement informatique des notions développées. En effet, de part leur construction arborescentes, les propositions et les preuves sont typiquement des structures de données que les machines savent manipuler. L'ordinateur est le candidat idéal pour construire, stocker et surtout vérifier une preuve formelle.

Remarquons à ce titre, qu'il est important de distinguer les tâches de *construction* et de *vérification* de la preuve. En particulier, lors de la vérification, on ne cherche surtout pas à rendre l'ordinateur « intelligent ». Au contraire, c'est le manque d'imagination de la machine, sa précision mécanique voire bureaucratique, qui permettent d'accorder à une preuve vérifiée par ordinateur, un grand degré de certitude. C'est évidemment le but recherché.

Chronologiquement, le premier système de traitement de preuve fut le logiciel Automath développé par l'équipe de N.G. de Bruijn dans les années 1960. On peut considérer ce pionnier comme l'ancêtre commun des systèmes de preuves actuels. La plupart sont développés en Europe ou aux Etats-Unis ; on peut citer parmi d'autres Coq, Isabelle et HOL (U.E.) ou PVS (USA).

Des preuves de programmes aux preuves mathématiques

Les systèmes de preuve, comme tous logiciels, vivent d'abord à travers l'utilisation qui en est faite. Si à leur débuts, les textes mathématiques, on pourrait presque dire les classiques, ont été au centre des tout premiers travaux de formalisation en général effectués par les concepteurs mêmes, les centres d'intérêt des formalisateurs se sont un moment éloignés des mathématiques pures, en même temps que l'on découvrait un domaine d'applications privilégiés des « méthodes formelles » : la preuve de propriétés de programmes informatiques.

En effet, à l'image d'un objet mathématique, un programme obéit à des règles formelles et précisément définissables. On utilise donc un raisonnement de type mathématique pour garantir que telle ou telle logiciel (comme une inversion de matrices) aura bien telle ou telle propriété (sera involutive). Malheureusement, « à mains nues » cela se révèle rapidement assez malcommode. En effet, une telle preuve de correction peut certes faire appel à des mathé-

matiques tout-à-fait subtiles et non-élémentaires ; mais la pratique montre que raisonner à propos d'un programme se révèle alors bien souvent plus bureaucratique et fastidieux que de démontrer un théorème. C'est en grande partie du au fait qu'un programme est lui-même un objet formel et le fait qu'il soit correct « dans les grandes lignes » n'empêche pas qu'une petite erreur suffise à faire échouer matériellement le programme.

La vérification de programmes est donc un domaine où :

1. La compréhension ou l'intuition du raisonnement est souvent moins utile que la pure rigueur ;
2. dont l'utilité des applications est évidente.

On comprend donc que la certification de programmes a fait l'objet de nombreux travaux théoriques et pratiques. On a donc intégré aux systèmes de preuves les techniques et les outils permettant de raisonner à propos de ces objets particuliers que sont les programmes.

Il est donc d'autant plus intéressant d'observer actuellement un certain retour de l'activité de formalisation vers les mathématiques. Une première raison est sans doute que les progrès de ces systèmes les rendent plus confortables et efficaces, et donc aussi plus intéressants pour les mathématiciens. Il semble toutefois qu'il y ait d'autres raisons plus profondes à ce mouvement, liées à la nature d'une partie des mathématiques contemporaines. C'est ce que nous allons essayer de décrire sommairement ici.

La question du calcul

Indépendamment de la question de la formalisation, le calcul informatique joue un rôle de plus en plus important dans un certain nombre de domaines mathématiques. Un exemple radical est la question des preuves de primalité : il ne viendrait à l'idée de personne de « prouver » une proposition comme « $2^{25964951} - 1$ est premier » sans recourir au calcul électronique. Si l'on veut toutefois réellement produire une preuve, la meilleure chose que l'on puisse espérer c'est de prouver que le programme utilisé établit bien la primalité. Autrement dit, si les mathématiques sont nécessaires pour prouver la correction de programmes, les programmes peuvent intervenir à leur tour dans des preuves.

Bien sûr, la propriété de primalité est directement liée au calcul ; mais il est aussi des théorèmes dont l'énoncé n'est pas essentiellement calculatoire et dont les seules preuves connues reposent pourtant sur des calculs importants, numériques ou symboliques. De ceux-là, le plus célèbre est sans doute le théorème des quatre couleurs. Ce dernier a été rejoint par la preuve de la conjecture de Kepler, également célèbre. De fait, à travers de tels résultats, l'ordinateur s'est invité au menu des mathématiciens et il importe de comprendre le statut des preuves qui l'utilisent.

La particularité des quatre couleurs

Le théorème des quatre couleurs dit, rappelons-le, qu'il est toujours possible de colorier une carte planaire, avec une couleur par pays, de telle manière que deux pays ayant une frontière commune ne soit pas de la même couleur. En général, on commence par ramener l'énoncé à la quatre-coloriabilité de graphes planaires (chaque pays correspondant alors à un sommet du graphe).

La suite de l'histoire est assez connue. Conjecturée en 1852, cette proposition simple et remarquablement « concrète » a défié pendant plus d'un siècle les efforts des nombreux mathématiciens, fameux ou anonymes, qui ont cherché à la démontrer. L'aura quelque peu mystérieuse de ce théorème auprès du grand public se renforçant encore lorsque la première démonstration fut annoncée en 1976, car cette dernière « faisait appel à l'ordinateur ».

Sans chercher, bien sûr, à exposer les détails de la preuve, il est intéressant de comprendre quel genre de tâche doit être dévolue à l'ordinateur. La preuve commence par restreindre le problème à une classe de graphes planaires triangulés appelés quasi 6-connexes. Ensuite, on se donne une liste de petits graphes particuliers appelés configurations. Dans la preuve originelle, on en comptait 1476, nombre ramené à 633 dans une preuve de 1995. On peut alors montrer que dans tout graphe triangulé et quasi 6-connexe apparaît au moins un de ces configurations ; c'est la propriété dite d'inévitabilité. Il faut noter que même si cette étape de la preuve est, pour des raisons évidentes particulièrement fastidieuse, elle reste à la portée d'une équipe de mathématiciens. De fait elle fut démontrée « à la main » pour la preuve de 1976.

On raisonne ensuite par récurrence sur le nombre de sommets du graphe. En se donnant un graphe triangulé et quasi 6-connexe, on sait donc qu'il contient une configuration. On peut alors « ôter » cette configuration (en fait la remplacer par un sous-graphe plus petit) et 4-colorier le graphe obtenu par hypothèse de récurrence. Il resterait alors à étendre ce coloriage du reste de la carte à la configuration pour conclure. Las, il n'est en général pas possible de trouver un coloriage de la configuration qui corresponde au coloriage du reste de la carte. C'est là qu'intervient l'explosion combinatoire de la preuve : en analysant, pour chaque configuration l'ensemble de tous les coloriages possibles, on arrive à montrer qu'il est toujours possible de *réorganiser le coloriage* du reste de la carte pour obtenir un coloriage qui s'étende à la configuration.

Il faut pour cela considérer l'ensemble des coloriages de chaque configuration (jusqu'à 20 000 pour une configuration), mais aussi l'ensemble des appareillements par composantes bicolores (jusqu'à 1 500 000). Même pour un ordinateur moderne utilisant un algorithme efficace, cela reste un calcul non-trivial. Cette partie est évidemment hors de portée d'un humain ou d'un groupe d'humains. On ne peut que faire confiance à la machine.

Un langage commun

La formalisation de la preuve des quatre couleurs en Coq signifie donc deux choses. D'une part on a prouvé formellement la correction des programmes utilisés pour vérifier la réductibilité des configurations. Mais surtout, on a pu, dans le même langage, celui de Coq, effectuer l'ensemble de la preuve, c'est-à-dire :

1. construire la théorie des graphes planaires et prouver, formellement, une série de lemmes et théorèmes,
2. écrire les programmes nécessaires à démontrer, formellement, le lien entre ces programmes et la propriété de réductibilité,
3. vérifier la preuve des quatre couleurs, processus qui fait intervenir les propriétés (1) et l'exécution des programmes (2).

On peut donc dire que c'est la première fois qu'une preuve du théorème des quatre couleurs est entièrement écrite : les textes existants jusqu'à maintenant ressemblaient à des textes mathématiques, jusqu'à la preuve de reductibilité. Il fallait ensuite passer des mathématiques à l'informatique en changeant de langage et donner un programme informatique ; on laissait le lecteur s'assurer qu'un résultat positif de l'exécution du programme signifiait bien l'existence d'une preuve.

Le langage de Coq évite cette rupture, puisqu'il inclut à la fois déduction et programmes. L'encadré 2 donne quelques détails sur la manière dont calcul et raisonnement peuvent s'articuler dans un formalisme comme celui de Coq.

Pour résumer la situation des preuves calculatoires, on peut donc dire d'abord que l'on a affaire à des preuves dont la vérification est hors de la portée de l'esprit humain sans assistance extérieure, et ce pour des raisons quasiment « mécaniques ». Le premier problème que cela pose est celui du langage dans lequel décrire ces preuves, puisque ce dernier doit inclure à la fois le discours mathématique traditionnel, mais aussi la possibilité de décrire des algorithmes informatiques. Les systèmes de preuve calculatoires proposent donc de tels langages hybrides, mais ils répondent aussi à la seconde interrogation à propos de l'interface entre ces deux modes : comment être sûr que le programme donné à bien les propriétés attendues, c'est-à-dire que tel ou tel résultat d'un calcul implique bien telle ou telle proposition mathématique (*cf* encadré).

D'un coté, on peut espérer que l'alliance entre raisonnement et calcul ouvre une nouvelle voie aux mathématiques : elle met à notre portée des résultats qui pour de simples raisons « mécaniques » (longueur de la preuve) étaient inaccessibles jusqu'à maintenant. On peut même imaginer l'appropriation par les mathématiciens d'un nouveau mode de recherche plus expérimental. Mais l'introduction de calculs informatiques rend plus difficile la vérification de ces résultats en même temps qu'est perdue une certaine forme d'intuition. La formalisation et la vérification informatique apparaissent comme le meilleur moyen de garantir la correction de ces nouvelles constructions mathématiques. Un atout étant que, comme évoqué plus haut, la vérification de logiciels est historiquement l'un des premiers débouchés pour la formalisation.

D'un autre coté, formaliser ces raisonnements d'un type nouveau nécessite une complexification du langage mathématique, puisqu'il faut lui incorporer un langage de programmation. Or, même si ce langage peut être plus épuré que celui qui sert à programmer un jeu vidéo, il se pose néanmoins la question du standard. On se trouve déjà dans une situation de concurrence entre plusieurs formalismes et plusieurs systèmes de preuve et la question est

ouverte de savoir lesquels pourront s'établir comme des standards pour la communauté mathématique et par quels processus. On peut toutefois penser que le défi pour les développeurs des systèmes de preuve est de les rendre suffisamment pratique pour les faire adopter au moins par les mathématiciens dont le domaine de recherche gagnerait à être formalisé. Pour les mathématiciens, il s'agira d'apprivoiser les systèmes de preuves, techniquement mais aussi socialement, en les intégrant aux critères de publication existants.

Conjecture de Kepler ou théorème de Hales ?

La question de la crédibilité des preuves mêlant raisonnement et calcul s'est reposée de manière brûlante voici peu de temps. Pendant vingt ans, la preuve du théorème des quatre couleurs est restée unique en son genre, et l'emploi important qu'elle faisait du calcul a surtout contribué à lui donner une aura particulière. Mais l'Histoire s'est répétée une première fois en 1998, lorsque Thomas Hales a proposé une preuve de la célèbre *conjecture de Kepler*. Là encore il s'agit d'une conjecture ancienne, puisqu' énoncée en 1611 et qui peut être expliquée dans un langage non-mathématique : il n'y a pas de façon de ranger des oranges qui soit meilleure que celle des étals des marchés.

A l'image de celle du théorème des quatre couleurs, la preuve de Hales repose sur des calculs informatiques importants. Elle est toutefois nettement plus complexe, d'une part parce que ces calculs interviennent à plusieurs niveaux dans la preuve, et surtout parce que la partie « conventionnelle », c'est-à-dire non calculatoire de la preuve est elle aussi plus longue et utilise plus de résultats pré-existant que celle des quatre couleurs.

Thomas Hales a soumis une série d'articles décrivant sa preuve en 1999, mais ces derniers ne paraîtront qu'en 2005 ; les relecteurs ne sachant pas comment traiter les parties calculatoires. Qui plus est, ces articles seront accompagné d'une mise en garde de l'éditeur précisant que ces parties informatiques de la preuve n'ont pas pu être vérifiées. On peut voir là le reflet d'une certaine réticence vis-à-vis de l'utilisation d'outils informatiques, mais il est indéniable que ce type de preuves est difficile à valider avec les procédures habituelles. D'une part parce que leur lecture complète nécessitent des compétences en programmation. Mais aussi parce que même pour un informaticien professionnel, la correction d'un programme est difficile à établir, pour les raisons déjà mentionnées.

Depuis cette relative mésaventure, Thomas Hales est devenu un ardent défenseur des mathématiques formelles. Qui plus est, ayant eu vent de l'effort de formalisation de la preuve des quatre couleurs, il a lancé un projet de formalisation de sa propre preuve qu'il a intitulé Flyspeck. Toutefois, il faudra sans doute attendre longtemps l'achèvement de ce projet. D'une part à cause de la complexité de la preuve elle-même, mais aussi parce que la littérature mathématique sur laquelle elle repose n'est pas encore formalisée.

Encadré 1

On peut prendre un exemple simple pour illustrer l'articulation preuve/calcul dans un formalisme moderne. Il est facile d'écrire un programme `test` *qui prend un nombre entier n en entrée et va essayer de le diviser par tous les nombres entiers compris entre 2 et* $n-1$. *Ce programme rendra* `true` *s'il ne trouve pas de diviseur et* `false` *sinon. Dans le formalisme, ce programme est une fonction des entiers vers les booléens. On peut alors facilement prouver le « théorème » suivant :*

$$\forall n.\texttt{test}(n) = \texttt{true} \Longrightarrow \texttt{premier}(n)$$

c'est-à-dire que ce programme est effectivement un test correct pour la primalité.
Si on applique ce théorème à, par exemple, 1789, on obtient une preuve de

$$\texttt{test}(1789) = \texttt{true} \Longrightarrow \texttt{premier}(1789).$$

Pour déduire que 1789 est premier, il nous suffit donc d'exhiber une preuve de $\texttt{test}(1789) = \texttt{true}$. *Or comme le programme* $\texttt{test}(1789)$ *rend la valeur* `true` *(on dit aussi qu'il s'évalue vers* `true`*), les objets mathématiques* $\texttt{test}(1789)$ *et* `true` *sont identifiés. Par congruence, cela veut dire que la proposition* $\texttt{test}(1789) = \texttt{true}$ *est elle identifiée à* $\texttt{true} = \texttt{true}$ *qui est une conséquence immédiate de la réflexivité de l'égalité.*
Pour prouver la primalité de nombres plus grands par la même méthode, il suffit alors d'utiliser des programmes moins naïfs et plus efficaces que `test`.

Encadré 2

La programmation fonctionnelle

La programmation fonctionnelle, à l'image de sa cousine la programmation orientée-objet, est un style de programmation commun à un certain nombre de langages de programmation. Les plus connu en France sont sans doute les différents dialectes de Caml, qui est aujourd'hui enseigné en classes préparatoires, mais on peut également citer Scheme, Standard ML ou Haskell. L'appellation de ces langages vient de ce qu'ils manipulent indifféremment des structures informatiques usuelles comme des entiers, chaînes de caractères ou tableaux, que des fonctions opérant sur ces structures. On peut ainsi définir une fonction prenant d'autres fonctions comme arguments.
*Ces langages permettent souvent une programmation claire et concise. Mais aussi, ils ont une nature plus mathématiques que les langages de programmation plus traditionnel. En effet, pour ces derniers, les programmes sont composés d'*instructions *dont l'exécution se traduit par une modification de la mémoire de l'ordinateur. C'est pourquoi on parle alors de langage* impératifs.
En programmation fonctionnelle, l'unité de base est la fonction et on laisse l'environnement d'exécution de l'ordinateur gérer l'espace mémoire pour calculer les valeurs de ces fonctions. De l'extérieur, un tel programme apparaît donc comme entièrement décrit par la manière dont il associe des valeurs aux arguments, exactement à l'image de la notion mathématique de fonction.
L'avantage pour les logiciens est qu'il devient possible de construire un formalisme logique dont les objets sont ces fonctions définies par un programme. De plus, ces programmes étant exécutables, l'expression 2+2 se simplifiera automatiquement en 4.

En sus de cet ambitieux projet, Hales a récemment annoncé et publié une preuve du théorème de Jordan, entièrement formalisée dans le système HOL-light.

Le monstre et la machine

Si l'ordinateur semble l'outil adapté pour dompter les monstres engendrés par les mathématiques contemporaines, ces derniers ne doivent pas tous leur caractère particulier au seul calcul. L'un des résultats les plus singuliers de l'époque récente est évidemment la classification des groupes finis. Il s'agit là d'un résultat mathématique que l'on peut dire « conventionnel » de part son énoncé comme de sa preuve, si ce n'était leurs tailles : la preuve a été estimée à 15 000, dispersées dans la littérature mathématique. Il paraît donc là aussi difficile de parler d'intuition ou d'évidence cartésienne à propos d'un tel ensemble de preuves et une formalisation de cet important corpus serait sans doute un apport à ce délicat édifice[1].

On peut espérer voir naître de tels efforts de formalisation dans un futur très proche. Il sera particulièrement intéressant d'en observer les progrès, car ce défi recouvre deux points importants et qui restent aujourd'hui encore particulièrement délicats à maîtriser. Le premier est qu'un tel travail ne pourrait être que collectif, nécessiterait donc de coordonner un important travail d'équipe autour d'un développement. Le second concerne la question des bibliothèques.

Une question de style

Si la communauté mathématique témoigne d'un intérêt grandissant pour la vérification informatique du raisonnement on doit aussi se poser la question des freins à cette évolution. Un premier obstacle à l'utilisation d'un système de preuve est la nécessité d'apprendre le langage avec lequel les preuves sont communiquées à l'ordinateur, mais si cela demande un certain investissement initial, c'est en général un obstacle surmontable particulièrement pour des mathématiciens rompus au langage algébrique. Un facteur plus critique actuellement reste la difficulté à construire des bibliothèques de développements mathématiques formels réellement pratiques et réutilisables. De fait, on observe qu'au début d'un développement important, les utilisateurs préfèrent encore souvent repartir de zéro plutôt que de commencer à construire sur des définitions ou des lemmes repris sur un autre travail. Ce phénomène est identifié depuis longtemps en programmation, où l'on parle souvent de la question de la « réutilisabilité du code ».

1. A une échelle plus humaine, il faut mentionner les travaux en géométrie algébrique comme ceux de Carlos Simpson. La complexité croissante de certains de ses énoncés l'ont conduit à formaliser ses travaux en Coq.

Si cette réutilisabilité ne va pas forcément de soi en mathématiques formelles, c'est que celle-ci sont particulièrement sensibles aux détails des définitions et des énoncés. Dans un texte traditionnel, on utilisera la notion de polynôme indifféremment pour désigner une classe particulière de fonctions réelles, ou une classe d'expressions algébrique (à savoir les sommes de monômes). De plus, on regardera le même polynôme parfois comme une fonction et parfois comme une expression algébrique. Si l'on travaille formellement, la fonction et l'expression sont deux objets distincts ; on peut certes définir les traductions de l'une vers l'autre, mais chaque invocation de ces traductions alourdira énoncés et preuves.

Or la lourdeur est l'écueil principal de la preuve formelle. Dans les faits, lorsqu'un développement ne peut être mené à son terme, c'est en généralement parce qu'à partir d'un moment il risque de crouler sous le nombre d'hypothèses locales, le nombre de « petits lemmes » ou le nombre d'étapes « triviales ».

Pour éviter cela, la meilleure arme reste un choix judicieux des définitions et des énoncés, qui permettra la preuve la moins bureaucratique possible, c'est-à-dire la plus élégante. En d'autre termes, et c'est sans doute une bonne nouvelle, tout en étant, comme on le lui demande, un vérificateur sans âme, l'ordinateur devient également un garant du beau style mathématique.

Pour en savoir plus

[Coq] Le Système Coq, Diffusé par l'INRIA, http ://coq.inria.fr

[Go] Gonthier (G.), A computer-checked proof of the Four Color Theorem. Disponible sur http ://research.microsoft.com/ gonthier/

[Hales] Hales (T.), divers articles et documents sur http ://www.math.pitt.edu/ thales/