Tiers exclu et choix dépendant

Jean-Louis Krivine

Université Paris VII, C.N.R.S.

21 octobre 2005

Colloque D. Lacombe, I.H.P. Paris

Je vais vous parler de logique et de mathématiques, car c'est ce que je connais le mieux vu que c'est ma principale occupation depuis maintenant une bonne cinquantaine d'années. En logique, on s'intéresse essentiellement aux démonstrations mathématiques, car le problème originel est de comprendre ce que c'est. Je me suis, bien sûr, posé cette question quand j'étais étudiant, dans les années 60, et cela m'a amené au séminaire Destouches, ici même à l'IHP, où j'ai rencontré Daniel Lacombe et Bernard Jaulin. C'était sans doute le seul endroit à Paris où on faisait de la logique.

Le bilan est très positif, en ce qui me concerne, car le problème est résolu : je sais ce qu'est une démonstration mathématique et je vais essayer de vous l'expliquer. Je ne suis pas bien sûr d'y arriver, non que je doute de vos capacités intellectuelles, mais comment expliquer en une demi-heure ce qu'on a mis cinquante ans à comprendre ? Ça vaut quand même la peine d'essayer de vous donner quelques idées sur un sujet qui est passionnant.

Le titre de mon exposé, « Tiers exclu et choix dépendant » se réfère à deux axiomes qui ont joué un rôle particulièrement important dans cette histoire, cela pour deux raisons. D'abord ils ont fait couler beaucoup d'encre au début du siècle dernier. Ensuite, ce sont eux qui ont opposé la plus farouche résistance à la solution du problème dont nous parlons : comprendre ce qu'est une démonstration. Mais en même temps, ils servent de pierre de touche : une fois qu'on a compris ce qui se passe pour ces deux axiomes, la situation devient parfaitement claire. Malheureusement, c'est un sacré morceau à digérer.

Le tiers exclu est un axiome du calcul des propositions. Il affirme « A ou non A » pour chaque proposition « A ». Brouwer est le premier à avoir compris qu'il était tout à fait à part dans le calcul propositionnel et qu'il fallait absolument l'isoler des autres axiomes et règles de ce calcul qui sont (dans le système de Hilbert) :

les axiomes $A \to (B \to A)$; $(A \to (B \to C)) \to ((A \to B) \to (A \to C))$;

le modus ponens, à savoir « de $A \to B$ et de A, on déduit B ».

Il s'est intéressé, à juste titre, à ce qu'on pouvait montrer sans utiliser cet axiome. On dit que de telles démonstrations sont « intuitionnistes ». Mais, comme toutes les idées importantes qui arrivent un peu trop tôt, l'intuitionnisme a été incompris puis marginalisé par les mathématiciens et même les logiciens. Brouwer lui-même y a sans doute pas mal contribué par son dogmatisme. Pour mieux isoler le tiers exclu des autres axiomes, il l'a accusé de peste aviaire et interdit formellement à ses disciples de seulement s'en approcher. Du coup, l'intuitionnisme s'est peu à peu transformé en secte et stérilisé. C'est l'arrivée en force de l'informatique qui a, dans

un premier temps, réveillé l'intérêt pour la logique intuitionniste. Mais cela s'est fait sur un malentendu : on a cru, pendant longtemps (jusque dans les années 90) que les seules preuves « constructives », c'est-à-dire susceptibles de donner lieu à des calculs, étaient les preuves intuitionnistes. Voilà, pensait-on, pourquoi la logique intuitionniste est intéressante en informatique (théorique). Mais ce sont les progrès de l'informatique (pratique) dans le domaine des langages de programmation qui ont finalement permis de comprendre l'axiome du tiers exclu et de l'interpréter correctement. Il faut noter que le premier à l'avoir fait est un informaticien, Tim Griffin, et non pas un logicien. On donnera plus de détails tout à l'heure.

L'autre axiome auquel je vais m'intéresser ici est l'axiome du choix dépendant. Il est beaucoup plus compliqué à énoncer que le tiers exclu, car c'est un axiome de la théorie des ensembles. Il est fondamental dans la partie des mathématiques qu'on appelle « l'Analyse », qui est un très vaste domaine : théorie des fonctions de variable réelle ou complexe, théorie de la mesure et probabilités, équations différentielles et aux dérivées partielles, etc. Il exprime que si, pour tout réel r il existe un réel s tel que l'on ait $P(r,s)$ ($P(r,s)$ est une proposition quelconque qui parle de r et s), alors il existe une suite de réels $r_0, r_1, \ldots, r_n, \ldots$ telle que l'on ait $P(r_n, r_{n+1})$ pour tout entier n. Bien entendu, il est intuitivement évident, mais c'est la moindre des choses pour un axiome.
Lui aussi a été le sujet d'une vive discussion au début du siècle dernier, entre Baire, Borel, Hadamard et Lebesgue. Mais il n'en est pas résulté la constitution d'une secte s'opposant à l'usage de cet axiome !

On reviendra un peu plus tard sur ces deux axiomes. Reprenons maintenant le problème de la nature des démonstrations mathématiques, qui est comme je le disais tout à l'heure, le problème essentiel de la logique. Pour le résoudre, les logiciens sont arrivés à la formalisation complète de ces démonstrations au moyen d'un ensemble très restreint de règles et d'axiomes. Ça a été un travail de très longue haleine, commencé avec Aristote et Euclide et qui s'est terminé avec Zermelo et Frænkel, en passant par Frege, Russell et le théorème de complétude de Gödel.
Pour ce qui est de l'Analyse, que l'on peut formaliser en logique du second ordre, on a le système suivant :

• Les règles et axiomes du calcul propositionnel que j'ai données tout à l'heure. Il y a la partie intuitionniste et l'axiome du tiers exclu. Notons une formulation équivalente du tiers exclu, qu'on appelle la *loi de Peirce* ou le raisonnement par l'absurde et qui est $(\neg A \to A) \to A$.

• Les règles et axiomes pour les quantificateurs du premier et du second ordre, à savoir :

• La règle de généralisation : si on a montré une formule $F[c]$, alors on a montré $\forall x\, F[x]$; c est un symbole de constante du premier ou du second ordre, *qui n'apparaît pas dans la formule* $F[x]$.
• L'axiome de particularisation, qu'on appelle aussi « élimination du quantificateur universel » :
au premier ordre, $\forall x\, F[x] \to F[t]$, où t est un terme quelconque ;
au second ordre, $\forall X\, F[X] \to F[\Phi]$ où X est une variable de prédicat d'arité arbitraire, et Φ une formule qui a le bon nombre de variables libres. Cet axiome est aussi appelé « schéma de compréhension », car il permet de définir un ensemble d'entiers par une propriété quelconque de ses éléments.

• Et enfin l'axiome du choix dépendant.

Le fait d'avoir découvert un système complet de règles et d'axiomes pour l'Analyse (et même pour la théorie des ensembles, mais je n'en parlerai pas ici) était évidemment une grande

réussite. On pouvait dorénavant formaliser et vérifier n'importe quelle preuve d'analyse, en fait n'importe quelle preuve de mathématiques. Cela est resté assez théorique pendant longtemps, car la formalisation d'une démonstration est un travail long et pénible et le résultat est, en général, illisible. Mais, là encore, l'arrivée de l'informatique et des programmes d'aide interactive à la preuve, ont changé les choses.

Toutefois, ce ne peut être là le fin mot de l'histoire. En effet, que savons-nous au bout du compte ? Eh bien, que prouver un théorème consiste à écrire un texte qui doit suivre des règles absolument rigoureuses et se terminer par l'énoncé du théorème en question. Mais pourquoi donc se fatiguer tellement à prouver des théorèmes, s'il ne s'agit que d'un jeu d'écriture formel ? Il faut croire que derrière ce jeu d'écriture se cache un enjeu très important, car des générations et des générations de mathématiciens se sont livrés à cette occupation depuis plus de deux mille ans et ça continue de plus belle. Mais, me direz-vous, c'est parce que les mathématiques ont des applications extraordinaires. Par exemple, le monde de la physique est régi par des équations aux dérivées partielles et par les propriétés des opérateurs dans l'espace de Hilbert : gravitation et loi de Newton, équations de Maxwell en électromagnétisme, représentations du groupe de Lorentz et variétés riemanniennes en théorie de la relativité, équation de Schrödinger en mécanique quantique, etc.
Mais cela ne fait que renforcer le mystère ! Ainsi donc, voilà un petit jeu d'écriture formel auquel jouent, non seulement les mathématiciens depuis deux mille ans, mais aussi maintenant le cosmos, voyez-vous ça. Et lui c'est depuis dix-huit milliards d'années. Comment peut-on croire un instant à de pareilles balivernes ?
A mon grand étonnement, j'ai rencontré dans le milieu scientifique beaucoup de gens qui croient cela, des physiciens, des mathématiciens et des logiciens.
Les physiciens et les mathématiciens purs qui s'intéressent à la physique, parce qu'ils ne connaissent pas de logique et ne savent pas que les mathématiques sont formalisables. Pis que cela, ceux qui se sont un peu penchés sur la question croient même que Gödel a démontré, dans son théorème d'incomplétude, que les mathématiques ne sont pas formalisables !
Les logiciens, en général, ne s'intéressent ni à la physique, ni aux applications des mathématiques et donc ce problème ne les touche pas. Ils veulent donc bien croire tout ce que l'on voudra à ce sujet. Quant aux mathématiciens dits appliqués, ils sont bien loin de tout problème philosophique. Les mathématiques financières ne portent pas vers les spéculations de ce genre !
Mais nous, nous voulons absolument y voir plus clair. Il faut donc chercher un peu plus loin si l'on ne veut pas mourir idiot. Est-ce que la logique a quelque chose à nous dire sur les démonstrations mathématiques en dehors de la formalisation ?
Et en effet, une idée nouvelle est apparue dans les années 60, qu'on appelle maintenant la correspondance preuves-programmes ou correspondance de Curry-Howard.
Haskell Curry est l'inventeur de la *logique combinatoire* : vous formez des termes avec deux lettres K, S et une opération binaire appelée *application* et notée « . » comme l'opération de produit dans les groupes. Mais il n'y a ni associativité ni commutativité. Par contre, vous avez une opération de *réduction* définie par : $Kxy \succ x$; $Sxyz \succ (xz).(yz)$. Par exemple, si on pose $I = (SK)K$, alors $Ix \succ x$ pour tout x.
Les termes de la logique combinatoire, qu'on appelle des *combinateurs*, peuvent donc être considérés comme des programmes. Un sous-terme de la forme Kxy ou $Sxyz$ est appelé *redex*. L'exécution n'est pas déterministe, si on se permet d'appliquer les règles de réduction à n'importe quel redex. Elle le devient si on décide de toujours réduire le redex le plus à gauche.

Or, Haskell Curry a observé qu'on pouvait naturellement associer un combinateur à chaque preuve de la logique propositionnelle *intuitionniste* dite *minimale* : dans ce système logique, vous n'avez que le connecteur $\rightarrow$ et pas de raisonnement par l'absurde.
La méthode consiste à associer K à l'axiome $A \rightarrow (B \rightarrow A)$, S à l'axiome $(A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))$, et l'application au modus ponens. On écrit :
$\vdash K : A \rightarrow (B \rightarrow A)$, $\vdash S : (A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))$ et
$\vdash t : A \rightarrow B, \vdash u : A \Rightarrow \vdash tu : B$.

Par exemple, on obtient de cette façon $\vdash I : A \rightarrow A$. On a, en effet :
$\vdash S : (A \rightarrow (B \rightarrow A)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow A))$, et donc
$\vdash SK : (A \rightarrow B) \rightarrow (A \rightarrow A)$. Mais si on pose $B = B' \rightarrow A$, on a $\vdash K : A \rightarrow B$ et donc $\vdash (SK)K : A \rightarrow A$.

William Howard a considéré aussi la logique intuitionniste minimale, mais a utilisé un autre système de preuve qui est la *déduction naturelle*. Il a également obtenu des programmes, qui sont alors écrits en λ-*calcul*.

La logique intuitionniste minimale est un système très pauvre et très peu expressif. Serait-il possible d'étendre cette correspondance à un vrai système où l'on peut faire des mathématiques, par exemple l'Analyse ? Et tout d'abord, passer au calcul propositionnel classique ?
Avec les préjugés de l'époque, la réponse a été un « non » catégorique. En effet, à ce moment-là, tout le monde « savait » que la logique intuitionniste était constructive, que la logique classique ne l'était pas et qu'un programme était un calcul. Donc la correspondance preuves-programmes ne pouvait exister que pour la logique intuitionniste. Le raisonnement était peut-être juste, seulement les trois prémisses étaient fausses. Et la conclusion aussi.
Pourtant, au début, tout s'est passé comme prévu. On a étendu très facilement la correspondance de Curry-Howard à la logique *intuitionniste* du second ordre. Il suffit d'expliquer la construction des termes qui correspondent aux règles et axiomes pour le quantificateur universel. Et c'est particulièrement simple.
Pour la généralisation, on a la règle suivante : si on a obtenu $\vdash t : F[c]$, alors on déduit $\vdash t : \forall x\, F[x]$.
Pour la particularisation, on a, au premier ordre, $\vdash I : \forall x\, F[x] \rightarrow F[t]$
et au second ordre $\vdash I : \forall X\, F[X] \rightarrow F[\Phi]$.
Il est d'ailleurs étonnant que le schéma de compréhension, qui contient souvent l'essentiel des idées d'une démonstration mathématique, corresponde a un programme qui ne fait rien. Mais la correspondance de Curry-Howard nous réserve encore bien des sujets d'étonnement.

A ce stade, tout le monde était content et on pensait que la correspondance preuves-programmes avait atteint sa plus grande extension possible avec la logique intuitionniste du second ordre. On ne pouvait transformer en programmes que les preuves dites constructives, ce qui semble tout à fait sensé. Malheureusement cela exclut pratiquement toutes les preuves mathématiques, car le tiers exclu est omniprésent dans le raisonnement mathématique.
C'est donc avec beaucoup de scepticisme et un peu de condescendance que les logiciens ont accueilli la découverte de Tim Griffin en 1990 : une obscure instruction d'un dialecte de LISP appelé SCHEME avait comme type la loi de Peirce $(\neg A \rightarrow A) \rightarrow A$, qui n'est autre que le raisonnement par l'absurde. Cette instruction, au nom bizarre de « call-with-current-continuation », servait essentiellement dans des tâches informatiques subalternes comme la programmation système ou la gestion des erreurs. Cet informaticien ignorant ne savait même pas qu'il ne fallait

pas utiliser la logique classique pour typer des programmes. En plus, il ne s'agit pas de vrais programmes car, en programmation système, on ne fait pas de calcul et les programmes sérieux sont des calculs.
Cette vision caricaturale de l'informatique a cours dans tout le milieu scientifique, logiciens, mathématiciens, physiciens, etc. L'informatique serait là pour servir d'outil. La programmation système est une affaire de techniciens besogneux et le scientifique sérieux s'en désintéresse complètement, pour se consacrer à ses calculs numériques ou formels. Pourtant, le monde réel nous montre à l'évidence que le système est, de très loin, le programme le plus important : l'empire de Microsoft est basé sur un « système d'exploitation » (la locution française pour « operating system » décrit particulièrement bien la situation).
L'importance de la découverte de Tim Griffin sur le tiers exclu est de nous obliger à voir les choses tout autrement : ce sont des programmeurs qui ont inventé l'instruction associée au raisonnement par l'absurde. Et ils l'ont fait pour des besoins pratiques de programmation système. Il est évident, vu la nature de cette instruction, que jamais un logicien n'aurait pu y penser. Cela veut dire que la relation entre les preuves et les programmes est beaucoup plus profonde qu'on le pensait au début : les « vraies » preuves (où l'on autorise le tiers exclu) correspondent aux « vrais » programmes (qui ne font pas de calcul, mais gèrent le système). On a évidemment fait là un progrès capital dans la compréhension de ce qu'est une démonstration.

Mais maintenant, on voit que, pour transformer en programmes toutes les preuves de l'Analyse, il ne manque plus que de trouver une instruction qui corresponde à l'axiome du choix dépendant. Et là encore, on trouve une instruction essentielle de la programmation système, à savoir *l'horloge*. C'est l'instruction qui compte le nombre de pas de programme exécutés.

Expliquons rapidement le fonctionnement de ces nouvelles instructions. Pour les mettre en œuvre, il faut étendre la logique combinatoire de deux façons. D'abord, on ajoute deux nouvelles constantes que je note cc (call-with-current-continuation) et $\hbar$ (horloge). Mais surtout, à côté du programme qu'on exécute, on considère un *environnement*, qui est une pile de termes (c'est-à-dire une suite finie). Ce qu'on exécute, c'est donc un couple, que je note $t \star \pi = t \star t_1 \cdot \ldots \cdot t_n$. On donne alors les règles d'exécution suivantes, qui fixent le fonctionnement de la machine. Les possibilités pour le terme t sont $t = uv$, $t = K$, S ou cc :

$uv \star \pi \succ u \star v \cdot \pi$ (push)
$K \star u \cdot v \cdot \pi \succ u \star \pi$
$S \star u \cdot v \cdot w \cdot \pi \succ u \star w \cdot vw \cdot \pi$
$\mathsf{cc} \star u \cdot \pi \succ u \star \mathsf{k}_\pi \cdot \pi$ (sauvegarder la pile)

Ici, k_π est une constante dans laquelle la pile π a été mémorisée. Pratiquement, c'est un pointeur vers une zone mémoire où l'on met une copie de sauvegarde de la pile π. En programmation, on appelle cet objet une *continuation*. Mais nous avons introduit ces nouvelles constantes k_π dans notre langage de programmation et il faut expliquer leur fonctionnement :

$\mathsf{k}_\pi \star t \cdot \pi' \succ t \star \pi$ (restaurer la pile)

La nouvelle instruction k_π efface donc la pile courante et met à la place celle qu'elle contient en mémoire.
On peut montrer que l'instruction cc a comme type la loi de Peirce, c'est-à-dire le raisonnement par l'absurde. Vous comprenez sans doute maintenant pourquoi seul un programmeur pouvait trouver cela.
Quant à l'instruction d'horloge $\hbar$, elle s'exécute comme suit :

$\hbar \star t \cdot \pi \succ t \star n \cdot \pi$
où n est un entier qui représente l'heure courante (le nombre d'instructions effectuées depuis le boot). Il est assez surprenant, dans ce cas aussi, que le type de cette instruction soit l'axiome du choix dépendant ; plus précisément, un axiome dont l'axiome du choix dépendant est conséquence logique.

Revenons à notre problème, qui est de comprendre la nature des démonstrations mathématiques. Nous avons vu au début qu'une preuve est un texte satisfaisant à des règles syntaxiques absolument strictes. Nous venons d'expliquer que les constituants de ce texte, à savoir les axiomes, correspondent à des instructions de programmation. La conclusion qui s'impose alors est que les démonstrations mathématiques ne sont pas autre chose que des programmes.
Cela nous donne une vision très claire et que je trouve très amusante des débats à propos de l'utilisation de tel ou tel axiome. Quand Brouwer met les mathématiciens en garde sur l'utilisation du tiers exclu, ou quand Baire et Borel font de même avec l'axiome du choix, on pense immédiatement aux mises en garde que l'on trouve dans les manuels de programmation à propos d'instructions comme `goto` ou de ce qu'on appelle les effets de bord. Finalement, le message de Brouwer est le suivant : surtout, ne programmez pas à l'aide de continuations, car c'est aller au-devant des bugs. Et le conseil de Borel est d'éviter d'utiliser l'instruction d'horloge, pour la même raison.

Ne nous arrêtons pas en si bon chemin : il reste deux questions que j'ai laissées en suspens au début de mon exposé.

1. Pourquoi y a-t-il eu tant de gens depuis deux mille ans pour écrire des mathématiques ?
Si les preuves sont des programmes, il n'y a qu'une façon de répondre (mais sinon, il n'y en a aucune !). Les mathématiciens écrivent donc des programmes ; comme il n'y a pas de raison d'écrire des programmes si on n'a pas d'ordinateur, c'est qu'ils en ont toujours eu un sous la main. Bien entendu, il ne peut s'agir que du cerveau humain.
Les mathématiciens écrivent des programmes, mais ils ne programment pas pour autant. En effet, chaque fois que l'on transforme une preuve en programme, on obtient du code que l'on ne comprend pas immédiatement (et même pas du tout, la plupart du temps). C'est exactement comme quand on lit des programmes directement dans la mémoire d'une machine. Cela a du sens, mais il est extrêmement difficile à comprendre.
Cela signifie que les mathématiciens lisent et recopient des programmes en code qui sont écrits dans leur propre cerveau. On comprend que cela soit une occupation fascinante. Je peux en témoigner, puisque c'est ma principale occupation depuis fort longtemps. Mais cela m'intéresse encore plus depuis que j'ai compris ce dont il s'agit.

2. Pourquoi le monde physique joue-t-il à se conformer à des lois mathématiques ?
Bien entendu, il n'en est rien, le monde physique n'a aucune raison de se conformer à des programmes écrits dans notre cerveau. C'est l'inverse qui est vrai : les programmes que l'évolution a écrits dans notre cerveau sont une excellente représentation du monde physique. Cela veut dire que Newton, Maxwell et Einstein ont trouvé les lois de la gravitation, de l'électromagnétisme et de la relativité en lisant dans leur propre cerveau et non en observant le monde extérieur.

Pour terminer, je voudrais dire qu'il y a là un champ de recherches passionnant, dans de multiples directions. Avec Yves Legrandgérard, nous avons implémenté la machine que je vous ai vaguement décrite tout à l'heure (sous licence libre !).

Elle fait tourner les programmes issus de la formalisation de théorèmes et les résultats sont fascinants. Les applications informatiques sont clairement à portée de main.
Mais vous aurez compris aussi qu'il y a là matière à sérieuses réflexions pour l'épistémologie et les sciences cognitives.

****