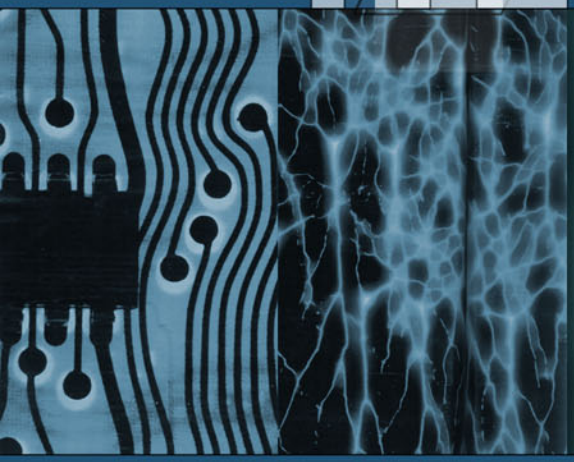
David Perrin
GMC04
P2002

utbm
université de technologie
Belfort-Montbéliard

HE02

le λ-calcul

le reTour de Babel ?



INTRODUCTION

« Tout le monde se servait d'une même langue et des mêmes mots. [...] ils trouvèrent une vallée au pays de Shinéar et s'y établirent ».
[Genèse XI/2]

« Confondons leur langage pour qu'ils ne s'entendent plus les uns et les autres. Dieu les dispersa de là sur toute la face de la terre et il cessèrent de bâtir la ville. Aussi la nomme t-on Babel, car c'est là que Dieu confondit le langage de tous les habitants de la terre ».
[Genèse XI/8]

___

C'est en ces mots que les écrits anciens évoquent l'échec d'un langage commun à tous…2000 ans plus tard, en élaborant pour la première fois une architecture globale du cerveau, une théorie mathématique est en train de révolutionner ce que l'on sait de l'intelligence mais aussi de l'informatique, de la linguistique ainsi que des rêves…

Comme déclencheur ; le célèbre théorème d'incomplétude de Gödel. Elaboré il y a un siècle , il pourrait ne pas parler uniquement de mathématiques. Traduit dans un *langage*, le lambda-calcul (ou *λ-calcul*), ce théorème transposé à notre époque correspondrait à un programme informatique. Au niveau neuronal ce pourrait être durant le sommeil que l'application répare les fichiers de notre cerveau… autant de points « flous », qui seront éclaircis dans ces quelques pages.

Ce rapport fait dans le cadre de l'unité de valeur HE02 - *histoire des mathématiques* - est l'occasion pour moi de dresser un bilan succinct des connaissances relatives au λ-calcul sans rentrer dans les démonstrations scientifiques associées à cette théorie – *c'est là, le rôle d'une culture générale*… -. Au-delà d'un catalogue scientifique, il sera surtout fait état des possibilités « pluridisciplinaires » que l'on est en mesure d'attendre de cette ramification particulière des mathématiques.

On gardera à l'esprit que ce rapport doit à de nombreux auteurs, mais surtout au mathématicien français Jean-Louis Krivine pour son ouvrage *Lambda-calcul, types et modèles*[1] ainsi qu'au journaliste Hervé Poirier pour son dossier exhaustif *Toute pensée est un calcul*[3] paru dans le magazine Science & Vie.

David Perrin
Département Génie
Mécanique et Conception
4ème année.

(dossier achevé en mail 2002).

SOMMAIRE

1. INTRODUCTION AU λ-CALCUL

1.1. Historique

IX^e s	al-Khwarizmi publie les premiers algorithmes.
XII^e s	Le philosophe Wilhelm Leibniz cherche « l'alphabet des pensées humaines ».
1900	David Hilbert propose d'axiomatiser et de formaliser les mathématiques.
190l	Edmund Husserl veut fonder les vérités logiques sur des processus psychiques avec la phénoménologie.
1932	Alonzo Church invente le lambda-calcul.
1944	John von Neumann conçoit le premier ordinateur.
1948	Haskell Curry voit dans le lambda-calcul la structure logique sous-jacente aux langues naturelles

1.2. Préceptes et notions de base

Le lambda-calcul est un langage inventé en 1932 par le mathématicien américain Alonzo Church pour résoudre des questions de logique pure. Les mots de ce dialecte sont des formules de calcul.

Trois opérations grammaticales sont autorisées :

1. l'opération **d'application**, consistant à appliquer une formule f à une formule x, noté f(x).
2. l'opération **d'abstraction**, qui consiste à baptiser une formule construite, noté λxf(x).
3. l'opération de **bêta-réduction**, qui consiste à exécuter le calcul proprement dit.

Dans les années 50, les informaticiens se sont rendus compte que cette syntaxe minimaliste se révèle parfaite pour programmer un ordinateur. Lorsqu'un programme est enregistré sur une machine, il faut, en effet, préciser à quelle adresse ces lignes de code ont été gravées dans la mémoire de l'ordinateur, et à quelle adresse sont stockées les données que le programme doit analyser. C'est exactement ce que permet le lambda-calcul, si l'on remplace la notion de formule par celle de programme.

Dès lors, pour un programme P :

1. l'opération d'application P(x) précise l'adresse x où se trouvent les données
2. l'opération d'abstraction λxP(x) définit l'adresse où le programme est stocké
3. la bêta-réduction fait tourner le programme

Plus surprenant, le λ-calcul permet de définir les mathématiques.

exemple : le nombre 2 est le lambda-terme λPλPxP(P(x)).

Explication : selon Church, l'entier 2 doit être l'instruction *« répétez deux fois le programme qui suit »*. Or, d'après la lambda-grammaire, l'expression λxP(P(x)) correspond à l'adresse de l'instruction répétant deux fois le programme P. En rajoutant λP devant, on tombe donc sur l'adresse du programme qui demande de répéter deux fois le programme qu'on lui présente.

En suivant le même raisonnement, on peut alors définir :

- l'addition : λmλnλfλxm(f)(n(f)(x))
- la multiplication : λmλnλfm(n(f))
- *toute l'arithmétique...*

En achevant un travail commencé cinquante ans auparavant, Jean-Louis Krivine[1], est parvenu à démontrer, en 1997, que le lambda-calcul permet d'exprimer tous les raisonnements ainsi que toutes les structures mathématiques possibles.

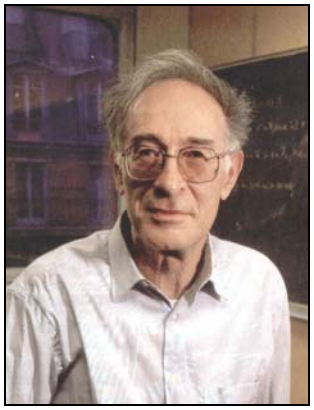


fig 1. Jean-Louis Krivine, mathématicien français

Non content d'être un langage informatique, ce monde fermé de formules qui s'appliquent les unes sur les autres dans un enchevêtrement complexe d'instructions serait aussi un langage mathématique universel.

Le lambda-calcul pourrait-il être le langage de la pensée ?

[1] Mathématicien français de renommée mondiale. Cousin d'Alain, porte-parole de la Ligue Communiste Révolutionnaire et d'Emmanuel, directeur de l'Orchestre Français des jeunes.

2. LA TRADUCTION MATHEMATICO-INFORMATIQUE

2.1. La correspondance Curry-Howard

Au début des années 80, Jean-Louis Krivine se consacre à l'étude de la correspondance de Curry-Howard[9]. Correspondance issue d'une remarque faite en 1958 par le logicien américain Haskell Curry, explicitée onze ans plus tard par William Howard, elle affirme que certaines démonstrations mathématiques correspondent à des programmes informatiques.

Une démonstration est achevée si, des hypothèses à la conclusion, on peut l'expliquer à une personne toute juste capable d'appliquer les quelques règles élémentaires de raisonnement qu'on lui a inculquées... rôle que peut tenir l'ordinateur. Cette correspondance ébauche un dictionnaire mathématico-informatique et montre que, malgré leurs différences de langages apparentes, ces deux constructions « humaines » peuvent s'exprimer en λ-calcul.

Haskell Curry a remarqué que ce dialecte permet aussi de traduire des règles de déduction utilisées en mathématiques, comme l'implication. Or, une démonstration mathématique n'est qu'une succession de règles de déduction qui s'appliquent sur des objets. Les démonstrations peuvent donc êtres traduites, ligne après ligne, dans le langage du lambda-calcul pour devenir de véritables programmes informatiques, c'est un travail mécanique d'assemblage.

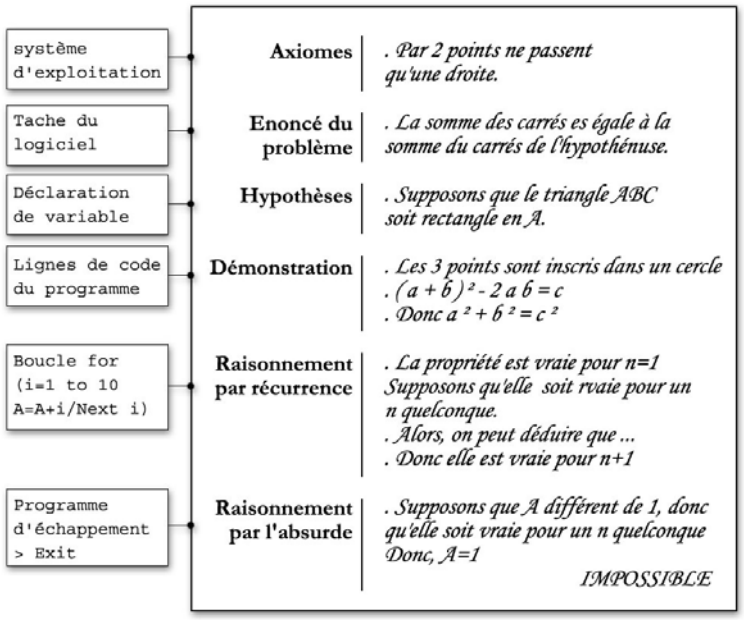


fig 2. Analogies mathématico-informatiques ; démontrer c'est programmer

La correspondance entre théorème et programme devient très élégante, elle est résumée dans le schéma suivant :

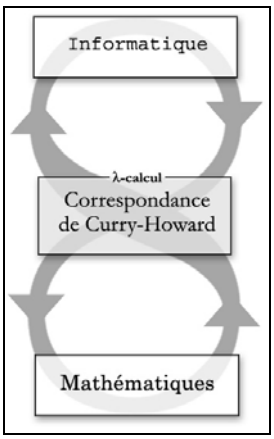


fig 3. principe de la correspondance de Curry-Howard

2.2. Le raisonnement par l'absurde

Depuis les premières pages de Curry, de grands logiciens, comme le Français Jean-Yves Girard[10], l'ont peu à peu complété. Mais ce dictionnaire était trop incomplet pour susciter l'intérêt des mathématiciens. Pour preuve, un raisonnement mathématique échappait à toute traduction : le raisonnement par l'absurde - qui suppose le contraire de ce que l'on veut prouver afin d'arriver à une contradiction -. Par conséquent et par faute d'un répertoire plus complet, les mathématiciens ont cru longtemps que seules les démonstrations sans ce raisonnement pouvaient être traduites dans le lambda-calcul.

Or, en 1990, les informaticiens américains Matthias Felleisen et Timothy Griffin ont découvert que le raisonnement par l'absurde correspondait, en fait, à un petit programme présent dans tous les ordinateurs : les instructions d'échappement[1].

2.3. La première traduction mathématico-informatique

Tous les raisonnements sont désormais traduits dans le lambda-calcul[2], le dictionnaire est désormais devenu universel ; tout théorème peut être lu en termes informatiques.

Ainsi, Jean-Louis Krivine s'est lancé dans la traduction quelques théorèmes, il choisit un texte particulier : le théorème de complétude de Gödel. A l'aide de son dictionnaire, il a traduit sa démonstration et chercha le sens du programme. Les démonstrations qui exploitent le raisonnement par l'absurde donnent un lambda-terme très compliqué, devant lequel il faut faire preuve d'agilité logique et de connaissance informatique voire même de chance pour voir à quoi ce programme peut bien servir. Il réussit finalement à entrevoir le sens caché du théorème de complétude de Gödel.

[1] Les instructions d'échappement mémorisent un état de référence de la machine pour y revenir en cas de problème. exemple : message d'erreur lorsqu'on demande à un ordinateur d'accéder à un lecteur sans disquette.
[2] Il faut ajouter (cc) dans le lexique du lambda-calcul, qui correspond à l'instruction d'échappement, équivalence informatique du raisonnement par l'absurde (de J.-L. Krivine).

Ce théorème semble correspondre à un logiciel bien connu des informaticiens ; le désassembleur[1], il fait tourner instruction par instruction d'autres programmes afin d'en comprendre le code, de le modifier ou de trouver les erreurs de programmation.

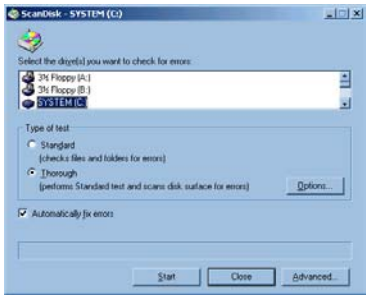


fig 4. réparateur de fichiers informatiques

Cependant le programme crée stipulait de faire du « pas à pas » partout, sauf lorsqu'il rencontre les instructions d'échappement qu'il doit systématiquement sauter. Mais les spécialistes de la programmation savent qu'un désassembleur doit inclure cette particularité répondra Emmanuel Saint-James[2][4] ; on ne peut pas demander au désassembleur de faire du pas à pas sur les instructions d'échappement, sinon, il plante l'ordinateur. La découverte est incroyable…

2.4. La véritable racine des mathématiques

En traduisant le théorème d'incomplétude démontré en 1931 par le logicien autrichien Kurt Gödel, Jean-Louis Krivine a donc réécrit précisément un programme informatique très élaboré qui intègre de façon parfaite la sécurité d'un système informatique, alors que l'ordinateur n'existait pas.

La situation est troublante : Kurt Gödel aurait élaboré un programme informatique indispensable au bon fonctionnement d'un ordinateur de bureau, mais dont la tache n'a, a *priori,* rien à voir avec son idée mathématique de départ - et tout cela, avant que l'ordinateur ne soit inventé ! il se serait inconsciemment référé à de la seule marque d'ordinateur commercialisée sur le marché depuis plusieurs milliers d'années : le cerveau humain.

De plus, selon la théorie de Krivine, ce programme doit aussi être présent dans notre cerveau, le rendant inutilisable pendant qu'il tourne. Quel pourrait donc être ce processus cognitif très contraignant ? *« Cela pourrait être une des fonctions du sommeil »,* propose le logicien, *« Car pendant que le sommeil, des choses défilent dans notre cerveau en veille ».* Le logicien autrichien croyait étudier les limites de la démarche mathématique, il pensait contredire le programme de Hilbert qui prétend atteindre toute vérité mathématique à partir d'axiomes et règles de départ. Il ne faisait qu'étudier le fonctionnement de son cerveau. II reconstruisait un programme informatique, peut-être indispensable à la psychanalyse !

Dans ces conditions, nos rêves seraient un moyen de restaurer les *fichiers* mal fermés de notre cerveau. La nuit, pendant notre sommeil, on rangerait les *programmes* de la journée, on quitterait les *applications* mal fermées.

C'est un autre trouble qu'induit la théorie de Jean-Louis Krivine : les psychanalystes essayent d'atteindre les couches du sujet par le haut, en partant du langage parlé, il devient possible de les aborder par le bas, en partant du lambda-calcul. De quoi reconstituer, peut-être, les couches de l'inconscient qui structurent la psychanalyse. C'est cette notion de couches qui rattache le λ-calcul à l'activité cérébrale, ce que l'on allons détailler dans la partie suivante

[1] Publiée par la revue Bulletin of Symbolic Logic, la correspondance entre le théorème de complétude et le désassembleur n'est pas encore rentrée dans les mœurs.
[2] Informaticien spécialiste du langage LISP

3. LA STRUCTURE DU CERVEAU

3.1. Une structure par couches

Krivine à eu l'intuition de voir dans le lambda-calcul la structure logique qui régit l'intérieur de notre crâne, le langage universel du réseau de neurones du cerveau humain, un petit peu comme le comportement intelligent d'une fourmilière : fruit des multiples actes individuels et obtus des fourmis.

Pour comprendre, il faut imaginer qu'au plus bas niveau de notre cerveau s'enchevêtre les neurones, desquels émergerait un langage structuré ; le lambda-calcul. Lequel serait lui-même représenté, à un niveau supérieur, par des couches de langages de plus en plus évolués, dont la plus haute correspond aux langues naturelles que nous parlons.

Il se superposerait donc dans notre cerveau des couches de langages de plus en plus sophistiquées, l'intelligence ne résiderait pas au niveau de l'activité neuronale. On comprend aisément ceci au regard de l'ordinateur, car il paraît difficile d'en comprendre son fonctionnement uniquement par observant du mouvement des électrons dans les circuits électroniques *(cf. ci-dessous)*. Nos pensées, conscientes et inconscientes, s'élaboreraient au niveau des couches de langages intermédiaires, précisément au-dessus du lambda-calcul.

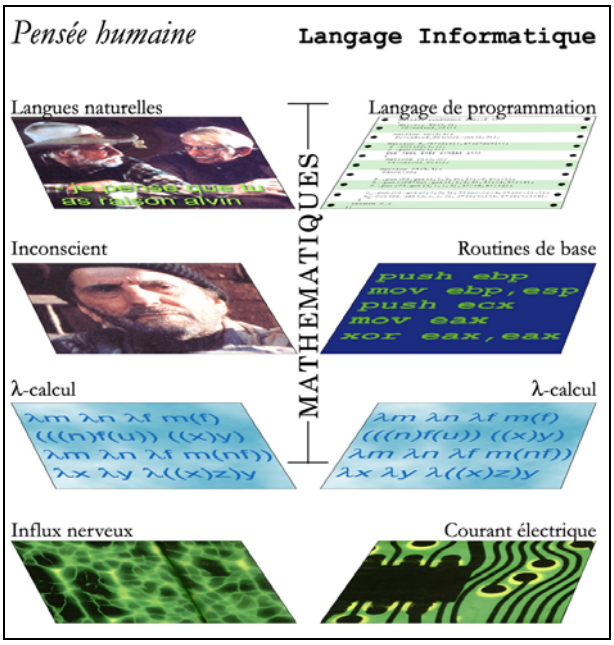


fig 5. analogies des couches cérébrales et informatiques

C'est un changement radical de perspective, car en intercalant son point de vue entre la couche inférieure des neurones et les couches supérieures des langages évolués, la théorie de Jean-Louis Krivine va directement à l'essentiel. Elle permet, de dévoiler l'origine de toute activité cognitive, de saisir le secret même de l'intelligence. Derrière nos pensées mouline le lambda-calcul. D'ores et déjà, les conséquences sont incalculables.

Ainsi, les mathématiciens sont persuadés, depuis trois mille ans qu'ils élaborent des théorèmes, perçant les vérités d'un monde idéal. Selon la théorie de Krivine - lui-même logicien -, tout mathématicien est en fait un programmeur qui s'ignore. Car, passées dans le λ-calcul, qui, rappelons-le, est un langage informatique, les démonstrations de leurs théorèmes peuvent dès lors être traduites, révélant des lignes de code informatique insoupçonnées.

Autrement dit, les mathématiciens ont le privilège de pouvoir changer de niveau, de monter jusqu'à des sommets d'abstraction en créant des langages et des structures très complexes, et de descendre vers les niveaux inférieurs en développant leurs calculs; inconscients de leur propre rôle, ils essayent en fait de comprendre comment un système aussi simple que le lambda-calcul crée sous leur propre crâne des langages de programmation et des programmes informatiques aussi élaborés.

Sans le savoir, lorsqu'ils démontrent des théorèmes, ils ne font que réécrire les programmes de leur propre cerveau ! *« Mais ils ne font pas le boulot jusqu'au bout »,* ajoute Jean-Louis Krivine. *« Il faudrait le terminer en redonnant aux théorèmes leur vrai sens, en montrant à quels programmes ils correspondent, en montrant quelle tâche ils exécutent dans le cerveau ».*

3.2. Le mimétisme de l'ordinateur

L'idée de comparer notre cerveau et l'ordinateur n'est pas nouvelle. Elle date des débuts de l'informatique, lorsque Alan Turing et John von Neumann conçurent le premier ordinateur (la machine *Mach I*) comme une copie du cerveau humain. Mais il y a prescription, car aujourd'hui, les informaticiens sont totalement dépassés par la complexité des programmes à construire. Avec la théorie de Krivine, cette comparaison retrouve pourtant ce qu'elle portait à l'origine d'intuition féconde. Car en fait ce n'est pas notre cerveau qui ressemble à l'ordinateur, explique le logicien, mais l'ordinateur qui essaie à toute force, par programmeur interposé, de ressembler à notre cerveau. Le changement de point de vue est radical.

Concrètement cette ressemblance ne se situerait pas au niveau du *hardware* (matériel), de la machine proprement dite, faite de fils électriques et de puces, mais au niveau du *software* (logiciel) du système de logiciels implémentés dans la mémoire. Si le lambda-calcul intervient entre la couche des circuits électroniques et les langages évolués de programmation dont se servent les informaticiens pour écrire leurs logiciels, c'est peut-être parce que notre cerveau procède de cette manière. A partir de là. toute activité calculatoire ou cognitive peut être vue comme l'expression de « programmes » implémentés dès le départ dans le cerveau.
Selon la même théorie ce serait grâce à l'évolution darwinienne que de tels programmes auraient été mis au point dans notre cerveau, par la nature en quelque sorte…

3.3. Vers une révolution pluridisciplinaire

Walter Fontana[1] aux Etats-Unis, s'est inspiré de la sélection darwinienne pour construire un « programme du prédécesseur », soit une application qui répond 12 quand on lui donne le nombre 13, ou 6 pour 7.

Or, en langage lambda-calcul, ce programme d'apparence très simple, s'est révélé difficile à mettre au point. Parti d'une séquence quelconque (un lambda-terme choisi au hasard). Walter Fontana l'a fait muter aléatoirement, en sélectionnant les programmes les plus proches de la tâche désirée, avant de lancer de nouvelles mutations. Résultat : au bout de plusieurs centaines de milliers de générations, il a obtenu quatre programmes du prédécesseur efficaces. Trois étaient déjà connus des informaticiens, mais pas le quatrième. Beaucoup plus court, simple et ingénieux que les autres, il n'avait jamais été trouvé en cinquante ans d'informatique...

Tout se passe donc comme s'il n'y avait eu qu'une seule manière de penser pendant les millions d'années de sélection naturelle, les dizaines de milliers d'années de babils humains, les trois mille ans de recherche mathématique et les cinquante ans de programmation informatique : le lambda-calcul, la source unique de l'intelligence, cet « alphabet des pensées humaines » qu'entrevoyait le philosophe et mathématicien allemand Gottfried Wilhelm Leibniz au XVIIe siècle, faute de pouvoir aller plus loin. Du coup, l'adéquation des mathématiques avec la réalité, recherchée au temps de l'école pythagoricienne et qui déconcerte les épistémologues, ne serait plus si « déraisonnable ».

[1] Chercheur autrichien de l'institut Santa Fe

Cette théorie reste cependant à valider. Il faut que les structures matérielles de cette hypothétique architecture informatique soient effectivement observées dans notre cerveau. *« On ne tombera peut-être pas exactement sur le lambda-calcul, prédit Jean Petitot[1] mais sur un « gamma-calcul » tout à fait analogue. Mais, une fois ce gamma-calcul connu, toutes les thèses de Krivine, selon moi. sont justes ».*

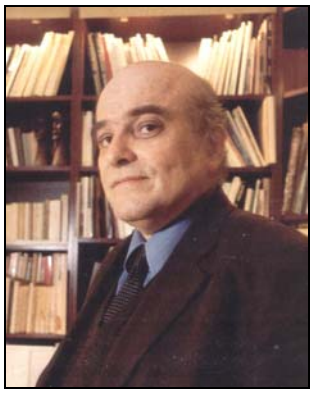


fig 6. Jean Petitot, directeur du Centre de recherche en épistémologie appliquée de l'Ecole polytechnique

Ce serait alors l'une des plus importantes découvertes scientifiques de tous les temps. En revenant aux sources de la pensée humaine, le logicien français promet aux chercheurs en intelligence artificielle, informaticiens, mathématiciens, linguistes, psychologues, psychanalystes, neurologues et cognitvistes de se rendre compte qu'ils étudient tous le même objet sous un point de vue de différent. Chacun pourrait alors profiter du savoir accumulé dans les autres disciplines.

Pour plonger dans ses arcanes, il suffit de décrypter ses programmes, en exploitant les connaissances accumulées par les mathématiciens et les informaticiens. Le fonctionnement parfois névrotique d'un PC permettra peut-être de comprendre certaines pathologies psychologiques. Ce que l'on sait du fonctionnement du cerveau devrait faire un bond en avant, il y eu l'humiliation cosmologique de Galilée qui ôté l'homme du centre de l'Univers, l'humiliation biologique de Darwin, qui l'a rejeté au même rang que les animaux, et l'humiliation psychologique de Freud, qui lui dénie la maîtrise consciente de ses actes, Krivine infligerait peut-être une quatrième humiliation, neurologique : le cerveau humain serait semblable aux simples circuits électriques d'un ordinateur de bureau.

[1] Directeur du Centre de recherche en épistémologie appliquée de l'Ecole polytechnique

4. LES LANGUES HUMAINES

4.1. L'essence des langues humaines

En linguistique, le lambda-calcul est déjà exploité pour structurer les langues naturelles parlées sur Terre. L'idée n'est pas nouvelle, souligne Jean-Pierre Desclés[1]. *« Dès les années cinquante, des logiciens et linguistes ont vu dans le lambda-calcul la structure logique sous-jacente aux langues naturelles. En étudiant la sémantique des phrases, comme les relations entre un nom propre, un verbe et un prédicat, tous les énoncés, tous les mots, même, peuvent être vus comme des lambda-termes, c'est-à-dire comme des programmes informatiques. »* Pascal Boldini[2], est persuadé que c'est la bonne voie. Pour lui cela permettrait de formaliser les langues naturelles de façon beaucoup plus concise que n'importe quelle autre théorie linguistique. »

De son côté, Jean Petitot, est arrivé, lui aussi, à une conclusion tout à fait identique en étudiant les relations entre le langage et la perception :

de la même façon que le langage mathématique peut être vu comme le commentaire (ou le « typage ») de programmes élaborés à partir de la couche du lambda-calcul, les langues naturelles bricolées par les êtres humains peuvent être vues comme l'interprétation, dans un langage de très haut niveau, des calculs neuronaux sous-jacents. L'existence de cette couche de lambda-calcul commune à tous les êtres humains expliquerait alors pourquoi deux langues en apparence aussi différentes que l'anglais et le mohawk (langage parlé par une tribu de la confédération iroquoise) présentent de profondes analogies. Jean Petitot en est convaincu que la théorie de Krivine est la première thèse sérieuse sur l'origine du langage chez les hominidés.

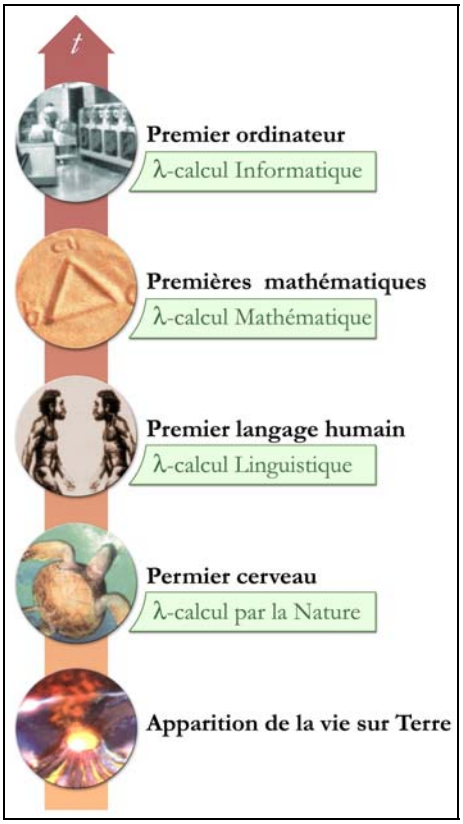


fig 7. évolution de l'humanité et des λ-calcul associés

[1] Directeur du laboratoire Langage, logique, informatique et cognition (LaLIC), à l'université Paris-Sorbonne.
[2] Membre du laboratoire LaLIC.

5. LE λ-CALCUL APPLIQUE

5.1. L'état actuel du savoir-faire

En effet, plus les programmes sont sophistiqués, moins les informaticiens sont capables de savoir s'ils fonctionnent correctement. Or, ces programmes deviennent indispensables dans les secteurs les plus sensibles de l'économie et de la vie sociale. Une prise de conscience récente, selon Roberto di Cosmo[1]. Il rappelle qu'il a fallu pour arriver à cela d'importantes défaillances informatiques avec d'immenses pertes financières, comme le blocage de l'aéroport de Denver en 1990 - à cause d'une défaillance dans le système de gestion des bagages -, ou encore l'explosion d'Ariane 5 en 1996 - due à un bug dans un petit programme périphérique - pour que ingénieurs, industriels et politiques prennent la question au sérieux.

En effet, jusque-la ; *« un logiciel était en effet considéré comme correct tant qu'il ne s'était pas révélé défaillant »*. La commission d'enquête sur l'accident d'Ariane a proposé d'adopter le principe opposé, selon lequel : *« un logiciel est présumé défaillant tant qu'il n'a pas été jugé correct. »*.

Un changement de point de vue radical et nécessaire car les programmes informatiques actuels sont élaborés selon des méthodes empiriques qui ne sont plus acceptables aujourd'hui. L'ampleur du problème fait frémir, le niveau de complexité de l'informatique nécessite un immense travail collectif de recherche, avoue Paul Horn[2]. Pour exemples ; Intel, a pour sa part investi plusieurs millions de dollars dans cette chasse aux bugs, quant au gouvernement français, il prévoit un recrutement exceptionnel de chercheurs à l'INRIA et au CNRS, au sein du nouveau département STIC[3].

Une des solution permettant d'éradiquer les bugs informatiques et de changer le point de vue de l'Informatique pourrait bien être de rendre les programmes aussi incontestables que des théorèmes, ce grâce au lambda-calcul.

5.2. Nouveaux fondements : prouver et démontrer

L'informatique d'aujourd'hui est confrontée à une crise des fondements, estime Pierre-Louis Curien[4] malgré son développement effréné, il s'agit d'une science très jeune, encore à la recherche de concepts fondamentaux. Prouver qu'un programme ne comporte pas d'erreur se règle traditionnellement par des tests afin de vérifier la réaction du programme dans différentes situations.

Néanmoins ces tests ne peuvent être exhaustifs, de la même façon qu'il est vain d'espérer prouver un résultat mathématique en le vérifiant sur quelques valeurs particulières. La transposition mathématique des programmes informatiques par la correspondance de Curry-Howard permettrait de raisonner avec la logique mathématique.

Les informaticiens sont loin d'avoir exploité tout le potentiel de cette correspondance, le but ne sera plus de prouver qu'un programme ne comporte pas de bug, mais de reconstruire le programme à l'aide d'un langage de programmation beaucoup plus naturel : le langage mathématique, rassuré par trois mille ans d'expérience. La correspondance est alors utilisée dans l'autre sens : on part de raisonnements mathématiques que l'on traduit en programme informatique. Les informaticiens pourraient ainsi être aussi à l'aise avec leurs lignes de code que les mathématiciens le sont avec leurs lignes de démonstrations...

[1] De l'Institut national de recherche en informatique et en automatique (INRIA)
[2] Responsable des laboratoires de recherche d'IBM
[3] Science et Technologie de l'Information et de la Communication
[4] Directeur du laboratoire Preuves, programmes et systèmes du CNRS et de l'université Paris-7

5.3. Etude d'applications spécifiques

5.3.1 Le logiciel COQ

Durant 20 ans, les chercheurs espéraient rendre les programmes aussi sûrs que les théorèmes. Ainsi, pendant les années 80, des informaticiens de l'INRIA ont mis au point le logiciel COQ, basé sur la correspondance de Curry-Howard. L'écriture d'un programme en Coq demande un travail un plus important que traditionnellement, mais une fois écrit, son fonctionnement est garanti. COQ est par exemple utilisé pour garantir les opérations effectuées par les cartes à puce, il formalise les programmes informatiques embarqués sur les cartes à puce[1] afin de prouver la validité des parties les plus complexes et sensibles.

Mais l'intérêt de COQ est plus qu'une garantie anti-bug ; il aide les informaticiens à mieux comprendre la structure des logiciels. Par exemple, les fabricants de cartes à puce avaient un problème avec la taille du programme vérifiant la validité des opérations effectuées sur les cartes. Ce filtre de plusieurs mégaoctets était réputé impossible à intégrer dans les cartes, pourtant, formalisé avec COQ, il à été réduit à 8 Ko.

5.3.2 Le langage orienté objet

La programmation-objet (des langages orientés objet, LOO) est un des concepts clés de l'informatique. En faisant appel à des objets, des sous-programmes définis dans une bibliothèque de données, elle permet de concevoir de gros logiciels sans se perdre dans un océan de lignes de code.

Les fondements de cette méthode de programmation vont peut-être bientôt être bouleversés. Jean-Louis Krivine s'est en effet attaqué au théorème du buveur, *« le plus trivial des théorèmes non triviaux »*, selon lui. Ce théorème, qui n'a aucun intérêt mathématique, s'énonce ainsi :

« dans tous les bars, il y a toujours une personne dont on peut dire
avec raison que, si elle boit, alors tous les autres clients boivent aussi ».

Démonstration :
Prenons un bar au hasard ;

- Soit tous les clients sont en train de boire et la phrase du théorème est vraie quel que soit le client choisi.

- Soit il existe des clients qui ne boivent pas et il suffit de choisir l'un d'eux pour que la phrase « s'il boit, alors tout le monde boit » soit vraie, puisqu'on l'occurrence, la personne concernée ne boit pas...

La traduction de cette démonstration dans le lambda-calcul est courte :

$$\lambda z(cc)\lambda k(z)\lambda d(cc)\lambda h(k)(z)h.$$

Jean-Louis Krivine à travaillé sur ce petit programme durant cinq ans. La transposition construisait plein de nouveaux programmes dont il n'arrivait pas à comprendre le sens, Il ne l'a saisi que très récemment : les programmes fabriqués ne doivent en fait pas être exécutés ; ce ne sont que des noms qui permettent de baptiser les objets de façon précise et impeccable.

Ce petit programme va donc peut-être permettre de restructurer la programmation-objet. Le plus simple des théorèmes pourrait ainsi se révéler d'une importance informatique considérable !

[1] Exigence gouvernementale : « les caractéristiques de sécurité annoncées doivent être vérifiées. »

6. CONCLUSION

« Il y eu l'humiliation cosmologique de Galilée, l'humiliation biologique de Darwin et l'humiliation psychologique la quatrième humiliation pourrait bien être neurologique. »
(Hervé Poirier, Science & Vie)

—

Les premiers résultats sont très encourageants. En traduisant un théorème simple, le logicien français est tombé sur un programme « d'une intelligence incroyable » qui pourrait révolutionner la façon de programmer des gros logiciel. De plus, la théorie de Krivine qui prolonge ces correspondances jusqu'au cerveau ouvre de nouvelles pistes de programmation. En linguistique, Jean-Pierre Desclés, propose de mettre au point de nouveaux langages informatiques pour faire profiter l'informatique de plusieurs millénaires de l'expérience du maniement des langues.

Une partie des attentes du λ-calcul ont étés tracés dans ce rapport et à sa lecture, ces éléments peuvent paraîtrent inconcevables. On notera que ces théories, notamment celle de Jean-Louis Krivine, sont « jeunes » à l'échelle humaine et que les applications futures que l'on est en mesure d'attendre ne sont comparables à rien de ce que l'on connaît aujourd'hui. Ces travaux requièrent un degré d'abstraction qui est désormais nécessaire aux domaines des mathématiques, de la linguistique, de l'informatique, etc… pour rompre leur isolement et se sauver d'elles-mêmes.

Par rapport à l'analogie énoncée en tête de ce rapport, à savoir le bien fondé d'une structure linguistique commune en comparaison à l'épisode de la Tour de Babel, on peut se demander si les enjeux et avancées que l'on attend nous sont nécessaires (dans le sens de nécessité pour le genre humain). Si cela peut paraître évident à nos yeux, il ne faut pas perdre de vue certaines dérives que pourrait permettre ce nouveau type de progrès « abstrait ». A titre d'exemple, selon Jean-Louis Krivine, le λ-calcul correspondant au programme de l'assembleur, s'il était implémenté dans notre cerveau, pourrait donner la possibilité à un individu de connaître l'intention d'un autre individu. Un avantage sélectif, qui, sorti du contexte euphorique de ces nouvelles innovations nous rappelle que les projets de regroupement linguistiques ou autres ont, au final, toujours connus des limites. On pourrait nommer cela *le paradoxe de la Tour de Babel*.

David Perrin
Département Génie
Mécanique et Conception
4ème année.

(dossier achevé en mail 2002).

BIOGRAPHIES

Church Alonzo
logicien américain (Washington 1903 -)
Professeur de mathématiques à l'université de Princeton de 1947 à 1967, puis professeur de philosophie et de mathématiques à l'université de Californie (Los Angeles), il a établi, dans un important théorème (1936), l'indécidabilité du calcul des prédicats du premier ordre. Church a en outre donné son nom à une thèse identifiant la notion intuitive de fonction effectivement calculable avec celle de « fonction récursive générale ». Ainsi, il a contribué à démontrer qu'un certain nombre de substituts formels pour la notion de calculabilité, notamment ce qu'il appelle la « lambda-définissabilité », étaient équivalents à la notion de récursivité générale. Il a écrit : *The Calculi of Lambda-Conversion* (1941), *Introduction to Mathematica Logic* (1944).

Gödel Kurt
logicien américain d'origine autrichienne (Brunn 1906 - Princeton 1978)
Avant de s'établir définitivement à Princeton, Gödel est, de 1933 à 1938, professeur à Vienne. Sa thèse de 1930 avait établi la complétude du calcul des prédicats. Plus connu pour ses deux théorèmes d'incomplétude *Uber formal unentscheidbare Sätze der « Principia Mathematica » und verwandter Systeme* (1931). Le premier théorème établit que, dans tout système formel assez fort, il existe au moins une formule intuitivement vraie qu'on ne peut démontrer. Le second théorème établit qu'on ne peut démontrer la consistance d'une théorie mathématique en n'utilisant que des procédés formalisables à l'intérieur de ce système. Gödel a également établi en 1940 des résultats essentiels de théorie des ensembles. A, en outre, donné une solution à certains problèmes de la théorie einsteinienne et travaillé dans le domaine de la cosmologie relativiste (équations de la gravitation). L'ensemble de son œuvre scientifique est sous-tendu par une inspiration philosophique « platonicienne ».

Neumann Johann ou John von
mathématicien américain (Budapest 1903 – Washington 1957)
II s'est surtout occupé de la théorie des ensembles, de la théorie des jeux et des calculateurs électroniques. Dès la fin des années 30, il a défini de façon théorique la structure possible d'une machine automatique de traitement de l'information à programme enregistré qui correspond à celle de la plupart des ordinateurs. Il a écrit avec Morgenstern *Theory of Games and Econmy Behavior* (1944). Considéré comme l'inventeur du programme informatique

Turing Alan Mathison
mathématicien anglais (Londres 1912 - Winslow, Cheshire 1954)
Dans une de ses plus importantes contributions à la logique mathématique, *On Computable Numbrs, with an Application to the Entscheidungsproblem* (1936-1938), il a élaboré le concept théorique d'une machine à calculer « universelle » (*machine de Turing).* Après une interruption due à la Seconde Guerre mondiale, passée au département des communications du ministère des Affaires étrangères, il reprit ses recherches sur la conception des machines à calculer dans le cadre du Laboratoire national de physique. A partir de 1950, Turing s'intéressa à l'intelligence artificielle. En butte à des poursuites officielles pour ses mœurs homosexuelles - considérées à l'époque comme un crime - on pense qu'il s'est suicidé, même si l'enquête a conclu à un accident (empoisonnement).

REFERENCES

On trouvera dans cette partie les ouvrages qui ont permis l'écriture de ce rapport. Les cinq premiers titres ont principalement servi de base première. Les autres titre pourront être lus à titre de complément, ils sont inscrits là pour indication, ils constituent des ouvrages phare dans l'étude du λ-calcul.

[1] Krivine Jean-Louis.- *Lambda-calcul, types et modèles*.- Etudes et recherche en informatique.- Paris : Masson, 1990. [Loc. : B.U.T.B.M.]

[2] Lassaigne Richard et de Rougement Michel.- *Logique et fondements de l'informatique, logique du 1er ordre, calculabilité et λ-calcul*.- DIST, Traité des Nouvelle Technologies, série informatique.- Paris : Hermes, 1993. [Loc. : B.U.T.B.M.]

[3] Poirier Hervé.- Toute pensée est un calcul.- *Sciences & Vie*.- Février 2002, n°1013, p. 40-57. [Loc. : B.U.T.B.M.]

[4] Saint James Emmanuel.- *La programmation applicative : du LISP à la machine en passant pas le lambda-calcul*.- Paris : Hermes, 1993. [Loc. : B.U.T.C.]

[5] *Inventeurs et scientifiques, dictionnaires de biographies*.- Paris : Larousse, 1994. [Loc. : B.U.T.B.M.]

[6] Barendregt H..- *Functional programming and lambda calculus*.- Handbook of theoretical computer science, Volume B : Formal models and semantics, Chapter 7.- Elsevier Science Publishers, 1990. [Loc. : B.M.I.]

[7] Barendregt H..- *The lambda calculus, its syntax and semantics*.- Studies in logic, Volume 103.- Amserdam : North-Holland Publishing Comp, 1981. [Loc. : B.M.I.]

[8] Church Alonzo.- *The calculi of lambda-conversion*.- USA: Princeton University Press, 1941. [Loc. : B.M.I.]

[9] Curry H. and Feys R..- *Combinatory logic*. Amsterdam: North Holland, 1958. [Loc. : B.M.I.]

[10] Girard J.-Y., Lafont Y. et Taylor P..- Proofs and types.- Cambridge tracts in theoretical computer science, Volume 7.- Cambridge University Press, 1989. [Loc. : B.M.I.]

[11] Mitchell J..- *Type systems for programming languages*.- Handbook of theoretical computer science, Volume B : Formal models and semantics, Chapter 8.- Elsevier Science Publishers, 1990. [Loc. : B.M.I.]

[12] Mosses P..- *Denotational semantics*.- Handbook of theoretical computer science, Volume B : Formal models and semantics, Chapter 11.- Elsevier Science Publishers, 1990. [Loc. : B.M.I.]