NOTES DE COURS, Master Cognition

Giuseppe Longo
Laboratoire d'Informatique, CNRS - Ecole Normale Supérieure
e-mail: longo@dmi.ens.fr

Introduction et Histoire: de la Logique à Informatique[1]

L'Informatique, ou Science des Ordinateurs, a ses racines en Logique, en particulier dans la Logique Mathématique de ce siècle. Plus précisément, la recherche de rigueur en mathématiques (et dans le raisonnement) va de pair, au cours des trois derniers siècles, avec l'histoire de la Logique, mais aussi avec l'histoire de la simulation mécanique des activités humaines (y compris la pensée).

Les mathématiques grecques faisaient un grand usage des images: *théorème*, en grecque, veut dire "voir", "spectacle". Ils s'agissait d'images platoniciennes, "visions" de formes immuables, des idées, mais aussi de constructions par figures, faites avec la règle et le compas, démonstrations construites sur le sable, le feuil, par des méthodes diverses. L'unité axiomatique d'Euclide n'est que partielle et ouverte à une pluralité de technique de preuve; les déductions font appel à l'intuition à différents endroits.

Chez Descartes, l'analyse de la déduction mathématique fait partie de celle du raisonnement; la méthode cartésienne part des intuitions claires et distinctes et se développe par *l'énumération pas à pas* d'une chaîne ininterrompue d'évidences (Regulae ad directionem ingenii, 1629). Dans les Regulae Descartes reproche aux mathématiciens grecques l'absence d'une méthode uniforme de déduction, l'utilisation d'artifices parfois obscurs et ambiguës, souvent dépourvus de généralité. La géométrie, sous son impulsion, se transforme partiellement en *calcul*, dans le langage de l'algèbre: la preuve géométrique est donnée grâce à l'encadrement des figures sous les axes cartésiens; sa certitude et généralité sont données par celles des pas élémentaires du calcul des équations algébriques.

Pascal soulignera explicitement, dans la définition de ce qui est la rigueur mathématique, le rôle du *langage*. C'est dans le langage que l'on donne des règles pour les définitions, les axiomes et les démonstrations (Esprit géométrique, Art de persuader, 1654-58). Pour Hobbes, on compose les idées, les concepts, en fait les noms, à partir des plus simples : (Computatio sive Logica *dans* De Corpore, 1655). La connaissance est en effet dans les noms, pour ce nominaliste anglais, ainsi que dans la "manipulation" des mots et des concepts auxquels ils sont rattachés.

---

[1] Cette introduction est la version préliminaire d'une entrée pour le **Dictionnaire d'Histoire et de Philosophie des Sciences,** Press Universitaire de France, 1999 (pages 586--590).

La philosophie de Leibniz est un autre passage obligé pour cette brève histoire du "raisonnement rigoureux comme calcul dans un langage (symbolique)", origine historique et philosophique de l'Informatique. Le philosophe allemand, vrai précurseur de la logique formelle moderne, considère Hobbes comme un de ses plus importants inspirateurs, car . En fait, pour Leibniz le raisonnement doit être *formalisé* dans une *Lingua Characteristica* (l' "idéographie logique" de De Arte Combinatoria, 1666), forme générale de la pensée, mathématisée si possible, *calculus ratiocinator*, une *mathesis* universelle. Il souligne les insuffisances de la méthode cartésienne, qui est essentiellement psychologique et rejette toute utilisation de la logique. Voila le projet de l'entendement sous sa nouvelle forme de rigueur logico-linguistique qui commence à se préciser. Mais les formalismes pour calculer et déduire de Leibniz sont toujours riches en signification : les mots, les symboles de la lingua characteristica se réfèrent toujours à "quelque chose". Quant aux mathématiques, pour Leibniz elles sont d'abord un "ars demonstrandi", basée sur la logique. En fait, le XVIII siècle est l'époque de la naissance du calcul infinitésimal, auquel Leibniz contribue grandement. La certitude du fini est perdue, le regard dans l'espace des grecques ne suffit plus, ni l'intuition des vérités claires et distinctes de Descartes: il faut une nouvelle rigueur pour trouver des certitudes face à l'explosion conceptuelle des mathématiques, face à la manipulation de l'infini en acte, des limites, du continu. Selon certains historiens, cette *exigence de rigueur*, qui grandit avec la croissance du corpus mathématique, est à l'origne des innovations de Leibniz en logique, en particulier pour l'importance qu'il accorde aux "bonnes notations (formelles)" en mathématiques.

L'analyse de l'entendement comme calcul va de pair aussi aux théories mécanicistes et, en fait, aux mécanismes fantastiques de l'époque. Les horloges, les automates du XVII et du XVIII siècle sont des machines merveilleuses. Ils sont aussi mécaniques que le raisonnement certain, celui qui se déroule pas à pas, comme calcul algébrique ou logique. Le clavecin est comme le logicien, le logicien comme le clavecin, pour D'Alembert et Diderot; ou l'homme, dans l'acte de la pensé, a la même mécanicité de l'instrument musical.

Toutefois, au début du XIX siècle, la "pratique" des mathématiques reste encore incertaine : les notions d'infini, de limite, de continu sont encore flous. Beaucoup d'invention, mais aussi de désaccords et d'incertitudes: on donne encore des preuves incomplètes ou erronées (Cauchy, par exemple, dans les preuves du Théorème des Valeurs Intermédiaires (1821) et de la continuité de la somme d'une série de fonctions uniformément continues, n'explicite pas clairement les hypothèses, en fait la *définition* de (uniforme) continuité; Poinsot croit avoir démontré que toute fonction continue est différentiable). Malgré les efforts de clarification (Pascal, Leibniz ...), les mathématiciens ne paraissent pas savoir ce que c'est qu'une "bonne définition". En fait, on continue à faire confiance au regard dans l'espace, sur le plan, sur le feuil, comme fait Cauchy dans ses preuves ou Gauss et maints autres : il y a encore l'intuition du "voir", malgré les certitudes qui émanent du raisonnement logique et les succès des

automates, ces machines logiques, dans les salons et dans l'industrie. Le fondement de l'algèbre est (encore) dans la géométrie : on interprète même l'unité imaginaire $i = \sqrt{-1}$ et les nombres complexes, dont l'existence est une conséquence de la manipulation algébrique, sur le plan cartésien (Argand-Gauss). Mais comment faire pour manier avec certitude les concepts difficiles d'infini en acte et de continu, désormais omniprésents en mathématiques? Il faudra attendre Weierstrass (1861) et, plus encore, Dedekind et Cantor (1873-80) pour arriver à *re*-construire le continu, son infinité, ses opérations de limite, à partir du discret, de façon algébrique, représentation du continu intuitif de l'espace par la rigueur du *discret linguistique* des nombres entiers.

Les héritiers du nominalisme outre-Manche s'en prennent autrement. Pour Woodhouse, chef de file des algébristes anglais, la certitude est dans la manipulation "mécanique" des symboles algébriques (1801). Voila donc Boole et son algèbre de la Logique (Le calcul du raisonnement déductif, 1847; Les Lois de la Pensée, 1854): la Logique est une sorte d'Arithmétique binaire. Et Babbage, un mathématicien de la même école, avec ses machines analytiques. L'algèbre, le calcul déductif, sont dans la machine, dans les engrenages mus par la puissance de la vapeur. Des vraies machines à penser. Encore des "automates", mais qui permettent de résoudre des équations différentielles. Plus tard, en développant une idée de Babbage, on ajoutera une certaine flexibilité à ce *matériel* (hardware) rigide : des fiches en carton, remplaçables, permettront d'utiliser les mêmes engrenages pour des tâches différentes.

<IT1> La crise de la Géométrie: de Gauss à Hilbert.

Les mathématiciens du continent aussi durent bientôt revoir les fondements de leurs certitudes mathématiques. Lobacevskij, Bolyai et Riemann, entre 1835 et 50, proposent les géométries non-euclidiennes, basées sur les deux négations logiquement possibles du cinquième axiome d'Euclide. Ils montrent donc que l'espace n'est pas si certain, canonique, unique: il n'y a pas une "intuition pure de l'espace en soi", en tant que variété euclidienne. Gauss les avait précédés de quelques années, mais il n'avait pas osé croire ou exprimer ce bouleversement de l'intuition géométrique.

Il ne reste alors que le langage, bien au-delà des intuitions de Leibniz, comme fondement certain des mathématiques. Pourvu qu'il soit basé sur une logique, sur La Logique. G. Frege fonde les mathématiques sur un calcul logique extrêmement puissant. Il propose (Idéographie, 1879; Fond. de l'Arithmétique, 1884) un langage logique, décrit avec la rigueur formelle que sera la nôtre et dit du "premier ordre", pour la *quantification* des variables individuelles, les "pour tout x" et "il existe x" utilisés si mal dans le langage commun, si souvent informellement importé dans la pratique mathématique. Pour Frege, le calcul arithmétique, coeur des mathématiques, est une déduction logique dans un système logique du premier ordre; le *fondement* et la *signification* du calcul algébrique sont dans la logique.

D. Hilbert, au contraire, ne veut donner aucun role au sens, dans la déduction : la certitude du calcul est dans l'*absence de signification*. La manipulation de suites finies de symboles par des règles entièrement formalisées donne au langage mathématique la certitude et le fondement, elle exprime et garantie l'invariance et la stabilité des concepts mathématiques, elle les soustrait aux ambiguïtés sémantiques, aux incertitudes et aux paradoxes de l'infini. Plus précisément certitude et fondement doivent se baser sur des *preuves de cohérence* ou d'absence de contradiction formelle (1900). Ces preuves doivent être données dans une mathématique (la *métamathématique*) dont *l'objet de travaille* sont les mathématiques. La métathéorie est décrite dans un métalangage qui permet de parler du langage objet propre aux mathématiques. Dans son programme fondationnel, Hilbert conjecture que l'on pourra démontrer aussi la *décidabilité* et la *complétude* (tout énoncé ou sa négation est démontrable) des axiomatiques formelles (des équations entre signes, des règles de déduction comme réécriture ou transformation de suites de signes dans d'autres) adoptées pour les différentes branches des mathématiques. En fait, Hilbert reprend en Logique l'hypothèse de Laplace sur la prédictibilité en Physique: les théories formelles des mathématiques permettent de "recouvrir complètement" et "décider" les propriétés des structures mathématiques, exactement comme les mathématiques de Laplace devaient "recouvrir" ou "prévoir" (décider dans l'espace et le temps) l'évolution des structures physiques. Les systèmes (axiomatiques) de signes pourrons donc décrire complètement le monde (des mathématiques, la géométrie en particulier). Une fois détaché le langage et ses signes du sens, le siècle est prêt pour des machines à penser, manipulatrices de symboles sans signification.

<IT1> La déduction effective et la calculabilité.

Au cours des années '30 les logiciens arrivent à préciser la notion hilbertienne de déduction formelle comme "procédure effective". On donne dans ce but différentes notions *de fonction calculable* ou algorithmique et on code les formules du langage par les nombres entiers (la *gödélisation*). Or, dans une théorie (formelle), les notions de formule et de démonstration, comme déductions pas à pas d'une formule à l'autre, sont métathéoriques, tandis que les nombres et les fonctions sont normalement décrits dans la théorie. Grâce à la gödélisation, vraie percé de K. Gödel, toute démonstration (métathéorique) devient alors une fonction (théorique) : une fois codés par des nombres les énoncés d'un langage formel donné, une démonstration est un algorithme ou fonction *récursive* (J. Herbrand et K. Gödel, 1930 et 1931) ou *lambda-calculable* (Church, 1932), qui va des axiomes aux thèses (codés), entièrement définie par les règles de déduction du système formel, sous forme d'équations et de règles de réécriture. Si l'entendement humain est déduction logique, on est presque arrivé à la mécanisation de la pensé : manquent seulement les machines pour réaliser pratiquement ces algorithmes. Toutefois, c'est exactement la puissance expressive du codage et de la déductibilité formelle comme calculabilité qui permettent de démontrer les limites des la

déduction formelle/mécanique. Dans le *Théorème d'incomplétude* de 1931, Gödel (voir [Girard, 1989], pour le texte de Gödel et une discussion), grâce au codage, démontre que l'on peut transférer dans l'Arithmétique elle-même toutes les procédures logiques et finitaires de preuve (comme fonctions calculables). En particulier, en codant dans la théorie la phrase métathéorique "cette phrase n'est pas démontrable" (G, disons), Gödel démontre que, si l'Arithmétique est cohérente, on ne peut pas ... démontrer G (ni sa négation). Par conséquent, l'Arithmétique formelle, sur laquelle Frege et Hilbert avait basé l'édifice des mathématiques, n'est pas complète, ni décidable; on ne peut pas en démontrer la cohérence non plus, car l'énoncé de la cohérence et le fait qu'elle implique G sont aussi codable. Le programme d'Hilbert échoue ainsi que son rêve de mécanisation "en principe" des mathématiques. Toutefois, l'Informatique moderne naît, car deux autres percés fondamentales voient le jour.

En 1936, A. Turing arrive à concevoir des machines qui manipulent des suites de symboles sans significations: les *Machines de Turing* sont des machines mathématiques, pure abstraction conceptuelle sans contrepartie physique, mais bien plus "concrètes" des fonctions calculables. Il les construit en reprenant à l'intérieur de son système la distinction entre langage et métalangage et entre syntaxe et sémantique (symbole et signification): elles se basent sur une distinction cruciale entre hardware et software (matériel et logiciel). Le matériel est une bande (potentiellement infini) sur lequel une tête d'écriture/lecture inscrit/efface/lit des symboles. Dans son logiciel, basé sur un langage très simple, Turing écrit des programmes, pour l'écriture/lecture de la tête et son mouvement, un pas à droite ou un pas à gauche. Il utilise aussi l'idée du codage de Gödel pour imaginer un programme qui exécute les calculs de n'importe quel autre programme, une fois codé comme nombre : sa *Machine Universelle* est un compilateur, dans la terminologie informatique contemporaine, elle reçoit comme argument un programme (codé) et un nombre, décode le premier et l'applique au deuxième. Encore une fois, ce travail est motivé par la Logique: Turing étudiait dans son article le problème de la décidabilité des théories logiques, posé par Hilbert.

L'autre grand résultat de 1936-37, dû à Kleene et Turing, est l'équivalence entre tous les différents approches au calcul déductif. Résultat très surprenant, car rien ne pouvait garantir que des systèmes si divers auraient permit de définir exactement la même classe de fonctions. Si cela confirmait le caractère "absolu" du théorème d'incomplétude de Gödel (sa notion de preuve finitaire ou algorithmique était donc tout-à-fait générale, car invariante par rapport au formalisme), il convainquit les logiciens qu'ils avaient touché à un vrai absolu: la notion définitive de calcul et déduction effectifs. Les Machines de Turing étaient enfin les vraies machines à penser, dont on rêvait depuis des siècles! Et, même si le théorème d'incomplétude de Gödel démontrait les limites à la déduction formelle, ces limites devaient être propres aussi à l'homme, le "human computor" dans l'acte de la déduction logique. C'est au moins ce qui pensaient et pensent maintes logiciens (les formalistes au moins), tandis que certains d'entr'eux (Gödel, par exemple) et la plupart des mathématiciens ont préféré plutôt croire à

l'existence de vérités mathématiques, inaccessibles aux systèmes formels, auxquelles l'esprit humain a un accès direct (un platonisme plus ou moins naïf, encore aujourd'hui très répandu).
Voila donc un itinéraire possible de l'analyse de la rationalité humaine: de l'énumération pas à pas du raisonnement cartésien, développement de l'intuition spatiale, aux langages riches de signification géométrique ou logique de Leibniz, Boole et Frege, aux purs calculs de symboles sans signification de Hilbert et Turing, mouvement à gauche et à droite d'une machine pour écrire/effacer des 0 et des 1. La rationalité, une fois restreinte et codée dans le langage, ensuite abstraite de la signification, est enfin réifié dans une machine. Ces machines, les ordinateurs, qui ont changé notre vie.
Évidemment d'autres parcours étaient possibles. Même à l'intérieur du débat sur les fondements des mathématiques, Riemann, Mach et Poincaré et d'autres, entre 1850 et le début de ce siècle, cherchent ces fondements dans notre action dans le monde sensible, dans la vision et le mouvement, bases de la Géométrie, et dans l'intuition de l'ordre et de l'induction, fondements de l'Arithmétique. Mais leurs intuitions philosophiques, aussi vagues que profondes, ont été balayées, dans ce siècle, par la rigueur et la force de l'analyse conceptuelle de Frege, par la précision des propositions mathématiques de Hilbert. Les (méta)mathématiques de ce dernier ont su en fait nous donner une notion de rigueur mathématique bien solide, la rigueur formelle, indépendante de la signification, et des machines extraordinaires.

<IT1> L'intelligence et les Machines.
Au cours des années '40, Turing et un autre logicien, J. von Neumann, réaliseront les premiers ordinateurs électroniques. En 1950, Turing propose un "jeu de l'imitation", dans lequel on n'arriverait pas à distinguer un être humain d'un ordinateur (voir [Turing-Girard, 1995]); son article est considéré le point de départ de *l'Intelligence Artificielle* (IA). Turing fait des nombreuses hypothèse, avec beaucoup de prudence: le jeu comporte seulement des manipulations de symboles discrets, car sa machine est à "états discrets", donc même le système nerveux n'en est certainement pas une, le jeu compare ce qui est codable, dans l'intelligence humaine, par des systèmes finitaires (discrets). L'IA, dans sa version "forte", et le *fonctionnalisme* (dans sa forme logico-computationnelle) fera de l'hypothèse du codage le socle sur le quel construire machines et thèses philosophiques. En fait, pour les logiciens formalistes, les systèmes et calculs formels peuvent seulement "reconstruire", a posteriori, le corpus mathématique, bâtit par les maints chemins possibles de la créativité; en IA, on va plus loin: c'est toute notre intelligence du monde qui procède par des calculs logiques, puisque (Hodges dans [Herken,1992]) et on peut enfin (Johnson-Laird,1993).
D'autres Philosophies de la Connaissance sont en train de se préciser, autour ou au delà des ordinateurs d'aujourd'hui. Pour les *connexionnistes*, les "réseaux de neurones"

mathématiques (voir [Hertz,1991]) simulent la plasticité, le continu, le flou du système nerveux, mais ces réseaux ne sont pas encore des vraies machines, quoique des ordinateurs, les "connection machines" en particulier, peuvent en implementer certains modèles. D'autres encore envisagent plutôt une revitalisation des intuitions de Riemann, Mach ou Poincaré.

<IT1> Les architectures et les algorithmes.

Mais revenons à la structure physique des ordinateurs de Turing et von Neumann. Ces machines n'ont plus rien à voir avec les "automates" du passé, de Vaucanson à Babbage, car leur performance n'est plus entièrement inscrite dans les engrenages: la distinction entre logiciel et matériel les rend programmables, en fait "universelles" comme la Machine de Turing. Deuxièmement, leurs circuits électroniques réalisent la logique binaire de Boole, en simplifiant grandement les taches trop élémentaires du "matériel" abstrait de Turing: les portes logiques des circuits sont exactement des combinaisons des "and" et "or" booléens. *L'architecture* de ces nouvelles machines se développe donc rapidement, en empruntant des outils de la Physique, mais suivants d'abord les lignes directrices proposées par la Logique: Boole (les circuits), Turing (la structure séquentielle), von Neumann (les "random access machines"). Dans les décennies suivantes le mariage entre Logique et Physique continuera, quand aux architectures, mais, tout récemment, la Physique a posé des défis entièrement nouveaux: on y reviendra.

Un autre grand secteur disciplinaire de l'Informatique concerne les *algorithmes* (voir [van Leeuwen, 1990]). Encore une fois, les algorithmes pour la programmation et la solution des problèmes mécanisables se basent sur les différentes classes de calculs formels définies dans les années '30, auxquels se sont ajoutées d'autres systèmes (Post, Markov ...), formellement équivalents. Un problème crucial de l'algorithmique est celui la *complexité de calcul* : quelle quantité de ressource (espace de mémoire ou temps, par exemple) est-elle nécessaire pour faire un calcul? La théorie de la complexité computationnelle a utilisé quelques idées de la Logique (les tableaux de vérité du calcul propositionnel, par exemple), mais surtout des outils des mathématiques discrètes ou combinatoires, souvent originaux. Même la Théorie des Nombres a fait son entré massive dans le domaine, en raison surtout de la *Cryptographie*, ou théorie de la "communication et mémorisation de données en présence d'adversaires", codage de messages rendus impénetrables par la complexité algorithmique du décodage.

<IT1> Les langages

Les algorithmes qui fonctionnent sur des architectures ont besoin d'etre décrits dans des langages. Dans le cas des langages de programmation, en fait formels, non seulement l'origine, mais aussi leurs développements sont marqués par la Logique.

Les travaux de Curry (1929) et Church (1932) constituent le fondement de la *Programmation Fonctionnelle*. Curry propose un calcul logique ou combinatoire, la

Logique Combinatoire : deux seuls symboles K et S, réglés par deux axiomes (deux équations), engendrent une algèbre extrèmement puissante. Par ces manipulations sans signification on pourra calculer, modulo un codage des nombres par des suites de K et S, toutes les fonctions récursives (les résultats d'équivalence de 1936-37). Church invente un système de notations logiques pour les fonctions mathématiques (le *λ-calcul*): une règle explicite de réécriture ("dans f(x,y), remplace la variable x par la lettre a") suffisent encore une fois à calculer toutes les fonctions Turing-programmables. En fait, l'équivalence entre Machines de Turing et les fonctions récursives de Herbrand et Gödel sera démontré en passant par le λ-calcul, par des méthodes qui en feront l'ancêtre du LISP (1960), Scheme et ML, des langages fonctionnels de programmation bien connus. D'autre part, Herbrand, en 1930, définit un "modèle minimal" pour la quantification logique, basé sur la syntaxe. Une démonstration devient alors un contrôle de vérité sur des "instances", dans ce modèle. Ce résultat a posé les bases mathématiques de la *Programmation Logique*.

Avec les Machines de Turing on passe des notations mathématiques à la programmation d'un calcul sur un support "matériel". Les règles qui constituent le "logiciel" sont des vraies instructions, les ordres de ce qui deviendra la *Programmation Impérative*: les "go to .... ", "do ... until" du Fortran et de maints autres langages.

Parmi ces trois styles de programmation (fonctionnel, logique et impératif), qui ont caractérisé une grande partie de la programmation jusqu'à nos jours, les deux premiers on continué à avoir des interactions importantes avec la Logique. En programmation fonctionnelle, en particulier, si un programme prend comme arguments des éléments de (l'ensemble) A et donne des valeurs dans B, on dira qu'il est de "type" $A \to B$, ou qu'il calcule une fonction f de A dans B. Donc on peut considérer ce $A \to B$ comme un ensemble de programmes (ou de fonctions), mais aussi comme implication logique, "A implique B", car ces programmes, ces calculs en fait, sont des preuves de propositions logiques : c'est l'idée guide des année '30. C'est à dire, f est *aussi* une preuve qui prend toute preuve de A, où A est maintenant à considérer comme un prédicat, et la transforme dans une preuve de B. Mais alors, f est une preuve de l'implication logique $A \to B$ (cette interprétation fonctionnelle de l'implication "→" est une version *intuitionniste* de la Logique). Voila donc comme A, B, $(A \to B)$ sont des types (des ensembles de données informatiques), mais aussi des propositions. Et f est (le nom de) un programme, mais aussi une fonction et une preuve.

Mais il y a, en mathématiques, une autre "théorie des fonctions" qui intéresse les informaticien: la Théorie des Catégories. Elle ne demande, pour commencer, que des objets, A, B, ... et des morphismes $f : A \to B$ entre objets, qui se composent et contiennent l'identité. Une autre "analogie", qui s'ajoute à la précédente, entre différentes structures logico-mathématiques. Ces analogies sont profondes et donnent en fait des "isomorphismes" bien précis entre Types, Propositions et Objets, dans le sens de la Théorie

des Catégories. En bref: Système fonctionnel = Système déductif = Catégorie. Ces correspondances mathématiques permettent de démontrer ou utiliser des résultats dans un des systèmes et de les transférer dans un autre. Par exemple, des démonstrations sur les *termes* typés prouvent la *terminaison* pour certains calculs en programmation par des méthodes "d'élimination des coupures", une sorte de preuve de "pureté de méthodes" en Logique: on peut éliminer, dans les *preuves*, tout ce qui n'est pas explicitement utilisé dans la thèse. En plus, les structures catégoriques donnent des *modèles* (ou une "sémantique mathématique" des Types et des Propositions, comme objets de catégories), c'est à dire des interprétations des systèmes formels qui enrichissent les langages de structures et ... d'idées. Une propriété des modèles, qui n'est pas démontrable dans le langage, peut suggérer des *extensions* du langage (e.g. les définitions récursives dans le langages ML d'Edinburgh, 1976, ont été suggérées ou permises par la *sémantique dénotationnelle*, voir dessous). Mais l'interprétation peut suggérer des variantes des systèmes de départ: en étudiant des modèles d'une extension du λ-calcul de Church, J.Y. Girard a pu concevoir un système différent des logiques classiques et intuitionnistes (la *Logique Linéaire*, [Girard,1987]), car ces modèles contiennent des fonctions "linéaires". Et encore: puisque les preuves sont des programmes (et viceversa), alors une déduction, pas à pas, est une *synthèse de programmes*. A l'inverse, un programme développe une preuve: voila un outil essentiel à la *démonstration automatique*. Si, par contre, on part avec des termes ou des programmes sans types, c'est une déduction qui leur "donne" des types. Cette dernière technique fournit un contrôle efficace, quoique partiel, de la correction d'un programme, comparable au "contrôle de dimension" des équations en physique : si le programme est "typable", il est presque sûrement "correcte", comme en physique, si dans une équation à la fin d'un calcul la dimension est préservé, le calcul a des bonnes chance d'être bien fait. Toutefois, en λ-calcul et dans certains langages de programmation (le LISP, par exemple) on peut considérer comme bien formés aussi les termes sans types, comme $x(x)$ ou l'auto-application, ce qui permet de donner dans le langage formel ou de programmation les définitions récursives ou définir une fonction en termes d'elle-même (par exemple, le "factoriel": $n! = 1\times2\times3 \ldots \times n = n\times(n-1)!$ est une fonction (primitive) récursive). Mais, si on peut formellement appliquer une fonction ou programme f à soi même et poser $f(f)$, quel est le sens mathématique de cette notion de "fonction"? Voilà une notation symbolique sans signification mathématique: y a-t-il en fait un univers mathématique où les fonctions peuvent s'appliquer à elles mêmes? Il faut un espace X, une structure "géométrique", où on puisse identifier fonctions et arguments. Il suffit en fait que X satisfasse l'équation: $X = X \to X$ ou $X = X^X$ (X puissance X), où $X \to X$ est un ensemble ou "domaine" de fonctions de X dans X (dit domaine exponentiel). On sait, depuis Cantor, qu'aucun ensemble, fini ou infini, n'est égal ni isomorphe à son exponentiel (sauf pour X égale à un singleton ou $X = 1$, car $1 = 1^1$, un cas inintéressant, puisque dans X on doit pouvoir interpréter tous les nombres et les

fonctions calculables). Une première solution a été donné par D.S. Scott en 1970: il construit une catégorie d'espaces topologiques tels que le domaine structuré $X \to X$ des fonctions continues sur X, est isomorphe à l'espace X lui-même. Grâce à ces "domaines de calculabilité", la *Sémantique Dénotationnelle* des langages de programmation permet de résoudre une large classe d'équations, bien plus complexes, qui sont souvent utilisées en théorie de la programmation.

<IT1> Les systèmes distribués, interactifs et asynchrones

La Physique permet aujourd'hui de distribuer les ordinateurs sur des grands réseaux de communication et leur donne une grande vitesse de calcul, grâce à une électronique et à des "puces" aux performances extraordinaires, quant à stockage et vitesse. Or, pour que deux processus interagissent d'une façon fiable au cours d'un calcul distribué, il faut qu'ils communiquent dans un temps de l'ordre de grandeur de l'unité de temps de calcul (quelques nanosecondes). La vitesse de la communication électrique ou de la lumière, qui est limitée, et l'absence d'une mesure absolue du temps (asynchronie essentielle des différents systèmes de référence physique) posent des problèmes physiques (relativistes, en fait) tout à fait nouveaux. Les langages de programmation qui essayent de s'adapter à ces nouvelles donnes paraissent s'échapper aux paradigmes proposés par les pères fondateurs de la Logique moderne: les Langages Orientés aux Objets, par exemple, où concurrence et interaction jouent un rôle central. Toutefois, les informaticiens utilisent encore des systèmes logiques pour essayer de traiter mathématiquement les nouveaux problèmes. La *Logique Modale*, avec ses aspects de logique temporelle, permet de traiter des problèmes pointus concernant le temps dans des systèmes distribués et concurrents (linear time, branching time ... pour la concurrence et le parallélisme). Le *mu-calcul*, qui en suit, donne des points fixes "maximaux", strictement reliés aux structures de la Théorie des Ensembles "mal-fondés", c'est à dire des ensembles qui admettent des chaînes descendantes d'ensembles l'un inclus dans l'autre, comme certains processus infinis de calcul (les systèmes d'exploitation, par exemple). Les *Catégories Fibrées* sont utiles comme modèles des algèbres de processus (processus asynchrones). Souvent ces systèmes logiques ou catégoriques servent juste pour une unification de la pluralité des langages utilisés, ce qui n'est pas un moindre succès. Parfois on obtient des vraies percés, des extensions, des langages nouveaux et intéressants. De plus en plus des méthodes géométriques paraissent essentielles, avec leur autonomie par rapport à la Logique, historiquement centrée sur les langages séquentiels et ses structures finitaires: en Logique Linéaire la géométrie fait désormais partie de la structure spatiale des preuves; les mathématiques des systèmes dynamiques, les lieux du temps et de l'espace physique, font leur apparition en Sémantique Dénotationnelle et dans l'analyse des systèmes distribués, concurrents et asynchrones.

<BIBLIOGRAPHIE>

- *Handbook of logic in computer science, vol. 1 - 4* (1992 - 1995) ouvrage coll. (Abramsky S. et al. eds) Claredon, Oxford.
- Asperti A., Longo G. (1991) *Categories, Types and Structures: an introduction to Category Theory for the working computer scientist* M.I.T. Press.
- Girard J.-Y. (1987) Linear Logic *Theoretical Comp. Sci.*, 50 (1-102).
- *Le Théorème de Gödel* (1989) ouvrage coll. (Girard ed.), Seuil.
- van Heijenoort J. (1967) *From Frege to Gödel*, Harvard Univ. Press.
- *The Universal Turing Machine,* (1995) ouvrage coll. (Herken R. ed.), Springer-Verlag.
- Hertz J. et al. (1991) *Introduction to the theory of neural computation*, Addison-Wesley.
- *Handbook of Theoretical Computer Science vol. A: Algorithms and Complexity* (1990) ouvrage coll. (van Leeuwen J. ed.), Elsevier.
- Mitchell J.C. (1993) *Introduction to Programming Language Theory*, M.I.T. Press.- Turing A., Girard J.Y. (1995) *La machine de Turing*, Seuil.