Sommaire

Introduction

1 / En 1936, dans un article intitulé « Sur les nombres calculables », Alan Turing introduit en logique une machine abstraite, mathématiquement définie, mais également présentée comme un dispositif susceptible d'être mécaniquement réalisé. Les machines de Turing, conçues à l'origine pour la solution de problèmes purement logiques, sont également, aujourd'hui, un modèle théorique des calculateurs universels que sont les ordinateurs. L'usage d'une notion de machine dans un domaine traditionnellement considéré comme une science du raisonnement correct, un instrument d'analyse du langage et des énoncés, et même un exercice du jugement ou un art de penser, pourrait passer pour une curiosité s'il était marginal ou singulier. Mais les machines abstraites, ainsi que d'autres formalismes élaborés pour l'étude des notions de calcul et de procédure effective, sont devenus l'un des principaux outils du logicien et, aussi bien, l'un des objets sur lesquels porte son analyse. Ce que l'on appelle « théorie de la calculabilité », à l'origine de laquelle se trouvent, notamment, les machines de Turing, est à la fois l'une des grandes parties de la logique contemporaine et l'un des fondements théoriques de l'informatique. Au cours des quelques décennies qui suivirent l'année 1936, les machines abstraites devinrent un objet commun à la logique et à l'informatique théorique, en sorte que les deux domaines sont maintenant devenus indissociables. Les problèmes du logicien sont très souvent, de fait, ceux que posent, à un niveau théorique, les calculateurs et autres machines à traiter l'information, et la recherche en informatique

rejoint très souvent des questions de logique. A cet égard, la théorie de la calculabilité et les machines logiques de Turing sont certainement l'exemple le plus clair et le plus évident des liens qui existent aujourd'hui entre la logique et l'informatique ; mais il s'en faut de beaucoup qu'elles constituent un cas exceptionnel ou singulier. Dès que l'on examine les choses de ce biais, les exemples se présentent à foison. Il est piquant de constater que les machines de Turing, modèle théorique de l'ordinateur, sont introduites en logique plus de dix ans avant la réalisation effective du premier calculateur universel à programme enregistré, l'IBM SSEC[1], construit entre 1945 et 1948. Mais l'étonnement grandit lorsque l'on s'aperçoit que de nombreuses notions, constructions et techniques inventées par les logiciens pour la solution de questions de logique, bien avant la conception des premiers ordinateurs, se sont révélées être des idées fondamentales pour le développement des calculateurs universels, des langages formels et des programmes écrits pour ces machines.

2 / Citons, parmi les exemples les plus évidents, quelques-unes des constructions, quelques-uns des problèmes et des résultats de logique devenus, ultérieurement, des éléments essentiels de l'informatique théorique.

• Depuis la *Begriffsschrift* de Frege, et les *Principia Mathematica* de Russell et Whitehead, une grande part de l'activité des logiciens consiste à donner une expression formalisée des énoncés logiques et mathématiques. Dans ces deux œuvres majeures, il s'agissait de présenter ainsi de manière systématique une certaine catégorie de vérités. Avec Hilbert, l'entreprise de formalisation est poursuivie, mais prend un sens différent : la conception de langages et de systèmes formels doit permettre l'étude de leurs propriétés syntaxiques et sémantiques d'un point de vue métalinguistique. Les systèmes hilbertiens sont la donnée d'un alphabet formel, de règles de formation des mots

1. Le *Selective Sequence Electronic Calculator* ne doit pas être confondu avec l'ENIAC, plus connu, qui n'était pas à proprement parler un ordinateur. Nous reviendrons sur ce point au chapitre III.

A cet égard, la théorie
Turing sont certaine-
des liens qui existent
e ; mais il s'en faut de
nnel ou singulier. Dès
emples se présentent à
machines de Turing,
duites en logique plus
emier calculateur uni-
, construit entre 1945
l'on s'aperçoit que de
ques inventées par les
ogique, bien avant la
révélées être des idées
alculateurs universels,
s pour ces machines.

vidents, quelques-unes
nes et des résultats de
ts essentiels de l'infor-

Principia Mathematica de
l'activité des logiciens
des énoncés logiques et
ires, il s'agissait de pré-
taine catégorie de véri-
on est poursuivie, mais
gages et de systèmes for-
s syntaxiques et séman-
es systèmes hilbertiens
s de formation des mots

confondu avec l'ENIAC, plus
. Nous reviendrons sur ce point

sur cet alphabet, d'un ensemble d'axiomes, et de règles d'inférence. Ils permettent ainsi d'écrire des formules et des preuves qui sont les traductions formelles, respectivement, des propositions et des démonstrations écrites dans les mathématiques usuelles. Ces constructions sont purement formelles dans la mesure où leur définition et leurs relations mutuelles peuvent être considérées indépendamment de leur signification. A la limite, un système formel est réductible à une procédure mécanique quelconque, permettant de produire des formules. Selon l'expression de Gödel lui-même :

> Le travail de Turing donne une analyse du concept de « procédure mécanique » (c'est-à-dire « algorithme » ou « procédure de calcul » ou « procédure combinatoire finie »). Il montre que ce concept est équivalent à celui de « machine de Turing ». Un système formel peut simplement être défini comme une procédure mécanique quelconque pour la production de formules, appelées formules démontrables. (...) Ce sens est requis par le concept de système formel, dont l'essence est que le raisonnement est complètement remplacé par des opérations mécaniques sur des formules[1].

Or les langages de programmation ne sont rien d'autre que des systèmes formels de signes, définis par un alphabet, des règles morphologiques et une syntaxe, qui permettent de communiquer à une machine l'expression formelle de connaissances, d'instructions et d'algorithmes. Dans les systèmes formels de la logique et dans les langages informatiques, les preuves et les programmes sont réduits à des suites de suites de signes, susceptibles d'être produites, au moins en théorie comme le suggère Gödel, par une procédure mécanique. La différence vient de ce que les uns permettent d'écrire des démonstrations, et les autres des algorithmes. Or nous verrons que cette différence n'est pas si grande qu'on pourrait le penser : certaines preuves, dites « constructives », peuvent, comme les programmes, être elles-mêmes considérées comme des procédures mécaniques.

• La théorie des modèles est une partie de la logique qui traite de l'interprétation des objets formels (langages, théories, formules et

1. On undecidable propositions of formal mathematical systems, 1934, Post-scriptum, ajouté le 3 juin 1964, *in* Kurt Gödel, *Collected works,* vol. I, New York, Oxford UP, 1986, p. 369-370.

preuves). Elle définit et construit une sémantique des systèmes formels, étudie les propriétés des domaines d'interprétation, traite de la vérité et de la validité des énoncés. Or la similitude entre les systèmes formels de la logique et les langages informatiques conduit à concevoir une sémantique des programmes, ainsi que des interprétations et des modèles pour ces langages et pour les programmes.

• Les théorèmes d'incomplétude de Gödel et d'indécidabilité de Church, qui déterminent certaines limitations des systèmes logiques et des théories mathématiques formalisées, ont des conséquences directes sur les possibilités et les limites des algorithmes et sur les capacités des machines informatiques.

• Les programmes sont souvent conçus comme une suite d'instructions écrites dans un langage déterminé, instructions que la machine est capable d'effectuer. Mais le calcul des prédicats[1], souvent considéré comme la partie la plus fondamentale de toute la logique formelle contemporaine, permet l'écriture d'une autre forme de programmes, constitués d'une base de connaissances sur laquelle sont définies certaines opérations. Il s'agit de la programmation logique, dont le langage informatique Prolog constitue l'une des réalisations particulières. Au fondement de cette conception de la programmation se trouvent d'une part une version particulière du calcul des prédicats (pour l'expression des connaissances), ainsi que plusieurs techniques de théorie de la démonstration dues à Gentzen (la règle de coupure du calcul des séquents) et à Herbrand (le théorème fondamental et les modèles de Herbrand)[2].

• En 1932, Church conçoit un système logique (modifié en 1933) qui est destiné à la formalisation des mathématiques. Kleene et Rosser démontrent cependant dès 1935 qu'il est possible

1. Ici, le mot « calcul » signifie que dans un système formalisé, la validité d'une déduction peut être vérifiée abstraction faite de tout recours à l'intuition, par une inspection purement mécanique des suites de formules. L'expression « *Funktionenkalkül* » (calcul fonctionnel), utilisée par Hilbert et Ackermann en 1928 dans les *Grundzüge der theoretischen Logik*, fut remplacée par « *Prädikatenkalkül* » (calcul des prédicats) en 1938 dans la seconde édition de cet ouvrage.
2. Cf. par exemple Jean Gallier, *Logic for Computer Science. Foundations of Automatic Theorem Prooving*, New York, Harper & Row, 1986.

de dériver une contradiction à partir de l'ensemble des postulats posés par Church dans son système. Pourtant, celui-ci n'est pas entièrement abandonné. Plusieurs idées sont retenues et permettent de construire un nouveau formalisme appelé « lambda-calcul », dans lequel sont représentées et calculées des fonctions. Or, ce formalisme, utile à la résolution de plusieurs problèmes majeurs de logique se révèle être équivalent aux machines de Turing et peut être interprété comme un langage de programmation.

• En 1935, Gentzen introduit la déduction naturelle et le calcul des séquents, deux nouvelles manières de formaliser les preuves mathématiques, rapidement devenues des techniques essentielles en théorie de la démonstration. La déduction naturelle est particulièrement bien adaptée à l'écriture des preuves en logique intuitionniste, logique qui n'admet pas le principe du tiers exclu (A ou non-A) à titre de formule valide. Or, il est possible d'établir une correspondance entre les preuves écrites en déduction naturelle et les programmes écrits dans le lambda-calcul, en sorte que les preuves puissent être considérées comme des programmes, et les propositions ou formules prouvées comme des *types de données*, au sens où l'on entend ce terme en informatique (on pense, par exemple, aux types des entiers, des booléens, des listes, des arbres binaires, etc.). Il s'agit de la correspondance de Curry-Howard, sur laquelle nous reviendrons longuement au chapitre VIII.

Ces quelques exemples, présentés ici de manière allusive, et dont la liste est loin d'être close, ne peuvent donner qu'une première idée de l'importance des relations qui existent entre la logique et l'informatique[1]. Si l'on admet cependant l'existence de

1. Importance attestée, d'ailleurs, par une abondante littérature : S. Abramsky, Dov M. Gabbay, T. S. E. Maibaum, *Handbook of Logic in Computer Science,* vol. 1 à 6, Oxford, Clarendon Press, 1992- ; Bruno Courcelle, *Logique et informatique. Une introduction,* Rocquencourt, Inria, 1990 ; Gallier, *op. cit.* ; René Lalement, *Logique, réduction, résolution,* Paris, Masson, 1990 ; Richard Lassaigne et Michel de Rougemont, *Logique et fondements de l'informatique,* Paris, Hermès, 1993 ; P. Odifreddi (ed.), *Logic and Computer Science,* Londres, Academic Press, 1990 ; Jacques Stern, *Fondements mathématiques de l'informatique,* Paris, McGraw-Hill, 1990 ; etc. A ces ouvrages s'ajoutent les actes de colloques comme l'*Annual IEEE Symposium on Logic in Computer Science.*

telles relations, en attendant qu'une analyse plus détaillée les mette en évidence, si l'on ajoute, d'autre part, que les programmes pour ordinateurs peuvent être considérés comme des machines abstraites, dont certaines sont de surcroît des machines universelles (en un sens qui devra lui aussi être précisé), nous sommes rapidement conduits à l'un des points nodaux de notre réflexion : *la logique ne traite pas seulement de la vérité, du raisonnement et du langage, mais également du calcul, des algorithmes et des machines*. L'objet du présent travail est à la fois d'expliciter le sens de cette affirmation, d'en mesurer la portée, et de tenter d'en dégager les implications philosophiques.

3 / Les philosophes se sont beaucoup intéressés à la logique dans ses dimensions épistémologiques et ontologiques, en tant qu'elle énonce certaines conditions générales de la connaissance, du raisonnement correct et de la pensée vraie, qu'elle détermine une grammaire de la réalité ou de notre compréhension des choses, ou encore, en tant qu'elle rend possible une analyse du langage. Cela se comprend aisément, dans la mesure où la philosophie se donne comme objet la connaissance elle-même, ou les formes de notre appréhension du monde ; et cela se comprend d'autant mieux que le langage est considéré comme l'une des principales conditions de possibilité de la connaissance, ou lorsque l'analyse logique du langage devient une voie privilégiée de la méthode philosophique. Les philosophes ont également considéré la logique en tant qu'elle prend la forme d'un calcul, ou en tant qu'elle construit des langues formulaires et des systèmes formels dont elle étudie les limites, les capacités et les propriétés. De ce point de vue, si le philosophe s'intéresse à la logique, c'est aussi parce qu'elle semble réaliser pour la première fois de manière effective et convaincante une idée qui est pourtant loin d'être nouvelle dans l'histoire de la philosophie, celle d'un rapprochement possible entre le calcul et le raisonnement, idée que l'on trouve exprimée, déjà, en des sens parfois différents, chez des auteurs comme Hobbes, Leibniz ou Condillac, mais qui n'avait jamais pris la forme d'une réalisation achevée, malgré les

efforts de Leibniz, de Lambert ou d'autres[1]. Et l'intérêt philosophique des calculs logiques qui formalisent certains raisonnements a paru d'autant plus grand que ces calculs furent prolongés, à partir de la fin des années cinquante, par la construction de machines capables de raisonner, et par le projet d'une intelligence artificielle. On caressa dès lors, et l'on caresse encore, l'espoir d'une psychologie cognitive, d'une compréhension des phénomènes de l'esprit inspirée des machines pour le traitement de l'information, par construction de modèles dits « computationnels » du mental. Cela explique donc aussi l'intérêt que les philosophes ont pu porter à la logique et à ses implications informatiques. En revanche, les philosophes ont accordé peu d'attention au fait, qui semble pourtant digne d'intérêt, voire paradoxal, que la logique puisse être considérée à la fois comme une analyse de certaines conditions de possibilité de la pensée vraie, du raisonnement correct, et de la connaissance en tant qu'elle s'exprime dans un langage, et d'autre part, comme une étude systématique du calcul, des algorithmes et des machines informatiques. Cette remarque relative à l'objet de la logique est souvent trivialisée, réduite à quelques idées devenues banales, et l'on estime qu'elle ne pose qu'une difficulté apparente. Très grossièrement, on sait en effet, et cela semble fournir une explication suffisante, que la logique s'intéresse aux *formes* déductives ; que la correction d'un raisonnement formalisé doit pouvoir être vérifiée par une sorte de calcul ; que la logique analyse, en conséquence, les possibilités et les limites du calcul en général, en introduisant notamment la notion de machine de Turing ; que les calculateurs universels et autres machines informatiques, enfin, constituent pour cette raison un nouveau domaine d'application de la logique. L'idée qu'il puisse être question de machine en logique n'est donc généralement pas considérée comme l'expression d'une

1. Cf. Robert Blanché et Jacques Dubucs, *La logique et son histoire*, Paris, Armand Colin, 1996 (éd. complétée du livre de Blanché, *La logique et son histoire d'Aristote à Russell*, 1970), p. 224.

difficulté qui, en elle-même, soit suffisamment digne d'intérêt. En quel sens pourrait-elle constituer un problème philosophique ?

4 / L'affadissement de cette idée procède de deux présupposés : premièrement, que si l'informatique rencontre la discipline logique, ce n'est pas parce qu'elles auraient l'une et l'autre des objets communs, mais parce que la première serait un domaine d'application bien particulier, voire marginal, de la seconde ; deuxièmement, que la notion de machine n'est pas centrale en logique, et que si elle joue un rôle important, c'est uniquement dans le secteur de la logique contemporaine que l'on appelle « théorie de la calculabilité ».

Lorsque l'on n'ignore pas purement et simplement l'importance que revêt la logique pour les systèmes de traitement de l'information, pour les machines abstraites et les calculateurs universels, on estime souvent qu'il s'agit là de nouveaux domaines d'*application* de la logique contemporaine, dont l'objet principal serait d'une tout autre nature. La logique constituerait un outil particulièrement utile pour l'étude des ordinateurs et des langages informatiques, au sens où l'on pense parfois que les mathématiques trouvent dans les phénomènes de la nature un domaine d'application privilégié, bien que les notions et les constructions mathématiques soient conçues de manière tout à fait indépendante de ces phénomènes. Si ce dernier point est lui-même douteux, l'image que l'on pourrait être tenté d'en tirer, par comparaison, pour comprendre les relations qu'entretiennent la logique et l'informatique doit être considérée comme tout à fait erronée. Il est vrai que si l'on considère les résultats et les techniques logiques qui ont été mentionnés ci-dessus parce qu'ils ont une importance fondamentale en informatique, on ne peut qu'être frappé par le fait qu'ils remontent presque tous à une époque (entre 1879 et les années trente) qui précède d'une dizaine d'années au moins la conception des premiers ordinateurs. On se gardera cependant d'en tirer des conclusions simplistes sur un prétendu rapport d'application d'une discipline à une autre, comme si les machines étaient une réalisation matérielle et concrète dont la logique (ou du moins certaines de ses parties) aurait analysé

et théorisé par avance les possibilités de calcul. En revanche, ce qui s'impose rapidement comme une évidence lorsque l'on examine les exemples qui ont été donnés, c'est la possibilité d'une double interprétation ou d'un double usage d'un grand nombre de notions, constructions et théorèmes de la logique : les langues formulaires sont tantôt des systèmes formels pour l'analyse et l'expression des propositions et des raisonnements, tantôt des langages de programmation dans lesquels sont représentés des algorithmes (le lambda-calcul est à cet égard un exemple tout à fait caractéristique) ; les modèles et les méthodes sémantiques servent à la fois à l'interprétation des énoncés et des systèmes formels de la logique et à celle des langages informatiques et des programmes ; des résultats aussi importants que le théorème d'incomplétude de Gödel, le théorème d'indécidabilité de Church, le théorème fondamental de Herbrand ou le *Hauptsatz* de Gentzen ont des conséquences et des interprétations à la fois en logique et en informatique ; les machines de Turing ont une définition purement mathématique, elles servent à résoudre des problèmes fondamentaux en logique, et sont également un modèle théorique des ordinateurs numériques. Ici encore, les exemples qui pourraient être ajoutés sont légion, mais le plus caractéristique est certainement la correspondance de Curry-Howard, qui permet d'identifier, sous certaines conditions, les programmes et les preuves, les types de données et les propositions, les calculs par réduction de termes et les normalisations de preuves. A ces rapprochements correspond l'absence de toute frontière nettement tracée entre les deux disciplines. Cela ne signifie évidemment pas qu'elles puissent être confondues, mais que les problèmes posés en informatique théorique sont souvent des problèmes de logique, et que l'inverse est également vrai. Ainsi, les questions de complexité concernent avant tout les calculs et les algorithmes : si certains problèmes sont insolubles[1], d'autres ont une solution algorith-

1. Au sens suivant : pour certains ensembles, il n'existe pas de procédure effective permettant de déterminer dans le cas général si un x quelconque est ou n'est pas élément de l'ensemble. Celui-ci est indécidable.

mique dont la complexité est rédhibitoire ; en théorie de la calculabilité, on cherche à mesurer cette complexité en lui donnant une définition précise, et à déterminer si elle peut être réduite. Or des questions similaires peuvent être posées au sujet des formules, des preuves et de leur normalisation[1]. Un autre cas, tout à fait caractéristique, est celui du système F, introduit par Girard en théorie de la démonstration, et découvert indépendamment par Reynolds en informatique[2].

Concernant les relations qu'entretiennent la logique et l'informatique, l'énumération de ces quelques exemples suggère fortement une remise en question des deux présupposés mentionnés précédemment. Premièrement, il s'agit de tout autre chose que d'un rapport d'*application* ; les deux disciplines ont clairement des objets et des problèmes communs, ainsi que des méthodes communes. Deuxièmement, les relations entre logique et informatique sont loin de se réduire à la seule théorie de la calculabilité ou aux machines de Turing ; elles concernent un grand nombre d'autres notions et parties fondamentales de la logique, et notamment le calcul des prédicats, les systèmes formels, les constructions sémantiques, la notion de démonstration et ses différentes formalisations. Or si l'on pouvait montrer avec précision en quel sens la logique en tant que telle a réellement pour objet le calcul, les algorithmes et les machines, aussi bien que les raisonnements, les formules valides et l'analyse du langage, alors, l'idée de machine en logique susciterait certainement l'attention du philosophe ; car elle ne manquerait pas de renouveler plusieurs questions classiques d'histoire ou de philosophie des sciences et d'en poser de nouvelles. Au sujet des machines, tout d'abord : qu'est-ce qu'une machine ? quels rapports y a-t-il entre la définition mathématique d'une machine logique et

1. Cf. Jean-Yves Girard, *Proof theory and logical complexity*, vol. 1, Naples, Bibliopolis, 1987, et, du même auteur, l'article Théorie de la démonstration, dans l'*Encyclopædia Universalis*, édition de 1995. Sur les rapports entre complexité logique et complexité algorithmique, cf. ci-dessous, chap. IX, p. 208-209.
2. Exemple cité dans Jean-Yves Girard, Yves Lafont, Paul Taylor, *Proofs and Types*, Cambridge, Cambridge UP, 1989, p. 82.

sa réalisation physique sous la forme d'un ordinateur ? quelle place la machine informatique occupe-t-elle dans l'histoire des machines en général ? Mais également au sujet de la logique elle-même : qu'est-ce que la logique, pour accorder un rôle essentiel à la notion de machine ? quelle lumière la logique contemporaine et ses relations avec l'informatique jettent-elles sur l'histoire de la logique dans son ensemble ? quels rapports y a-t-il entre l'ancienne logique, issue d'Aristote et de Chrysippe, la logique mathématisée postfrégéenne, et la logique qui sert de fondement aux sciences et techniques du traitement de l'information ?

5 / L'idée de machine en logique poserait aussi, et surtout, la question de l'intérêt que présente la logique pour le philosophe, et de l'usage qu'il fait de cette discipline. On connaît les principales raisons pour lesquelles le philosophe est aussi logicien (en dépit de l'opposition marquée par certains philosophes à l'égard de la logique, ou à l'égard de son application aux méthodes philosophiques). Si le philosophe est aussi logicien, c'est surtout, premièrement, parce que la logique porte sur la vérité, la connaissance et la science, ou qu'elle détermine une grammaire des choses du monde, et deuxièmement parce qu'elle rend possible une forme d'analyse du langage, considérée par certains comme la méthode philosophique par excellence. De ce point de vue, on voit évidemment mal quel intérêt pourrait présenter pour le philosophe un examen des relations entre logique et informatique, et l'on comprend que les philosophes-logiciens n'aient consacré qu'une faible part de leurs analyses à la notion de machine. Ceux qui ont fait porter leur réflexion sur les machines informatiques visaient plutôt des questions comme celles du mécanisme, de la pensée des machines, ou des modèles computationnels de l'esprit ; et dans ces contextes, ce n'est pas de la logique dans son ensemble qu'il s'agit, mais de certains résultats et de notions bien précises, comme les théorèmes d'incomplétude de Gödel, les machines de Turing ou les logiques spécifiques utilisées en intelligence artificielle. Pourtant, si l'on réussit à mettre en évidence la nature et la profondeur des relations qui

existent entre la logique et l'informatique, et à montrer, en conséquence, que la machine n'occupe pas en logique une place marginale, ce sont d'autres problèmes philosophiques qui devront être posés : comment une discipline dont l'intérêt philosophique principal est d'ordre épistémologique, gnoséologique ou ontologique en vient-elle à traiter des calculs, des algorithmes et des machines ? Et si l'on admet que l'idée de machine est devenue suffisamment importante en logique pour ne plus pouvoir être négligée par la philosophie (ce point reste à démontrer), le problème le plus délicat devient alors de savoir ce qu'il est légitime d'en conclure pour la connaissance, pour la logique et pour la notion de machine elle-même. Comment définir l'intérêt philosophique de cette idée sans verser dans une thèse généralisante et par trop contestable sur la réduction de la pensée ou des phénomènes de l'esprit à un calcul ? Cela suppose d'abord que l'on en mesure la portée. On pourra ensuite se demander quelles conséquences tirer du constat suivant, pour le moins étonnant au premier abord : la logique est à la fois une discipline du discours vrai, du raisonnement et de la connaissance, et une science des algorithmes et des machines informatiques.

La problématique qui vient d'être exposée se distingue assez nettement des questions habituellement traitées en philosophie de la logique. Apparemment étrangère aux problèmes majeurs de ce domaine (la vérité, la référence, le sens, l'existence, etc.), la machine en logique et en informatique n'a guère suscité la réflexion philosophique que d'un point de vue bien différent, à l'intérieur d'une problématique très particulière : celle de l'intelligence artificielle, des modèles computationnels du mental en psychologie cognitive, ou de la pensée des machines. Dans cette perspective, on ne retient de la logique que certains résultats et quelques outils (essentiellement les théorèmes d'incomplétude, d'indécidabilité, la thèse de Church et les machines de Turing), que l'on transfère hors de leur discipline d'origine vers la philosophie de l'esprit ou la psychologie, pour un usage auquel rien ne les prédispose *a priori*. Or malgré les conséquences parfois contra-

dictoires que les uns et les autres croient pouvoir tirer de résultats logiques comme les théorèmes d'incomplétude, on discute moins de la légitimité de ce transfert conceptuel, qui semble tenu d'emblée pour acquis, que des thèses qu'il est censé devoir servir. D'où les questions de philosophie de l'esprit inspirées par l'idée de machine et par quelques résultats logiques connus : que nous apprennent les théorèmes de Gödel sur les capacités de notre pensée ? une machine peut-elle penser ? qu'en est-il de la théorie fonctionnaliste de l'esprit ? quelles connaissances des phénomènes psychiques le modèle des machines informatiques nous permet-il d'acquérir ? quel est l'intérêt philosophique de l'intelligence artificielle ? Si l'importance de toutes ces questions ne fait aucun doute, elle ne doit pas nous faire oublier que les machines auxquelles il est fait référence sont des objets de la logique avant d'être d'hypothétiques outils pour la philosophie de l'esprit ; et qu'en ce sens, le problème de la machine en logique est premier, et plus fondamental, que celui du mécanisme en psychologie.

En définitive, ce qui motive notre effort pour replacer la machine dans son lieu d'origine, c'est la volonté de comprendre la logique dans sa forme présente, afin de déterminer quel peut être son rapport à la philosophie. La logique de Russell n'est ni celle d'Aristote ni celle de Leibniz. Les implications philosophiques s'en trouvent, d'un auteur à l'autre, profondément modifiées. Mais la logique n'est pas seulement celle d'un auteur ou d'une école de philosophie. Elle est aussi, surtout depuis un siècle environ, une discipline qui, fût-elle multiforme et variée dans ses développements, n'en demeure pas moins élaborée collectivement par une communauté scientifique. Comme l'écrit Jacques Dubucs,

> la logique a indiscutablement cessé d'être une partie de la philosophie, et elle a fini par constituer plutôt un secteur, aujourd'hui de moins en moins marginal, des mathématiques elles-mêmes[1].

Lorsqu'il traite de logique, s'il entend parler de cette discipline dans sa forme présente, le philosophe ne saurait ignorer cette évo-

1. Blanché et Dubucs, *op. cit.*, p. 355-356.

lution historique ni penser comme si ce secteur des mathématiques n'avait pas accordé, depuis Turing, une importance croissante à l'idée de machine. Il ne s'agit évidemment pas de soutenir que toutes les parties de la logique sont également concernées par ce rapport aux machines et à l'informatique, mais plutôt de mesurer le chemin parcouru dans cette direction depuis les années trente, afin de comprendre que notre logique, à cet égard, n'est plus celle de Frege ou de Russell. D'où l'interrogation qui anime le présent travail : si la logique connaît une vie nouvelle dans sa relation aux machines et à l'informatique, quelles en sont les conséquences pour le philosophe ? En quoi la question de la logique en philosophie se trouve-t-elle modifiée ou déplacée ?

I

LE PROGRAMME DE HILBERT

A L'ORIGINE D'UNE THÉORIE DE LA CALCULABILITÉ

Pour toute recherche qui porte sur les relations entre les machines informatiques et la logique, la théorie de la calculabilité se présente à première vue comme le domaine le plus significatif. D'une part, elle trouve son origine dans la confluence historique, en 1936, de méthodes et de solutions relatives à l'une des principales difficultés posées par la logique mathématique au début de ce siècle – le problème de la décision –, et d'autre part, elle définit, élabore et analyse un grand nombre des concepts fondamentaux pour l'étude théorique des machines informatiques. Elle traite précisément de ce qui est commun aux calculs logiques et aux calculateurs programmables que sont les ordinateurs. Pourtant, une simple référence à la notion polysémique de calcul ne suffit pas à convaincre qu'il s'agit, ici et là, du même sens de ce mot, et n'éclaire pas encore ce qui fait l'importance du rapprochement. Pour comprendre de quelle manière la logique traite de machines, il faut montrer comment les recherches logiques des années trente ont permis de réunir un faisceau de méthodes qui convergent, d'abord, vers la solution du problème de la décision, et qui s'épanouissent dans l'élaboration d'une théorie de la calculabilité, avant de trouver dans le développement de l'informatique théorique, chacune à sa manière, différentes interprétations en termes de

machines, d'algorithmes ou de programmes[1]. L'examen de ce parcours historique récent, par lequel la logique, dans les problèmes qu'elle pose, rejoint les fondements d'une théorie des machines informatiques, doit permettre de répondre plus précisément à la question suivante : quelle lumière la notion de calcul jette-t-elle sur la machine en tant qu'elle est un objet de la discipline logique ? Question préalable à une autre interrogation, renvoyant, cette fois, à une histoire plus ancienne : quelle place le calcul occupe-t-il en logique, et dans l'histoire de la logique ? La logique prend-elle nécessairement la forme d'un calcul ? Ce qui est en question, alors, c'est le sens et l'unité de la logique en tant que telle.

Les méthodes définitoires de la notion d'algorithme auxquelles on a fait allusion sont principalement les *fonctions récursives*, dont la définition, à l'origine, est due à Herbrand et Gödel, le *lambda-calcul* inventé par Church, puis étudié et modifié par Rosser et Kleene, et les *machines de Turing* ; à ces trois méthodes, qui sont sans doute les plus connues, il faut ajouter les systèmes canoniques de Post, ainsi que celles qui furent introduites ultérieurement, comme les algorithmes de Markov ou les systèmes formels de Smullyan[2]. Bien que les trois premières aient été conçues à peu près en même temps mais de manière indépendante, toutes ces méthodes se sont révélées être équivalentes en un sens qu'il conviendra de préciser, mais dont on peut donner une première idée en disant qu'elles définissent exactement la même classe de fonctions, celles que l'on appelle les « fonc-

1. Pour distinguer les programmes des algorithmes, disons qu'un *programme* est une procédure effective ou un algorithme écrit dans un langage formel déterminé. Alors que la littérature mathématique offre, depuis longtemps, de nombreux exemples d'algorithmes, on parle plutôt de « programmes » au sujet des procédures effectives écrites, en informatique, dans un langage de programmation. Dans la suite, il arrivera cependant que l'on utilise indifféremment l'un ou l'autre des deux termes.
2. Sur les origines de la théorie de la calculabilité, on pourra consulter Martin Davis, Why Gödel didn't have Church's thesis, *Information and control,* 54, 1982 ; Rolf Herken, *The Universal Turing Machine. A Half-Century Survey,* Oxford University Press, 1988, 2e éd., Vienne, Springer, 1994 ; Stephen C. Kleene, Origins of recursive function theory, *Ann. Hist. Computing,* 1981, 3-1 ; Jean Mosconi, *La constitution de la théorie des automates,* thèse d'État, Univ. de Paris 1, 1989 ; Barkley J. Rosser, Highlights of the history of the lambda-calculus, *Ann. Hist. Computing,* 1984, 6-4.

tions calculables ». Ce qui retiendra notre attention, dans l'examen qui suit, c'est moins la convergence historique de ces méthodes que les relations qu'elles contribuent à établir, sous différentes formes, entre la logique et les machines informatiques. On ne peut manquer, cependant, d'examiner le sens du problème de la décision ; d'abord parce que sa solution est le résultat principal vers lequel s'infléchissent les mouvements convergents dont il a été question, mais également parce que l'on verra qu'il peut, lui aussi, être interprété en termes de machines.

On reconnaît généralement dans le problème de la décision l'une des tâches assignées au logicien par le programme de Hilbert, et dans les solutions négatives données par Church ou par Turing les coups ultimes portés à ce programme, après ceux que les théorèmes de Gödel lui avaient déjà infligés en 1931. Ici encore, ces différentes péripéties nous concernent seulement par les conséquences qu'elles peuvent avoir sur le développement ultérieur de l'informatique, et sur la machine conçue comme un objet de la logique. Mais elles sont, à cet égard, tout à fait décisives. Des différentes étapes de cette histoire, émaillée de questions et de résultats théoriques, on ne rappellera donc, dans ce qui suit, que celles qui ont pu recevoir, ultérieurement, une interprétation en termes de machines informatiques (sans prétendre pour autant en faire une revue exhaustive) ; or cela exige quelques précisions préalables sur la manière dont ces résultats ont été compris, en premier lieu, dans leur signification purement logique. Apparaîtront ensuite les relations qui existent entre les origines logiques de la théorie de la calculabilité et ses conséquences fondatrices pour l'informatique théorique. On verra qu'à cet égard, la notion de procédure effective est l'une de celles qui établissent les rapports les plus étroits entre la logique et les machines, et qu'il s'agit bien du problème originel et central de la théorie de la calculabilité. Pourtant, la question de savoir comment caractériser formellement l'idée de procédure effective s'est présentée dans le contexte de la théorie de la démonstration telle qu'elle fut conçue par Hilbert, et à partir de quelques-unes des grandes questions de la logique mathématique du début de ce siècle.

LE PROGRAMME DE HILBERT

Le programme de Hilbert est indissociable d'une philosophie des mathématiques, autant que d'une conception bien particulière de ce qui devrait constituer leur fondement. En effet, ce qui dépend de l'interprétation hilbertienne des mathématiques, ce n'est pas seulement la solution qu'il espère apporter aux principales questions relatives à leurs fondements (celles de la consistance, de la complétude – aux sens sémantique et syntaxique – et de la décidabilité), mais également la signification même de ces questions, que l'on ne comprendra donc pas avant d'avoir saisi les principes qui devaient permettre, selon Hilbert, de les résoudre.

Dans une conférence prononcée en 1928, Hilbert soutient une idée sur laquelle reviennent plusieurs de ses écrits :

> En mathématiques, il n'y a pas d'*ignorabimus*, nous pouvons toujours trouver une réponse à une question pourvu qu'elle ait un sens[1].

A cet égard, il n'y a pas de différence essentielle, pour Hilbert, entre les mathématiques usuelles et les questions relatives à leur fondement : les mathématiques doivent pouvoir apporter une réponse mathématique à tous les problèmes qu'elles posent. En 1900, lors du Congrès de Paris, dans sa communication *Sur les problèmes futurs des mathématiques*, Hilbert posait notamment le suivant :

> Démontrer que les axiomes ne sont pas contradictoires ; c'est-à-dire démontrer qu'en se basant sur les axiomes l'on ne pourra jamais arriver à des résultats contradictoires au moyen d'un nombre fini de déductions logiques[2].

1. Probleme der Grundlegung der Mathematik, *Math. Annal.*, 102, 1929, trad. franç., Problèmes de fondation des mathématiques, *in* Jean Largeault, *Intuitionisme et théorie de la démonstration*, Paris, Vrin, 1992, p. 185.
2. David Hilbert, Sur les problèmes futurs des mathématiques : les 23 problèmes, *Comptes rendus du II^e Congrès international de mathématiques tenu à Paris du 6 au 12 août 1900*, Gauthier-Villars, 1902, réimpr., Sceaux, Gabay, 1990, p. 15.

Encore faut-il, pour parvenir à une telle démonstration, élaborer une philosophie des mathématiques qui la rende possible. Dans une perspective comparable à celles de Frege ou de Russell, par exemple, il semble bien que cette question de la contradiction des axiomes ne puisse pas faire l'objet d'une démonstration au sens mathématique du terme[1]. En effet, si l'on pose, comme le font Frege et Russell, que la logique est universelle, au sens où ce dont elle parle est « tout ce qui est », si tous les raisonnements strictement déductifs doivent pouvoir être logiquement formulés, et si l'on ne conçoit, par conséquent, aucun point de vue extérieur à la logique, alors, il semble difficile de déterminer les limites d'un système formel et d'éprouver sa consistance, si ce n'est en l'utilisant effectivement pour dériver des théorèmes, et en mettant à l'épreuve sa capacité démonstrative de manière quasi expérimentale. De ce point de vue, le fondement des mathématiques n'est donc pas assuré par une démonstration de consistance de forme mathématique. *A contrario,* dans une communication donnée lors d'un congrès à Heidelberg, Hilbert esquisse dès 1904 une preuve mathématique purement syntaxique de la cohérence de l'arithmétique[2]. S'il conçoit la possibilité d'une telle démonstration, c'est parce qu'il élabore une philosophie des mathématiques différente, qui devrait, selon lui, apporter une solution définitive au problème des fondements, par le double moyen d'une formalisation des mathématiques, et d'une théorie de la démonstration, autrement nommée « métamathématique ».

Pour Hilbert, la formalisation n'a pas l'ambition, qui était celle de Russell, de montrer qu'il est possible de ramener les concepts et

1. Ce point est souligné par Dreben et Van Heijenoort dans Gödel, *op. cit.*, p. 44-48, au sujet d'un autre problème, celui de la complétude. Sur la vérité et la non-contradiction des axiomes, lire la correspondance Hilbert-Frege dans François Rivenc et Philippe de Rouilhan, *Logique et fondements des mathématiques. Anthologie (1850-1914)*, Paris, Payot, 1992.
2. Über die Grundlagen der Logik und der Arithmetik, *Verhandlung des Dritten Internationalen Mathematiker-Kongresses in Heidelberg vom 8. bis 13. August 1904*, Leipzig, Teubner, 1905, trad. franç., Sur les fondements de la logique et de l'arithmétique, *in* Rivenc et de Rouilhan, *op. cit.*, p. 245-270.

le discours mathématiques à des notions et à une expression purement logiques :

> La liaison étroite entre vérités arithmétiques et vérités logiques, et leur inséparabilité, rendent nécessaire une construction simultanée de l'arithmétique et de la logique formelle[1].

Il s'agit plutôt de réduire les mathématiques usuelles telles qu'elles sont écrites à une combinaison de signes (aussi appelés « symboles ») qui puissent être considérés du point de vue de la forme, indépendamment de leur signification. Dans la construction d'un système formel, on distinguera une collection de *signes primitifs*, l'alphabet formel, une collection de *suites de signes* appelées « expressions bien formées » ou « formules », dont un sous-ensemble réunit les *axiomes*, et une collection de *suites de formules*, les démonstrations formelles.

> Tout ce qui jusqu'à présent rentrait dans la mathématique proprement dite sera dorénavant formalisé rigoureusement, en sorte que la *mathématique proprement dite* ou au sens strict se constitue en un stock de formules démontrables[2].

Le raisonnement mathématique s'applique alors à ces collections de formules et de démonstrations elles-mêmes. Le formalisme peut ainsi être considéré d'un point de vue extérieur et faire l'objet d'une analyse métamathématique, ce qui semble beaucoup plus difficile dans une perspective comme celle de Frege ou de Russell, pour qui la logique est un langage universel, sans extériorité mathématique possible[3]. Pour le formalisme hilbertien, « c'est la démonstration elle-même qui est quelque chose de concret et de tangible »[4] et qui peut faire l'objet d'une théorie. Ce sont les propriétés du sys-

1. Neubegründung der Mathematik. Erste Mitteilung, *Abhandl. aus dem Math. Seminar d. Hamb. Univ.*, Bd. 1, 1922, trad. franç., Nouvelle fondation des mathématiques. Première communication, *in* Largeault, *op. cit.*, p. 128.
2. *Ibid.*, p. 127.
3. Philippe de Rouilhan précise que si des logiciens universalistes comme Frege et Russell ont eu, de fait, des réticences à poser des problèmes métasystématiques, la perspective universaliste n'exclut pas, en droit, la possibilité de poser de tels problèmes de l'intérieur. Sur cette question, on pourra se reporter aux éclaircissements que donnent Philippe de Rouilhan, De l'universalité de la logique, *L'âge de la science*, 4, t. 1, Paris, Odile Jacob, 1991, p. 93-119, et François Rivenc, *Recherches sur l'universalisme logique. Russell et Carnap*, Paris, Payot et Rivages, 1993 (notamment dans l'Introduction).
4. Hilbert, *in* Largeault, *op. cit.*, p. 124.

tème formel qui peuvent être interrogées et mises à l'épreuve d'une analyse métamathématique. Parmi les différents problèmes qui sont soulevés de ce point de vue, on retiendra les quatre suivants :

La complétude sémantique

Un système formel au sens de Hilbert ne permet pas de dire qu'une formule est vraie ou fausse. Seule une distinction syntaxique est faite entre les formules démontrables, appelées « théorèmes » et celles qui ne sont la formule finale d'aucune démonstration. Les notions sémantiques supposent que l'on définisse ce qu'est une interprétation du langage et des formules, dans un domaine particulier quelconque. Et pour chacun des domaines d'interprétation, on peut alors demander si telle formule est satisfaite, ou si elle ne l'est pas ; dans le premier cas, on dit que l'interprétation considérée est un modèle de cette formule, et l'on appelle « formules (universellement) valides » celles qui sont satisfaites par toutes les interprétations du langage. Deux questions métamathématiques peuvent alors être posées :

(i) est-ce que tous les théorèmes sont des formules valides ? si tel est le cas, on dit que le système est correct, ou fiable ; se pose alors une seconde question :

(ii) est-ce que toutes les formules valides sont des théorèmes ? si tel est le cas, le système est sémantiquement complet[1].

La complétude syntaxique

Dans la conférence intitulée *Problèmes de fondation des mathématiques*, qui fut prononcée en 1928, Hilbert donne une définition de la

1. On définit, plus généralement, les notions de dérivation syntaxique (« $\Gamma \vdash A$ » se lit : « La formule A est syntaxiquement dérivable de l'ensemble de formules Γ ») et de conséquence sémantique (« $\Gamma \models A$ » se lit : « A est conséquence sémantique de Γ », ce qui signifie que tout modèle de Γ est modèle de A). Un système est sémantiquement complet si ces deux relations ont la même extension. Pour une définition rigoureuse de toutes ces notions, on peut se reporter à François Rivenc, *Introduction à la logique*, Paris, Payot, 1989, ou à René Cori et Daniel Lascar, *Logique mathématique*, t. 1 et t. 2, Paris, Masson, 1993. Le problème de la complétude sémantique pour la logique du premier ordre semble avoir été posé pour la première fois par Hilbert et Ackermann, *Grundzüge der theoretischen Logik*, Berlin, Springer, 1928. Sur l'histoire de ce problème, cf. Gödel, *op. cit.*, p. 44-59.

complétude syntaxique d'un système d'axiomes, dans le cas particulier de l'arithmétique :

> Si une formule qui appartient à l'arithmétique, mais non démontrable, est ajoutée aux axiomes de l'arithmétique, on peut dériver une contradiction de cette extension du système des axiomes[1].

Cette définition, appelée parfois « complétude au sens de Post » n'est pas la seule possible. On dit aussi, de manière équivalente, qu'un système formel est syntaxiquement complet si pour toute formule F le système permet de démontrer F ou de démontrer ¬ F (la négation de F) ; ou encore, si toute formule est soit démontrable, soit réfutable. On dit aussi que dans un système syntaxiquement complet, il n'y a aucune formule qui soit *formellement indécidable*.

La consistance

Un système formel est consistant s'il n'est pas contradictoire, c'est-à-dire s'il n'existe aucune formule F telle que F et sa négation soient l'une et l'autre des théorèmes. Et l'on sait que dans un système inconsistant, toutes les formules sont des théorèmes, ce qui réduit à néant l'intérêt qu'il peut y avoir à construire un tel système. On a vu que Hilbert posait, dès 1900, la question de la contradiction des axiomes.

La décidabilité

En 1928, dans les *Grundzüge der theoretischen Logik*, Hilbert et Ackermann définissent le problème de la décision pour la logique du premier ordre :

> Le problème de la décision [*Entscheidungsproblem*] est résolu si l'on connaît une procédure qui permette de déterminer, en utilisant un nombre fini d'opérations, la validité, respectivement la satisfaisabilité d'une expression logique donnée.

1. Problèmes de fondation des mathématiques, *in* Largeault, *op. cit.*, p. 183.

Plus généralement, un ensemble est décidable s'il existe une procédure effective permettant de déterminer, pour un élément quelconque d'un surensemble donné, s'il fait partie de cet ensemble. Ou encore, une propriété ou un prédicat $P(x_1, \ldots x_n)$ à n places est décidable s'il existe une procédure effective permettant de déterminer, pour un n-uplet quelconque $(a_1, \ldots, a_n)$ donné s'il satisfait ou ne satisfait pas la propriété ou le prédicat P. On aperçoit dès lors l'enjeu de la décidabilité pour la formalisation des mathématiques, puisque s'il était possible de construire un système formel pour lequel la propriété « la formule F est un théorème » fût décidable, il suffirait de mettre en œuvre une procédure effective pour déterminer s'il existe ou s'il n'existe pas de démonstration d'une formule donnée, F, quelconque.

Si, comme le pense Hilbert, il n'y a pas d'*ignorabimus* en mathématiques, il doit être possible, en principe, d'apporter une réponse par la théorie de la démonstration à chacune de ces quatre questions. Cependant, comme le remarque Gandy[1], ce postulat hilbertien n'exclut pas absolument la possibilité de solutions négatives, comparables à celle du problème de la trisection de l'angle à l'aide de la règle et du compas. En ce sens, une réponse négative apportée au problème de la décision ne devrait pas être comprise comme une mise en échec du programme de Hilbert. L'interprétation que suggère Gandy, textes à l'appui, est cependant différente : tout laisse à penser que Hilbert espérait apporter à l'*Entscheidungsproblem* une solution positive, même si, dans ses textes publiés, il ne présente jamais explicitement l'existence d'une telle solution comme un postulat de sa philosophie des mathématiques.

Il reste à comprendre quelle est la valeur de la stratégie métamathématique ; car celle-ci semblera vaine si l'on ne précise pas ce qui garantit la validité des raisonnements en théorie de la démonstration. Si l'on veut donner une preuve de consistance convain-

1. Robin Gandy, The confluence of ideas in 1936, *in* Herken, *op. cit.*, p. 58-59 (pour cet ouvrage, la pagination indiquée sera toujours celle de la seconde édition, chez Springer, 1994).

cante, il faut bien s'assurer qu'elle ne repose pas sur un système d'axiomes métamathématiques qui est lui-même inconsistant, ou qui suppose implicitement la consistance que l'on veut démontrer. A cet égard, l'entreprise de formalisation n'est-elle pas entraînée dans une régression à l'infini, ou prise au piège d'un cercle vicieux ? Hilbert échappe à cette difficulté en introduisant une distinction, essentielle pour sa philosophie des mathématiques, entre les méthodes concrètes et les méthodes abstraites, ou idéales, dont un exemple caractéristique est l'arithmétique des nombres infinis ou la théorie des ensembles de Cantor. Les mathématiques ont, depuis longtemps, utilisé l'abstraction comme un moyen d'élaborer de nouveaux concepts et de nouvelles théories, et Hilbert reconnaît que « seule l'introduction des propositions idéales fait qu'il peut y avoir une science mathématique »[1]. Mais à l'époque de Hilbert, l'histoire récente montrait aussi que des méthodes abstraites comme celles de Cantor permettent d'engendrer paradoxes et contradictions, et qu'elles ne constituent pas, pour cette raison, des voies aussi sûres que celle des mathématiques concrètes. Bien que les propositions concrètes des mathématiques n'aient jamais été formellement définies, ce qu'en dit Hilbert laisse penser que l'adjectif « concret » s'applique aux énoncés ou aux méthodes qui n'enveloppent pas la considération d'objets infinis conçus comme des touts achevés, qui ne traitent l'infini que sous la forme d'une potentialité, comme celle de la suite illimitée, non bornée, des entiers : 0, 1, 2... De telles méthodes relèvent de raisonnements qu'Hilbert appelle « finitaires »[2]. L'arithmétique élémentaire fournit des exemples typiques de cette catégorie, par opposition à « l'arithmétique et l'algèbre plus avancées, par exemple quand nous voulons obtenir des assertions sur une infinité de nombres ou de fonctions »[3]. L'im-

1. Die Grundlagen der Mathematik, conférence prononcée en juillet 1927, et publiée in *Abh. aus dem Math. Sem. d. Hamb. Univ.*, 6, 1928, trad. franç., Les fondements des mathématiques, *in* Largeault, *op. cit.*, p. 153.
2. Cf. par exemple la Nouvelle fondation des mathématiques. Première communication, *in* Largeault, *op. cit.*, et les remarques de Kleene, dans Gödel, *op. cit.*, p. 128 et 138.
3. Largeault, *op. cit.*, p. 119.

portance de cette distinction vient de ce que les raisonnements finitaires ne posent aucun problème de fondement, et sont suffisamment sûrs pour que l'on ne fasse peser sur eux aucun soupçon d'inconsistance ; à l'inverse

> une condition, une seule, mais indispensable, est toujours liée à l'emploi de la méthode des éléments idéaux : c'est qu'on prouve la consistance[1].

Le problème de la consistance se pose donc seulement pour les mathématiques idéales. Or les systèmes formels sont précisément conçus de telle sorte que la théorie de la démonstration, qui les prend pour objets, n'y trouve que des éléments finis : des formules, suites finies de signes finis, et des démonstrations, suites finies de formules. Par conséquent, la théorie de la démonstration traite d'objets comparables à ceux de l'arithmétique élémentaire :

> Les axiomes, formules et démonstrations, en quoi consiste l'édifice formel, sont exactement ce qu'étaient les chiffres dans la construction précédemment esquissée de l'arithmétique élémentaire[2].

Dès lors, la métamathématique devait pouvoir se restreindre aux raisonnements finitaires, jugés suffisamment sûrs ; et ce réquisit, explicitement énoncé, devait permettre au programme de Hilbert d'échapper à la fois au *regressus* et à la *petitio principii*.

La distinction entre méthodes élémentaires et méthodes abstraites permet aussi de comprendre un second aspect du programme, celui d'un idéal de *pureté des méthodes*. Il s'agit d'établir que la preuve d'un énoncé arithmétique élémentaire peut toujours faire l'économie des méthodes « idéales » ou « abstraites ». Ces dernières sont couramment utilisées en mathématiques parce qu'elles réduisent considérablement la longueur des preuves en autorisant l'application de résultats généraux déjà connus ou plus facilement démontrables à des cas particuliers. Hilbert veut établir un résultat de *conservativité*, prouvant que leur usage n'est jamais indispensable, et que la preuve d'une proposition « concrète » ne requiert aucun

1. Les fondements des mathématiques, *in* Largeault, *op. cit.*, p. 153.
2. Nouvelle fondation des mathématiques. Première communication, *in* Largeault, *op. cit.*, p. 119.

autre moyen démonstratif que ceux des mathématiques élémentaires. Plus précisément, ce deuxième aspect du programme de Hilbert demande que l'on montre comment réduire toute démonstration arithmétique qui utiliserait des méthodes abstraites à une démonstration qui ne reposerait que sur des moyens concrets, ou finitaires. Tel est le sens de l'idéal de *conservativité* posé par Hilbert.

La solution des problèmes soulevés par le programme de Hilbert

Hilbert espère résoudre définitivement le problème du fondement des mathématiques par la réalisation de son programme : construire un système formel (pour l'arithmétique, puis l'analyse et la théorie des ensembles), et développer, corrélativement, une théorie de la démonstration, point de vue extérieur au système, permettant notamment, par raisonnements finitaires, d'en prouver la consistance et de répondre aux questions de complétude, ainsi qu'à l'*Entscheidungsproblem*.

Ainsi conçu, le programme de Hilbert est plongé tout entier dans la problématique logique du fondement des mathématiques, et il semble éloigné de toute considération relative aux machines. Seuls les efforts des logiciens pour répondre à ce défi font apparaître la nécessité de caractériser la notion de procédure effective, et donnent ainsi l'occasion à Turing d'introduire pour la première fois, en 1936, une notion de machine en logique, pour apporter une solution négative à l'*Entscheidungsproblem*. Quelques années auparavant, les travaux de Gödel avaient permis d'apporter une réponse, également négative, aux problèmes de complétude syntaxique et de consistance. Or les logiciens énoncèrent ultérieurement une version généralisée de ces résultats négatifs, qui supposait elle aussi une élucidation de la notion de procédure effective et qui s'appuyait donc, comme on le verra, sur les machines de Turing. Enfin, rétrospecti-

vement, on s'aperçoit que la solution apportée par Gödel au problème de la complétude sémantique peut également être comprise en termes de machines. Au total, si le programme de Hilbert est avant tout conçu, initialement, dans la perspective du fondement des mathématiques, les quatre principaux théorèmes qui répondent aux questions qu'il pose reçoivent une interprétation relative à la notion de machine.

Replaçons maintenant ces quatre résultats logiques dans l'ordre historique de leur démonstration. En 1929, dans sa thèse de doctorat, réécrite en 1930 sous la forme d'un article, Gödel donne une preuve de ce que l'on appelle aujourd'hui le « théorème de complétude », montrant qu'il est possible de construire un système formel S sémantiquement complet pour la logique du premier ordre. Ce système définit une relation syntaxique de dérivabilité (que l'on note $\vdash$).

> Théorème de complétude. Pour tout ensemble Γ de formules et pour toute formule A, $\Gamma \vdash A$ si et seulement si $\Gamma \models A$.

D'après ce théorème, toutes les méthodes mathématiques que l'on pourrait utiliser pour démontrer une formule A dans une théorie Γ peuvent être réduites par un travail logique de formalisation à l'usage de quelques règles d'inférence très simples, celles du système formel S. De cette manière, la démonstration de tout théorème mathématique peut être formalisée, et sa validité peut être vérifiée par des moyens purement mécaniques, c'est-à-dire par une machine[1].

Après la démonstration du théorème de complétude sémantique, on pouvait espérer montrer qu'il existe un système d'axiomes pour l'arithmétique démontrablement consistant et complet au sens syntaxique. Or en 1931, Gödel fait usage du formalisme des *Principia Mathematica* de Whitehead et Russell, l'un des systèmes les

1. Selon des recherches récentes dues à Jean-Louis Krivine (Une preuve formelle et intuitionniste du théorème de complétude de la logique classique, *The Bulletin of Symbolic Logic*, vol. 2, 4, décembre 1996), l'interprétation du théorème de complétude en termes de machines peut être comprise en un sens beaucoup plus fort et plus profond à la lumière de la correspondance de Curry-Howard (dont il sera question au chapitre VIII).

plus riches, à l'époque, pour la formalisation des mathématiques, pour montrer qu'il existe des propositions formellement indécidables dans tout système semblable, pourvu qu'il permette de formaliser une partie élémentaire de l'arithmétique et qu'il soit consistant[1]. Tel est le sens de ce que l'on appelle souvent « le premier théorème d'incomplétude », qui montre que dans les systèmes apparentés à celui des *Principia Mathematica*, la formalisation des mathématiques ne peut être adéquate dès qu'elle enveloppe l'arithmétique élémentaire : pour chaque système d'axiomes répondant à cette condition, il existe des propositions non démontrables et dont la négation n'est pas non plus démontrable, pourvu que le système soit consistant. Dans le même article, Gödel réussit de surcroît à exprimer la propriété métamathématique de consistance dans le système formel, et à montrer que la formule exprimant cette propriété est elle-même formellement indécidable. Ce que l'on traduit souvent en disant qu'un système consistant permettant de formaliser une partie élémentaire de l'arithmétique permet d'exprimer, mais ne permet pas de prouver sa propre consistance.

Ces deux résultats négatifs sont généralement considérés comme une réfutation du programme de Hilbert, en dépit du fait que Gödel soutient explicitement le contraire[2]. Quoi qu'il en soit, demeurait encore en suspens, non résolue par l'article de 1931, la question cruciale de la décidabilité. Une solution positive du problème de la complétude syntaxique eût entraîné celle de l'*Entscheidungsproblem*. Mais l'inverse n'était pas vrai. L'une des principales difficultés revenait à donner une caractérisation formelle de la notion de procédure effective. Car pour s'assurer qu'il n'existe aucune telle procédure pour la solution du problème de la décision, il faut réussir à circonscrire mathématiquement la classe informellement définie de toutes les procédures effectives possibles. Or c'est en

1. Nous ne cherchons pas, ici, à distinguer ce que Gödel a réellement démontré dans l'article de 1931, de ce qui est dû aux améliorations ultérieures du résultat original. Sur cette question, cf. Gödel, *op. cit.*, p. 126-195 et 338-371.
2. On pourrait en effet imaginer, explique Gödel, une démonstration de consistance par des moyens finitaires non formalisables dans le système considéré, voir Gödel, *op. cit.*, p. 194.

ce point précis que se rejoignent les diverses méthodes dont la convergence a déjà été soulignée. Les machines de Turing, les termes du lambda-calcul, les systèmes canoniques de Post et les fonctions récursives permettent de caractériser des classes de procédures différemment définies mais équivalentes au sens où elles correspondent toutes au même ensemble de fonctions numériques, celui des fonctions dites « calculables ». Pour résoudre l'*Entscheidungsproblem*, Turing avance un ensemble d'arguments pour soutenir la thèse suivante : la notion de machine qu'il introduit en 1936 offre une caractérisation formelle adéquate de ce que l'on appelle couramment une « procédure effective ». De cette manière, le problème de la décision peut être ramené à une question susceptible de recevoir une solution logico-mathématique : existe-t-il une machine de Turing permettant de déterminer si une formule quelconque (d'un système formel pour la logique du premier ordre) est un théorème ? C'est à ce problème que Turing apporte une solution négative. Dans des travaux indépendants de ceux de Turing, d'autres logiciens montrèrent que le problème pouvait également être exprimé, de manière équivalente, en termes de fonctions récursives, ou encore par le formalisme du lambda-calcul. La solution négative de l'*Entscheidungsproblem* put ainsi être donnée de manière indépendante par des voies différentes. En examinant quelques-unes de ces voies, on s'apercevra que chacun des formalismes logiques qu'elles élaborent peut être compris en termes de machines, d'algorithmes et de programmes informatiques.

II

LES MACHINES DE TURING

Les machines de Turing sont introduites dans un article de logique mathématique daté de 1936, corrigé et complété en 1937, *Sur les nombres calculables, avec une application à l'Entscheidungsproblem*[1]. Elles peuvent être considérées comme l'outil principal et le nerf de la solution apportée par Turing au problème de la décision, mais leur importance logique et épistémologique est telle que l'on ne peut dire avec certitude si, dans l'esprit de l'auteur, cette solution était l'objet principal de l'article, ou une application particulière d'une analyse consacrée à la caractérisation des nombres calculables en termes de machines. On sait qu'en 1935, Turing, alors âgé de 23 ans, étudiant de King's College, à Cambridge, prit connaissance des travaux de Gödel et de l'*Entscheidungsproblem* en écoutant un cours de logique mathématique professé par M. H. A. Newman, et qu'en mai 1936, l'article sur les nombres calculables était achevé.

A cette date, les machines de Turing n'étaient qu'un outil mathématique destiné à une fin purement logique ; mais cinquante ans après la construction des premiers ordinateurs, elles sont aussi, pour nous, le schéma abstrait d'un mécanisme physiquement réalisable. Turing introduit pour la première fois en logique une notion abstraite de machine, en vue de résoudre un problème relatif au fon-

1. On Computable Numbers, with an application to the Entscheidungsproblem, *Proceedings of the London Mathematical Society*, series 2, vol. XLII, p. 230-265, reproduit dans Martin Davis, *The undecidable. Basic papers on undecidable propositions, unsolvable problems and computable functions*, Hewlett, NY, Raven Press, 1965 ; trad. franç. *in* Jean-Yves Girard et Alan Turing, *La machine de Turing*, Paris, Éditions du Seuil, 1995.

dement des mathématiques et, tout à la fois, donne par avance les éléments d'une théorie des machines informatiques. Comment un seul et même dispositif peut-il à la fois résoudre l'un des problèmes majeurs de la logique mathématique du début du siècle et servir de modèle théorique aux calculateurs universels que sont les ordinateurs ? L'intérêt de l'article vient de ce qu'il ouvre la voie à une double interprétation de questions et de résultats dont la portée, à l'origine, est uniquement logique, et qu'il rend ainsi possible, par avance, la mise en rapport d'une science des machines et d'une discipline du raisonnement. Encore faut-il comprendre les modalités précises de cette mise en rapport, afin de pouvoir en tirer des conséquences philosophiques suffisamment bien étayées. On examinera donc le dispositif des machines de Turing et son fonctionnement, afin de comprendre à la fois la solution qu'il permet d'apporter au problème logique de la décision, et la lumière qu'il jette sur les machines informatiques réellement existantes. Pour cela, on s'intéressera moins aux singularités du dispositif tel qu'il fut originellement conçu par Turing en 1936 qu'aux principaux traits caractéristiques des machines, qui montreront à la fois comment la solution du problème logique est possible, et quel est le rapport avec les ordinateurs[1].

UNE FORMALISATION DE LA NOTION DE CALCUL

Toute solution négative de l'*Entscheidungsproblem* requiert une caractérisation formelle de ce que l'on entend par « procédure effective ». Il faut en effet apporter une preuve métamathématique

1. Ainsi, par exemple, Turing aborde le problème de la décision à partir d'une analyse de la notion de « nombre calculable ». L'histoire ultérieure de la logique et de l'informatique a plus souvent retenu une approche en termes de « fonction calculable ». Sur ce point comme sur d'autres, l'analyse qu'on va lire n'est pas fidèle à la lettre de l'article original de Turing. Mais l'un des points importants est précisément la possibilité, due au caractère abstrait des machines, de multiples constructions, toutes équivalentes entre elles d'un certain point de vue.

qu'il n'existe aucune telle procédure capable de le résoudre. Or l'exemple type d'une procédure effective est celui d'un calcul arithmétique sur des nombres entiers : les conditions auxquelles l'opération est possible sont bien définies, chaque étape du calcul peut être achevée en un temps fini et détermine parfaitement l'étape suivante, en sorte qu'à chaque moment, on sait sans aucune ambiguïté si l'opération est achevée ou si, au contraire, une étape supplémentaire est requise. Pour des cas comme le produit ou l'addition, on sait même d'avance que le calcul pourra être achevé, ce qui n'est pas le cas général : par exemple, pour l'extraction d'une racine, le calcul des décimales n'est pas achevé, en général, après un nombre fini d'étapes. Tout calcul arithmétique usuel est donc l'application d'une procédure effective, bien que l'inverse ne soit pas vrai. Or, en s'appuyant sur le paradigme de l'opération arithmétique, Turing donne une représentation codifiée de ce qu'est un calcul dans le but de caractériser par une construction formelle la classe de tous les calculs possibles. Cette représentation prend la forme d'une machine, et la procédure effective celle d'une succession réglée d'opérations effectuées par cette machine sur les éléments d'une suite de signes. Encore faut-il en déterminer les modalités pour comprendre son fonctionnement : quelles opérations ? quelles règles ? quels signes ? Les réponses apportées à ces différentes questions ne sont pas arbitraires. Turing les tire d'une analyse minutieuse de ce que fait un calculateur humain, et montre qu'il n'y a rien, dans l'acte par lequel un calcul est ainsi effectué, qui ne puisse être également réalisé par un dispositif mécanique extrêmement fruste.

Dans le modèle proposé par Turing, tout calcul est effectué sur des signes. On se donne donc un alphabet de signes, fini puisque pour chaque procédure de calcul particulière, et donc pour chaque machine de Turing, les différents signes utilisés sont en nombre fini. Ces signes peuvent être écrits sur un ruban découpé en cases adjacentes ; ruban dont l'une au moins des extrémités est de longueur illimitée puisqu'il n'est pas possible, pour un calcul quelconque, de fixer par avance une borne à sa longueur. On convient que chaque

case du ruban comporte au plus un signe. Un dispositif appelé « tête de lecture-écriture-effacement » peut se déplacer d'une case à une case voisine pour y lire, écrire ou effacer un signe. La machine est un mécanisme « qui passe par bonds soudains d'un état parfaitement défini à un autre »[1], et les étapes d'un calcul sont donc bien distinctes les unes des autres. Pour comprendre comment la succession des opérations élémentaires est réglée, il faut remarquer qu'un calculateur humain ne se trouve pas à chaque instant du calcul dans la même disposition. Par exemple, s'il effectue l'addition de plusieurs nombres, il s'apprête tantôt à lire le chiffre suivant dans la colonne des unités, tantôt à reporter une retenue, tantôt à vérifier s'il reste encore une colonne de chiffres à additionner, etc. Turing distingue donc, corrélativement, différents « états » de la machine, qui correspondent, dans le cas paradigmatique de l'addition d'entiers, à ces différentes dispositions. Chaque état détermine entièrement le comportement de la machine à l'étape suivante, en fonction du signe lu sur le ruban[2]. Dans son analyse, Turing montre pourquoi l'on peut réduire à un nombre fini les différents états du calculateur humain, du moins en tant qu'il est occupé à effectuer un calcul (la variabilité de ses états affectifs, par exemple, est hors de propos). Par conséquent, l'alphabet et le nombre d'états étant l'un et l'autre finis, un calcul, conçu comme une succession d'opérations élémentaires, est déterminé par une règle qui associe à chaque état de la machine, et pour chaque signe lu, un comportement parfaitement défini du triple point de vue suivant :

(i) le signe lu peut être laissé intact, effacé, ou remplacé par un autre signe ;
(ii) la tête de lecture peut se déplacer d'une case, à droite ou à gauche, ou rester en place ;
(iii) la machine peut changer d'état, ou demeurer dans le même.

1. Turing, *op. cit., in* Anderson, *op. cit.*, p. 46 (trad. franç.). Turing écrit que ce sont des « machines à états discrets » *(discrete state machines), ibid.*
2. Turing appelle « configuration de la machine » la donnée du signe lu et de l'état dans lequel se trouve la machine.

Cette règle prend donc la forme d'un tableau de commande, différent pour chacune des machines. A un instant donné, si s_i est le signe lu, et q_i l'état de la machine, le tableau associe à la configuration (q_i, s_i) un triplet (s_j, M, q_j), où s_j est le signe qui remplace s_i, M représente le mouvement de la tête (aller sur la case de droite, sur celle de gauche ou rester immobile), et q_j l'état dans lequel passe la machine lorsqu'elle se trouve dans la configuration (q_i, s_i). Le tableau de commande n'est donc rien d'autre qu'une liste finie de quintuplets de la forme (q_i, s_i, s_j, M, q_j), où les cas $q_i = q_j$ et $s_i = s_j$ sont possibles. Pour chaque état q_i, le tableau détermine aussi le comportement de la machine dans le cas où la case observée sur le ruban ne comporte aucun signe.

On peut maintenant comprendre le mode de fonctionnement d'une machine de Turing. Avant le début du calcul, on inscrit sur le ruban une suite de signes, sur lesquels la machine effectuera des opérations. Par exemple, pour une machine qui effectue l'addition, on inscrit sur le ruban les nombres que l'on veut additionner, en utilisant les symboles d'un alphabet prédéfini, celui de la machine. On peut convenir, par exemple, que la tête de lecture est placée, au départ, sur le symbole le plus à gauche parmi ceux qui sont inscrits sur le ruban (le cas où aucun signe n'est inscrit sur le ruban n'étant pas exclu)[1]. La machine peut alors effectuer une suite d'opérations élémentaires, conformément au tableau de commande. L'un des états de la machine est distingué comme étant l'état initial, celui de la première configuration. On convient aussi que l'un des états q_i de la machine, appelé « état final », est tel que la machine n'effectue plus aucune opération lorsqu'elle y entre. Dans ce cas, on dit que le calcul est achevé, et l'on appelle « résultat du calcul » ce qui reste inscrit sur le ruban au moment où la machine est dans l'état final. Rien ne garantit cependant qu'une machine de Turing quelconque par-

1. Dans ce qui précède, il n'a pas été précisé si l'on choisissait de construire des machines de Turing dont le ruban est de longueur illimitée à l'une de ses extrémités seulement ou au deux. Le choix entre ces deux possibilités n'engage que des différences techniques et inessentielles. Dans le cas où les deux extrémités sont illimitées, et où aucun signe n'est inscrit sur le ruban, la tête se trouvera placée sur une case quelconque au début du calcul.

vienne à l'état final après un nombre fini d'étapes de calcul. Si l'état final n'est jamais atteint, le calcul ne produit aucun résultat. Ce qu'une machine détermine parfaitement, ce n'est donc pas le résultat d'un calcul, mais la procédure effective par laquelle il est recherché[1].

On voit que le fonctionnement général d'une machine de Turing étant compris, chaque machine particulière est parfaitement définie par son tableau de commande, c'est-à-dire par une liste de quintuplets. On voit aussi que pour chaque machine de Turing, le résultat d'un calcul est une fonction de ce qui est inscrit au départ sur le ruban, bien que cette fonction ne soit pas nécessairement totale (puisqu'il peut n'exister aucun résultat). En particulier, les machines de Turing peuvent être utilisées pour le calcul de fonctions numériques comme la somme ou le produit. Dans ces deux exemples, si les données initiales sont deux entiers codés dans un langage particulier comme l'alphabet binaire, le résultat du calcul est l'expression de leur somme ou de leur produit. Chaque machine définit donc, par son tableau de commande, une fonction des données initialement inscrites sur le ruban. A l'inverse, étant donné une fonction quelconque, on pourra demander s'il existe une machine de Turing capable de la calculer.

Comment exprimer formellement que la machine de Turing M est une procédure de calcul pour la fonction numérique partielle F à k places[2] ? Soit A, l'alphabet de M, i un codage des entiers dans l'ensemble A* des mots finis sur l'alphabet A[3], et j un codage de

1. On appellera « machine de Turing » la donnée d'un ensemble fini de signes (l'alphabet), d'un ensemble fini d'états comprenant un état initial et un état final, d'un tableau de commande, et du dispositif qui indique comment ces données doivent être utilisées. En ce sens, la procédure effective déterminée par une machine de Turing peut être appliquée à n'importe quelle suite de signes inscrite avant le début du calcul sur le ruban. Mais il arrive parfois que ce que l'on appelle « machine de Turing » comprenne non seulement l'alphabet, les états, le tableau de commande et le dispositif de fonctionnement, mais aussi la donnée initiale inscrite sur le ruban.
2. Si la fonction numérique F associe un entier à tout k-uplet d'entiers ($k = 1, 2, 3...$), on dit que F est totale. Cependant, on considère plus généralement les fonctions dites « partielles », qui ne sont pas nécessairement définies pour chaque k-uplet. En ce sens, les fonctions totales ne sont rien d'autre que des fonctions partielles particulières.
3. A* est l'ensemble de toutes les suites finies de symboles de A (éventuellement répétés).

l'ensemble des k-uplets d'entiers dans A*[1]. Soit u un k-uplet d'entiers quelconque, et $j(u)$ son code. On note $M(j(u))$ le résultat de la procédure M appliquée à $j(u)$ si ce résultat existe (ce que l'on note $M(j(u))\downarrow$), et l'on écrit $M(j(u))\uparrow$ si la procédure M appliquée à $j(u)$ ne conduit à aucun résultat, ou si le résultat $M(j(u))$ n'est le code d'aucun entier par le codage i. On dit alors que M est une procédure de calcul pour F si et seulement si :

(i) si $F(u)$ est défini, alors $M(j(u))\downarrow$ et $M(j(u)) = i(F(u))$;
(ii) si $F(u)$ n'est pas défini, alors $M(j(u))\uparrow$.

On dit aussi qu'une fonction numérique est Turing-calculable, s'il existe une machine de Turing qui soit une procédure de calcul pour cette fonction. Et plus généralement, une fonction symbolique (c'est-à-dire de A* dans A*) est Turing-calculable s'il existe une machine de Turing qui soit une procédure de calcul pour cette fonction.

Il est suffisamment clair que chaque machine ainsi définie est une procédure effective de calcul, et qu'en ce sens, Turing définit un calcul symbolique généralisé. « Symbolique », parce qu'il n'est pas seulement numérique, mais porte sur des suites de signes quelconques. « Généralisé » parce que l'exercice apprend rapidement que toutes les procédures de calcul usuelles, ainsi que d'autres, peuvent être effectuées par une machine de Turing. Mais la solution de l'*Entscheidungsproblem* suppose que cette généralisation soit comprise en un sens plus fort, à savoir que toute procédure effective, quelle qu'elle soit, puisse être effectuée par une machine de Turing. Ou encore, en termes de fonctions, que toute fonction pour laquelle il existe une procédure effective de calcul soit également Turing-calculable[2].

1. Un codage des entiers dans A* est une injection de N dans A* qui vérifie les deux conditions suivantes :
(i) il existe une procédure effective permettant de calculer le code de n'importe quel entier en un temps fini ;
(ii) il existe une procédure effective qui permet de déterminer, pour tout élément de A*, s'il s'agit du code d'un entier, et, le cas échéant, de déterminer quel est cet entier.

2. Pour des raisons de cardinalité, on ne peut pas espérer que toute fonction symbolique, ni même toute fonction numérique définie en termes ensemblistes puisse être calculée par une machine de Turing. Il existe en effet un nombre indénombrable de fonctions numériques selon la théorie des ensembles, et seulement un nombre dénombrable de machines de Turing puisque chaque machine est entièrement caractérisée par un tableau de commande fini.

Une fonction est dite « calculable » s'il existe une procédure, déterminée par des moyens finis, permettant de calculer sa valeur en chacun des points de son domaine (si la fonction n'est pas totale, le calcul ne produit aucun résultat pour certains de ces points)[1]. Ce qui reste informel, dans une telle définition, c'est la nature des « moyens finis » susceptibles d'être utilisés, et c'est précisément ce qui est rendu explicite et formellement exprimable par les machines de Turing : un alphabet fini, un nombre fini d'états, un tableau de commande déterministe (au sens où le comportement de la machine est parfaitement déterminé pour chacune de ses configurations), un fonctionnement divisé en étapes bien distinctes, et à chaque étape, une possibilité de mouvement limitée à un déplacement élémentaire pour la tête de lecture-écriture-effacement. La thèse avancée par Turing en 1936, est que ces moyens extrêmement restreints suffisent à couvrir toutes les procédures effectives possibles, et donc qu'ils suffisent pour définir toutes les fonctions calculables. Elle pose ainsi une généralisation extrêmement forte de la notion de calcul.

Comme le souligne l'article de 1936, la thèse de Turing[2] n'est pas susceptible d'être mathématiquement démontrée, puisqu'elle revient à caractériser une notion informelle, celle de procédure effective, par une construction formelle, celle des machines de Turing. Plusieurs arguments peuvent cependant être avancés en faveur de cette thèse, dont les principaux sont les suivants :

(i) *Argument inductif.* On ne connaît aucun exemple de fonction calculable au sens informel qui ne puisse être calculée par une machine de Turing, et cela malgré tous les efforts qui ont été déployés pour tenter de mettre la thèse en échec. Toutes les fonc-

1. L'usage de l'expression « fonction calculable » n'est pas parfaitement bien fixé. Pour certains auteurs, les fonctions calculables sont des fonctions totales ; pour d'autres, il s'agit de fonctions partielles (au sens où ce terme a été défini dans la note 2, p. 36).
2. L'expression « thèse de Church » est plus fréquemment usitée, et désigne la caractérisation des procédures effectives ou des fonctions calculables à partir de la notion de fonctions récursives. Dans la suite, on parlera tantôt de la « thèse de Turing », tantôt de la « thèse de Church », tantôt de la « thèse de Church-Turing ». Chacune des trois expressions fait référence à un énoncé qui pose une caractérisation de l'effectivité.

tions que l'on reconnaît aujourd'hui comme étant calculables sont aussi Turing-calculables.

(ii) *Argument de diagonalisation.* On utilise fréquemment en logique une méthode dite de « diagonalisation », classique depuis Cantor, permettant de prouver que telle ou telle énumération des éléments d'un ensemble infini dénombrable est incomplète. Par cette méthode, on se sert de l'énumération supposée complète pour construire un élément qui n'en fait pas partie. Or cet argument ne s'applique justement pas au cas d'une énumération des fonctions calculables par une machine de Turing.

(iii) *Argument de convergence.* Comme cela a déjà été souligné, plusieurs caractérisations de l'ensemble des fonctions calculables ont été proposées indépendamment les unes des autres par différents logiciens dans les années trente. Or on a pu prouver que les fonctions définissables par une machine de Turing, par un terme du lambda-calcul, par une fonction récursive ou par un système canonique de Post correspondent toutes à un même ensemble de fonctions, et qu'elles sont en ce sens équivalentes. Au sujet de l'une de ces définitions, celle des fonctions récursives, Gödel fait la remarque suivante, qui s'applique aussi bien aux machines de Turing :

> ... avec ce concept, on a réussi pour la première fois à donner une définition absolue d'une notion épistémologique intéressante, *i.e.* qui ne dépende pas du formalisme choisi[1].

(iv) *Argument par appel à l'intuition.* Dans l'article de 1936, Turing propose une analyse minutieuse de l'idée intuitive que nous pouvons avoir des opérations effectuées par un calculateur humain, distinguant soigneusement ce qui est essentiel au calcul et ce qui ne l'est pas. Il justifie ainsi, l'une après l'autre, chacune des limitations qu'il impose à ses machines. La réduction d'un calcul quelconque à la suite des opérations élémentaires qu'effectuent les machines de Turing est donc loin d'être arbitraire : elle abstrait les éléments essentiels d'un exemple paradigmatique.

1. Kurt Gödel, *Remarks before the Princeton bicentennial conference on problems in mathematics*, 1946, reproduit *in* Davis, *op. cit.*, p. 85.

Bien qu'ils ne puissent fournir aucune certitude d'ordre mathématique, ces arguments sont généralement jugés suffisants pour étayer cette thèse surprenante à première vue : qu'un dispositif mécanique aussi rudimentaire que celui des machines de Turing soit suffisant pour la mise en œuvre de n'importe quelle procédure effective. Pourtant, si radicale que puisse paraître cette réduction, elle ne constitue qu'une première étape puisque Turing montre, dans le même article de 1936, qu'il existe une machine bien particulière, appelée « machine universelle », capable d'effectuer à elle seule un travail équivalent à celui de n'importe quelle autre de ses machines. Cette seconde étape de la réduction ne prend pas, comme la première, la forme de l'énoncé d'une thèse. Il s'agit d'une proposition mathématiquement démontrable d'où il résulte, par la thèse précédemment avancée, qu'une seule et même machine de Turing, pourvu qu'elle soit convenablement choisie, est capable de calculer n'importe quelle fonction calculable. Pour que cette machine bien particulière fonctionne comme une machine universelle, une condition doit pourtant être remplie : que l'on inscrive sur son ruban non seulement les données initiales sur lesquelles le calcul doit être effectué, mais également une représentation de la procédure effective qui doit être appliquée à ces données. Pour prouver l'existence d'une machine universelle, Turing montre donc d'abord qu'une suite finie de signes suffit pour le codage de n'importe quelle machine. Lorsque l'on sait qu'une machine est entièrement caractérisée par un tableau de quintuplets fini, la possibilité de ce codage n'est guère difficile à concevoir (bien que sa réalisation effective soit complexe). Turing appelle « représentation standard » de la machine M le code de M. Encore faut-il montrer que l'on peut effectivement construire une machine universelle, capable de décoder n'importe quelle représentation standard inscrite sur son ruban, en appliquant la procédure qu'elle exprime aux données initiales également inscrites sur le même ruban. Les principes d'une telle machine universelle sont donnés par Turing au § 7 de l'article de 1936.

UNE RÉSOLUTION DU PROBLÈME DE LA DÉCISION

Dans l'analyse qui précède, les machines de Turing se présentent surtout comme le schéma abstrait de dispositifs physiquement réalisables, capables d'effectuer de multiples calculs numériques ou symboliques. De ce point de vue, elles peuvent être comparées aux machines à calculer réellement existantes, dont elles représentent un modèle. Mais on risque alors d'oublier qu'elles n'appartiennent pas du tout à la tradition des mécanismes de calcul, et que leur acte de naissance est un article dans lequel Turing résout un problème de logique. Pour comprendre ce qu'est une machine de Turing, il ne suffit donc pas d'en analyser le fonctionnement, comme on pourrait comparer les mécanismes respectifs des machines à calculer de Pascal et de Leibniz. Au reste, même lorsque Turing fait appel à une description imagée en termes de ruban, de cases, de tête de lecture, il ne s'intéresse pas du tout au mécanisme physique qui pourrait faire fonctionner le dispositif. L'intérêt des machines de Turing ne réside pas dans l'ingéniosité d'un mécanisme mais dans leur fonction et leurs conséquences logiques. Il est vrai qu'elles apparaissent aujourd'hui comme l'un des principaux modèles théoriques des machines réelles que sont les ordinateurs ; mais avant d'examiner ce point, il convient de replacer les machines de Turing dans une perspective proprement logique, où elles servent avant tout à résoudre le problème de la décision. En outre, en esquissant, dans ses grandes lignes, la solution de l'*Entscheidungsproblem* due à Turing, on apercevra plus précisément comment une notion abstraite de machine vient s'insérer dans la démonstration d'un théorème de logique.

De même que les problèmes métamathématiques posés par Hilbert portent sur les limites et les propriétés des formalismes logiques, de même Turing interroge d'un point de vue métathéorique les capacités de ses machines. A cet égard, l'idée de les coder par une représentation standard joue un rôle décisif, car elle permet de

démontrer rigoureusement certaines de leurs propriétés ou certaines de leurs limites. On voudrait savoir, par exemple, s'il existe un moyen effectif de résoudre le problème de l'arrêt, c'est-à-dire un moyen effectif de déterminer pour une machine de Turing quelconque M, et une suite de signes quelconque S inscrite sur son ruban, si M entrera dans l'état final après un nombre fini d'étapes de calcul. Par la thèse de Turing, la question revient à déterminer s'il existe une machine M_a qui, lorsqu'on inscrit sur son ruban la représentation standard d'une machine M et la suite de signes S, s'arrête après un temps fini, ayant inscrit sur son ruban, par exemple 0 si M ne s'arrête pas pour la donnée initiale S, et 1 dans le cas contraire. Dans la seconde formulation, on ne s'interroge plus sur l'existence d'un moyen effectif, mais sur celle d'une machine M_a. On se donne ainsi la possibilité d'une réponse négative, en démontrant formellement que l'existence d'une telle machine M_a conduirait à une contradiction. On voit aussi que cette seconde formulation suppose la possibilité d'une représentation standard des machines[1].

Mais comment prouver que l'existence d'une machine M_a serait contradictoire ? Supposons que l'on puisse construire M_a. M_a est une machine qui résout le problème de l'arrêt. Il n'est pas difficile de modifier M_a pour obtenir une machine M'_a résolvant le problème suivant : « Est-ce que la machine M s'arrête lorsque l'on inscrit sur le ruban de M sa propre représentation standard ? ». M'_a vérifie les propriétés suivantes :

(i) si $M(M)\uparrow$, alors $M'_a(M)$ s'arrête après avoir inscrit 0 sur son ruban.

(ii) si $M(M)\downarrow$, alors $M'_a(M)$ s'arrête après avoir inscrit 1 sur son ruban.

1. Dans ce qui suit, la notation $M(S)\uparrow$ (respectivement : $M(S)\downarrow$) signifie que la machine M n'entre jamais dans l'état final (respectivement : s'arrête après un nombre fini d'étapes de calcul) lorsqu'on inscrit S sur son ruban. M(S) représente le résultat du calcul, lorsqu'il existe, et ne représente rien dans le cas contraire. Dans le cas particulier où S n'est rien d'autre que la représentation standard d'une machine de Turing M_1, on utilise les notations $M(M_1)$, $M(M_1)\uparrow$ et $M(M_1)\downarrow$, identifiant ainsi abusivement la machine M_1 et sa représentation standard.

Par une modification de M'_a, on obtient alors aisément une nouvelle machine M''_a qui diffère de M'_a sur le seul point suivant : dans le cas où $M(M)\downarrow$, au lieu d'écrire 1 et de s'arrêter comme M'_a, la machine M''_a entre dans un état qui ne la conduit jamais à l'état final. M''_a vérifie donc les propriétés suivantes :

(i) si $M(M)\uparrow$, alors $M''_a(M)\downarrow$ (après avoir inscrit 0 sur son ruban).

(ii) si $M(M)\downarrow$, alors $M''_a(M)\uparrow$.

On voit alors que la contradiction surgit dès que l'on considère le cas particulier où M n'est rien d'autre que M''_a. D'où il appert qu'aucune machine de Turing ne peut résoudre le problème de l'arrêt ; par la thèse de Turing, ce problème est donc insoluble (on dit aussi qu'il reçoit une solution négative).

Pour résoudre l'*Entscheidungsproblem*, il suffit alors de montrer que sa solution permettrait aussi de résoudre le problème de l'arrêt. Ce qui suppose la possibilité d'une mise en relation des machines de Turing et des systèmes formels de la logique, puisque le problème de la décision porte sur ces systèmes. Plus précisément, cela suppose que l'on puisse décrire le fonctionnement des machines par une formule logique. Or Turing montre qu'une telle description est effectivement possible, pourvu que l'on introduise dans le langage logique les prédicats adéquats. Par exemple, pour chaque symbole s_i de l'alphabet d'une machine M, on se donne un prédicat binaire $R_{s,i}$ et l'on interprète $R_{s,i}(x, y)$ par la fonction propositionnelle suivante : « Dans la configuration x de la machine M, le symbole inscrit sur la case y du ruban est s_i » ; ou encore, un prédicat binaire I, tel que l'interprétation de $I(x, y)$ soit : « Dans la configuration x de la machine M, c'est la case y qui est observée par la tête de lecture. » Or, en introduisant un nombre suffisant de prédicats semblables, dont chacun permet de caractériser un des aspects de la machine, on pourra représenter par des formules logiques à la fois son tableau de commande et son fonctionnement, c'est-à-dire la succession des descriptions complètes de la machine à chaque étape du calcul (état dans lequel elle se trouve, numéro de la case obser-

vée par la tête de lecture, signe lu, état du ruban à droite et à gauche de la tête de lecture) ; et par conséquent également la proposition : « Il existe une étape du calcul où la machine M entre dans l'état final », par une formule existentielle, $\exists x A_M[x]$, où $A_M[x]$ est une formule qui dépend de la machine M, et dans laquelle x est une variable libre. Or en construisant une formule logique adéquate, Turing montre que si l'*Entscheidungsproblem* admet une solution, il existe une procédure effective permettant de déterminer, pour toute machine M, si la formule $\exists x A_M(x)$ est un théorème, et donc de résoudre le problème de l'arrêt. Mais puisque le problème de l'arrêt est insoluble, l'*Entscheidungsproblem* n'admet pas de solution.

Dans cette esquisse de la preuve apparaît l'un des points nodaux de la théorie de la calculabilité, à savoir la possibilité de faire correspondre au calcul d'un résultat par une machine la démonstration d'une formule dans un système logique. Ce qui peut être exprimé plus formellement et plus précisément de la manière suivante. Posons que la notation $M(S) = R$ signifie : la machine M entre dans l'état final après un nombre fini d'étapes de calcul lorsque S est inscrite sur son ruban au début du calcul, et R est le résultat de ce calcul. Cela peut être exprimé par une certaine formule d'un système logique, $F_{M, S, R}$, qu'il est possible de construire de manière effective. Alors, la correspondance entre le calcul effectué par une machine d'une part, et la démonstration d'une formule logique d'autre part signifie que l'on a $M(S) = R$ si et seulement si $F_{M, S, R}$ est un théorème. Calculer le résultat de la procédure définie par M lorsqu'elle est appliquée à S revient à chercher la démonstration d'une formule $F_{M, S, R}$ pour un certain R. Où l'on aperçoit, corrélativement, une correspondance entre l'insolubilité de l'*Entscheidungsproblem*, et l'insolubilité du problème de l'arrêt, c'est-à-dire entre la solution négative d'un problème de logique, et celle d'une question posée au sujet des machines.

Les machines de Turing et les systèmes formels hilbertiens sont les uns et les autres des formalismes permettant la combinaison réglée de suites de signes. Ils se distinguent en ce que la machine

abstraite permet d'effectuer un calcul, c'est-à-dire d'appliquer une procédure effective à une suite de signes, alors qu'un système formel est conçu pour la démonstration de théorèmes. La formalisation rend cependant cette différence de fonction purement intentionnelle dans la mesure où une correspondance précise peut être établie entre les calculs et les démonstrations. En effet, à la recherche d'un résultat par une machine de Turing, on peut associer la recherche d'une démonstration dans un système formel, et à la suite des étapes de calcul qui produisent un tel résultat, la preuve d'un théorème[1]. A l'inverse, un système formel étant donné, il est possible de construire une machine de Turing capable d'énumérer les preuves de ce système, et par conséquent aussi les propositions démontrables. Ce qu'une machine de Turing ne peut pas faire, c'est déterminer de manière effective si telle proposition quelconque est un théorème, de même qu'elle ne sait pas prévoir si un calcul quelconque conduira au bout d'un temps fini à un résultat. Ainsi se construit une correspondance entre le calcul et la démonstration, dévoilant un premier aspect des relations qui existent entre les machines et la logique.

Cette correspondance ne montre pas encore, cependant, la possibilité d'une mise en rapport des problèmes de logique avec les questions informatiques relatives aux machines réellement existantes. Les machines de Turing sont des machines abstraites, définissables par des moyens logico-mathématiques. Si le dispositif décrit par Turing est certes physiquement réalisable, on devine que sa mise en œuvre sous la forme d'un mécanisme réel ne produirait qu'un outil inefficace, sans usage pratique, et qu'elle n'aurait donc qu'un intérêt très limité. Les machines de Turing appartiennent avant tout à la logique, et ne s'inscrivent pas dans la tradition séculaire des machines à calculer. Qu'une notion abstraite de machine ait un rôle cardinal dans la logique contemporaine ne montre donc

1. Les systèmes formels dont il est question ici satisfont la condition minimale suivante : il existe une procédure effective permettant de déterminer, pour toute suite finie de formules qui est une démonstration, qu'il s'agit bien d'une démonstration.

pas encore quelle est la nature de ses relations avec les calculateurs et ordinateurs réellement existants, ni comment les problèmes de la logique rejoignent ceux de l'informatique théorique. Pour apercevoir cela, il faut comprendre pourquoi les machines de Turing sont aussi un modèle théorique des machines réelles.

III

MACHINES ABSTRAITES, CALCULATEURS ET PROGRAMMES

UNE PERSPECTIVE LOGIQUE SUR L'HISTOIRE DES CALCULATEURS

Une machine à calculer est un dispositif physique qui produit la représentation d'un nombre, appelé résultat du calcul, après qu'on lui a communiqué le ou les nombres sur lesquels le calcul devait être effectué. On distingue les machines analogiques des machines numériques. Pour la représentation des nombres, les premières utilisent des grandeurs physiques continues comme la valeur d'un écart angulaire, une quantité de matière fluide, ou l'intensité d'un courant électrique. Leur usage est aujourd'hui moins répandu et seules les secondes retiendront notre attention, qui utilisent des états physiques discrets comme des cartes perforées, des ergots disposés sur un cylindre, ou la différence de potentiel électrique d'un composant conçu pour n'admettre qu'un nombre limité de valeurs stables (deux valeurs stables seulement pour les machines qui utilisent la représentation binaire). Les machines à calculer se distinguent les unes des autres par leurs caractéristiques techniques et leurs performances : la taille des nombres dont elles permettent la représentation, le type d'opérations qu'elles sont capables d'effectuer, la vitesse à laquelle les résultats sont produits, les matériaux, les composants ou l'énergie utilisés pour leur fabrication et leur

fonctionnement, leur fiabilité, l'ingéniosité des mécanismes qui permettent la réalisation des calculs, le type de numération qu'elles utilisent (binaire, décimale, hexadécimale, etc.), ou la commodité de leur usage. Il s'agit donc de machines concrètes, physiquement réalisées, bien que leur finalité ne soit pas la production d'un travail ou la communication d'un mouvement mais la détermination d'une information.

Il semble que la première machine à calculer ait été réalisée par un professeur d'astronomie, d'hébreu et de mathématiques, Wilhelm Schickard (1592-1635), à Heidelberg en 1623, bien qu'il ne subsiste aucun exemplaire de cette machine, connue seulement par la description qu'en fait Schickard dans sa correspondance avec Kepler. Elle fut suivie par la Pascaline, construite par Blaise Pascal indépendamment de Schickard en 1643, et capable d'additionner, de soustraire, aussi bien que de convertir les monnaies. En 1673, Leibniz conçoit une machine qui effectue de manière automatique l'addition, la soustraction, la multiplication et la division. Toutefois, jusqu'à la fin du XVIII^{e} siècle, ces machines restent des curiosités admirées dans les cabinets et les salons, ou des instruments dont dispose une élite de géomètres, de comptables, ou d'astronomes. Ce n'est qu'au début du XIX^{e} siècle que l'on passe de la fabrication artisanale de quelques exemplaires à la production industrielle, notamment grâce à l'arithmomètre de Thomas, dont 500 exemplaires furent vendus entre 1821 et 1865, puis 1 000 exemplaires entre 1865 et 1878[1].

Ainsi définies, les machines à calculer ne semblent guère entretenir plus qu'un rapport métaphorique avec les machines de Turing. Le seul point commun entre une multiplicatrice mécanique et une machine abstraite conçue pour la multiplication est la relation entre les données initiales et le résultat, ainsi que l'existence d'une procédure permettant de passer des premières au second. Encore faut-il noter que dans une multiplicatrice comme la Pasca-

1. Cf. Jean Marguin, *Histoire des instruments et machines à calculer. Trois siècles de mécanique pensante. 1642-1942*, Paris, Hermann, 1994.

line, l'opération est nécessairement limitée au cas de nombres inférieurs à une borne fixée, et que s'il existe bien deux procédures effectives, elles sont difficilement comparables : dans le cas de la machine réelle, la procédure est liée à un mécanisme physiquement réalisé, propre à l'ingéniosité de son inventeur, alors que dans le cas de la machine de Turing, elle dépend de ce qu'il faut bien appeler la singularité d'une syntaxe, celle du tableau de commande et des signes lus, effacés, ou inscrits sur le ruban. La possibilité de mettre en rapport les machines de Turing et les calculateurs réellement existants suppose à la fois que l'on fasse abstraction de certaines contraintes et limitations physiques des machines et, aussi bien, que les machines à calculer dépassent le régime des opérations numériques usuelles pour atteindre, comme les machines de Turing, la possibilité d'un calcul généralisé. C'est dire que sans l'évolution technique qui conduit des premiers calculateurs mécaniques à l'ordinateur, on en serait réduit à constater cette platitude, que toutes les fonctions d'une machine comme la Pascaline sont Turing-calculables. La mise en correspondance des machines abstraites et des calculateurs concrets aura une tout autre portée si l'on réussit à montrer que certaines machines réelles sont effectivement capables de calculer toutes les fonctions Turing-calculables, abstraction faite de la durée des calculs et de l'espace nécessaire à leur écriture, deux grandeurs qui, même dans les cas finis, peuvent atteindre une taille astronomique, au point d'être rédhibitoire pour toute machine existante. Cependant, même si l'on ignore provisoirement la nécessité de faire abstraction de ces deux dimensions, il reste qu'aucune comparaison éclairante n'est possible avant qu'une multitude d'idées ingénieuses et de progrès techniques n'ait permis la transformation progressive des machines à calculer, depuis celle de Schickard jusqu'aux ordinateurs que nous connaissons. Alors, et alors seulement, il devient éclairant de considérer les machines abstraites comme le modèle théorique des calculateurs réels.

Pour autant, l'histoire de ces machines et de leur évolution est loin de suivre un cours régulier. Dans cette histoire, les origines, les filiations et les captages d'idées forment un réseau complexe. Il

inclut l'histoire de la numération, celles des techniques de calcul, de l'horlogerie mécanique, des automates, l'usage de la programmation pour le tissage mécanique de motifs répétés, ainsi que d'autres techniques apportant ou empruntant aux machines à calculer les conditions d'un progrès. Ce n'est qu'en s'autorisant du point de vue logique des machines de Turing que l'on jugera rétrospectivement des progrès, des oublis, des étapes décisives, des impasses, des découvertes essentielles, de longues périodes de stagnation, des précurseurs, des emprunts, des divergences et des mouvements confluents ; que l'on apercevra dans ce réseau d'échanges, la ligne de filiation qui unit, de la machine de Schickard aux ordinateurs, l'ensemble des calculateurs. Du moins est-ce là l'une des perspectives historiques parmi d'autres possibles. Orientée par les développements ultérieurs de l'informatique, et la mise en évidence *in fine* d'une correspondance étroite entre les machines abstraites de Turing et une certaine définition de ce qu'est un ordinateur, cette lecture de l'histoire des machines, fût-elle arbitraire ou partiale, a le mérite de lui donner un sens et de définir le principe d'une évaluation de ses étapes. Ainsi, il est possible, à la lumière des machines de Turing ou d'autres caractérisations des fonctions calculables, de reconnaître comme essentielles plusieurs découvertes ou inventions dans l'histoire des machines. Lorsque l'on adopte ce point de vue, on retient notamment l'apparition de l'enregistrement des résultats intermédiaires d'un calcul dans une partie de la machine appelée « mémoire », la mise en séquence d'un nombre limité d'opérations élémentaires éventuellement répétées un grand nombre de fois, la conception d'un organe de commande séparé du mécanisme qui effectue les opérations, la distinction entre la suite des opérations et la suite des données auxquelles ces opérations s'appliquent, le branchement conditionnel (qui détermine l'une ou l'autre de deux suites d'opérations en fonction d'un résultat intermédiaire), la possibilité de répéter une suite bien déterminée d'opérations aussi longtemps que telle ou telle condition est ou n'est pas réalisée, le codage numérique des adresses-mémoires ou des opérations, l'idée d'un langage de commande propre à la machine, ainsi que l'idée de

programme enregistré, ou encore la représentation codée de données non numériques quelconques[1]. Chacun de ces exemples a non seulement une importance technique dans l'évolution des machines à calculer, mais également une signification logique eu égard au modèle abstrait des machines de Turing. « Logique », aussi, au sens où ces exemples concernent tous l'organisation fonctionnelle de la machine, ainsi que l'évolution du concept de machine à calculer[2]. Lorsque l'on considère les machines actuelles qui disposent de toutes ces capacités, on en vient à douter qu'il existe un rapport de filiation entre la machine de Schickard et l'ordinateur, utilisé pour des tâches qui n'ont souvent, du moins à première vue, plus rien à voir avec le calcul numérique, comme son nom en français l'indique suffisamment[3]. Ce qui justifie ce rapport est l'idée d'une généralisation de la notion de calcul, posée du point de vue bien particulier de la logique, selon lequel les machines de Turing peuvent être considérées comme le modèle théorique des ordinateurs. En quel sens cela doit-il être compris ?

La machine de Turing universelle comme modèle théorique des ordinateurs

On s'accorde à nommer « ordinateur » un calculateur universel à programme enregistré. C'est en explicitant les termes de cette définition que l'on parvient à mettre en correspondance les

1. Si l'on voulait dater l'invention de chacune de ces techniques, ou en faire l'histoire, c'est certainement à l'œuvre de Charles Babbage et en particulier à son projet de machine analytique que l'on accorderait la plus grande importance. Sur cet aspect de la question, qui dépasse notre propos, cf. Mosconi, *op. cit.*
2. Plusieurs des progrès énumérés sont directement liés à l'émergence d'une réflexion sur les machines, conçues non seulement comme des artefacts utiles mais également comme l'objet d'une étude théorique, aux implications mathématiques et logiques. Ici encore, on pourra se référer à Mosconi, *op. cit.*
3. La langue anglaise distingue explicitement deux aspects de la discipline informatique : le traitement de données *(data processing)* et le calcul *(computing)*.

machines abstraites de Turing et les machines réelles[1]. Pour cela, on pourra se servir d'une caractérisation fonctionnelle à laquelle répondent la plupart des ordinateurs actuellement en usage, et qui fut donnée en 1944 par von Neumann dans un texte aujourd'hui classique, le *First Draft of a Report on the EDVAC.* Cinq parties fonctionnelles distinctes définissent ce que l'on appelle, par référence à cet article, les « machines de von Neumann » : une mémoire, un organe d'entrée, un organe de sortie, un dispositif de commande, et une unité de calcul.

1 / L'unité de calcul est capable d'effectuer successivement des opérations arithmétiques ou logiques dans un ordre déterminé par l'organe de commande. Les opérations arithmétiques peuvent être, par exemple, le successeur, la somme, le produit, ou la différence bornée[2]. Les opérations logiques sont celles du calcul propositionnel, effectuées sur les valeurs booléennes 0 et 1 (conjonction, disjonction, négation, etc.).

2 / L'organe de commande détermine l'ordre des suites d'opérations que la machine effectuera, ainsi que les données initiales du calcul, ou du moins l'adresse à laquelle ces données doivent être recherchées dans la mémoire.

3 / La mémoire est une partie de la machine qui permet de stocker des informations quelconques : nombres, résultats de calculs intermédiaires, codage de données initiales ou de suites d'opérations à effectuer.

4 / L'organe d'entrée est un dispositif qui permet à l'utilisateur de la machine d'y introduire toutes les informations utiles au déroulement du calcul, notamment les données auxquelles des pro-

1. Dans ce qui suit, on s'intéresse uniquement aux rapports logiques et théoriques entre les ordinateurs et les machines de Turing, sans considérer les influences historiques qui ont pu exister entre le développement de la théorie de la calculabilité d'une part et la construction des calculateurs universels d'autre part. Sur cette dernière question, on pourra consulter par exemple les ouvrages suivants : H. H. Goldstine, *The computer from Pascal to von Neumann*, Princeton, NJ, Princeton UP, 1972 ; Herken, *op. cit.* ; Hodges (cf. réf. ci-dessous, p. 63, n. 1) ; Mosconi, *op. cit.* ; B. Randell, *The origins of digital computers. Selected papers*, Berlin, Springer, 1972.
2. Pour la différence bornée, $x \dot{-} y = 0$ dans les cas où $y \geqslant x$.

cédures de calcul seront appliquées, ou l'expression codée de ces procédures elles-mêmes ; ces informations sont placées dans la mémoire avant d'être utilisées par la machine lorsqu'elle effectue les opérations.

5 / L'organe de sortie permet de recueillir les différents résultats des calculs, ou d'être informé sur le contenu de la mémoire. Cependant, tous les résultats des calculs intermédiaires ne sont pas nécessairement accessibles à l'utilisateur par l'organe de sortie.

Les machines de von Neumann passent d'un état à un autre par transitions discontinues, et sont capables d'effectuer successivement, dans l'unité de calcul, une suite d'opérations programmées par l'utilisateur. Deux opérations élémentaires ne peuvent être effectuées dans le même temps, ou parallèlement, et en ce sens, ces machines sont dites « séquentielles ». C'est l'organe de commande qui détermine l'ordre dans lequel les opérations sont effectuées, en respectant le programme établi par l'utilisateur. Comme on le voit, la définition de ces machines est purement fonctionnelle et fait donc entièrement abstraction des matériaux servant à sa construction, de l'énergie qui rend possible son fonctionnement, de la numération utilisée, de l'espace-mémoire disponible pour l'enregistrement des informations utiles aux calculs, de la vitesse d'exécution des procédures, ainsi que d'autres caractéristiques techniques des machines réelles ; ce qui permet d'apercevoir une première correspondance avec les machines de Turing, elles aussi séquentielles et fonctionnant par transitions discontinues entre les états. Le ruban de la machine de Turing sert en effet à la fois d'organe d'entrée, de mémoire et d'organe de sortie. Aux capacités de l'unité de calcul correspondent les possibilités de lecture, d'écriture, d'effacement et de déplacement de la tête par rapport au ruban. Et dans la machine de Turing, chacune de ces opérations, ainsi qu'un éventuel changement d'état, est déterminé par un tableau de quintuplets, qui remplit exactement la fonction d'un organe de commande. On voit donc qu'une machine de Turing physiquement réalisée dispose bien des cinq parties fonctionnelles des machines de von Neumann.

Lorsqu'il est question de l'organe de commande, au paragraphe deux du *First Draft,* von Neumann introduit une distinction importante entre

> les instructions spécifiques données pour un problème particulier, qui définissent ce problème, et les organes généraux de contrôle qui s'assurent que ces instructions – quelles qu'elles soient – sont effectuées[1].

Des suites d'instructions différentes forment, dans la machine de von Neumann, autant de programmes susceptibles d'être enregistrés dans la mémoire. Pour reprendre la comparaison avec les machines de Turing, ces suites d'instructions peuvent être rapprochées des différentes listes de quintuplets possibles. Par conséquent, dans cette comparaison, une machine de Turing particulière ne correspond pas à une machine de von Neumann mais à l'un des programmes qui peuvent être enregistrés dans sa mémoire. Ce que l'on appelle « une machine de von Neumann » est donc un dispositif physique qui devient une machine particulière lorsqu'on lui communique une liste d'instructions bien déterminée. En ce sens, il y a donc autant de machines particulières que de programmes donnés à l'organe général de commande, de même qu'il existe autant de machines de Turing que de listes de quintuplets répondant aux spécifications requises dans l'article de 1936.

Se pose alors naturellement la question de savoir si un calculateur comme une machine de von Neumann permet de mettre en œuvre toutes les procédures effectives, ou, en d'autres termes, de calculer toutes les fonctions calculables. Par la thèse de Turing, la question revient à déterminer s'il peut calculer toutes les fonctions *Turing*-calculables. Comme on l'a déjà souligné, une réponse positive suppose que l'on fasse abstraction de plusieurs limitations physiques des machines réelles, puisqu'il n'existe aucune borne à la longueur d'un calcul dans sa durée, ni à l'espace d'écriture qu'il requiert. On supposera donc que la taille de la mémoire d'une machine réelle est arbitrairement grande (ou qu'il est toujours possible de l'augmenter), et

1. Nancy Stern, *From ENIAC to UNIVAC. An appraisal of the Eckert-Mauchly Computer,* Bedford, Mass., Digital Press, 1981, p. 183.

qu'il n'existe aucune limite au nombre des opérations successivement effectuées (ou encore que la machine peut fonctionner pendant une durée illimitée). Or, abstraction faite de ces deux contraintes physiques, on peut prouver qu'un petit nombre de conditions sur les capacités de calcul et de programmation suffisent pour qu'un dispositif à la von Neumann soit universel, c'est-à-dire pour qu'il soit capable, par une programmation adéquate, de calculer n'importe quelle fonction Turing-calculable. Pour prendre un exemple, on démontre en théorie de la calculabilité que si la machine peut effectuer l'addition, le produit, et la différence bornée, si la machine dispose de l'itération conditionnelle[1], et s'il est possible de programmer n'importe quelle suite finie d'opérations (choisies parmi celles qui sont disponibles), alors toute fonction numérique Turing-calculable est programmable[2]. Ces quelques conditions suffisent donc pour qu'un dispositif à la von Neumann soit un calculateur universel.

La construction d'une correspondance entre les machines de Turing et les machines réelles est donc possible, pourvu que celles-ci disposent de quelques capacités minimales. Ce qui vient d'être montré, ce n'est pas qu'une machine de Turing est physiquement réalisable, car cela supposerait la présentation d'un dispositif qui en assure le fonctionnement, par une connexion du ruban, de la tête de lecture-écriture-effacement et du tableau de commande. Comme on l'a déjà souligné, l'intérêt d'une telle réalisation serait très faible : ce que l'on attend d'une machine réelle est un ensemble de performances et une commodité d'usage dont Turing ne s'est évidemment pas du tout soucié en définissant ses machines abstraites, uniquement conçues pour la solution d'un problème de logique. Ce qui vient d'être montré, c'est plutôt qu'un certain type de machine à calculer, héritage entrecroisé de techniques de numérations, d'une maîtrise des mécanismes, en horlogerie notamment,

1. L'itération conditionnelle est une opération qui demande la répétition d'une suite d'instructions aussi longtemps qu'une certaine condition, exprimée sous la forme d'un test, est ou n'est pas réalisée. Elle réunit deux capacités : la répétition d'une même séquence d'instructions, et le branchement conditionnel.
2. Cf. Herken, *op. cit.*, p. 52-53.

ainsi que d'une multitude d'autres savoirs et savoir-faire, fruit de l'histoire et de ses contingences, des besoins humains et de ses aléas, limité par les contraintes physiques des matériaux, peut être très précisément mis en correspondance avec un outil mathématique abstrait initialement conçu pour la solution d'un problème de logique. Et qu'en ce sens, ce que dit la logique au sujet des machines ne concerne pas seulement des dispositifs abstraitement définis, mais également, par la construction de cette correspondance, les machines réelles que nous utilisons. Ou encore, que les unes servent de modèle théorique aux autres.

Pour compléter cette interprétation, il reste encore à comprendre la signification logique de la notion de programme enregistré, sans laquelle un calculateur, même universel, n'est pas encore un ordinateur. Communiquer à une machine le programme des calculs qu'elle effectuera est un problème susceptible de recevoir de multiples solutions techniques. Le projet de machine analytique de Babbage, inspiré sur ce point par une méthode qu'utilisait Jacquart pour le tissage, prévoyait que la programmation se ferait grâce à l'usage de cartes perforées : selon l'emplacement des perforations sur la carte, tel ou tel motif était tissé, et telle ou telle procédure de calcul devait être effectuée. Une machine comme l'ENIAC, premier calculateur universel électronique, inauguré en 1946, utilisait un tableau de connexions sur lequel on pouvait changer l'emplacement des fiches afin de déterminer un programme particulier. L'idée que le programme puisse être enregistré dans la mémoire de la machine fut clairement exprimée pour la première fois dans le *First Draft.* Bien que ce texte soit signé « Von Neumann », cette idée est vraisemblablement due à Mauchly et Eckert, deux membres de la Moore School de l'Université de Pennsylvanie, à Philadelphie, qui travaillaient sur le projet de l'EDVAC[1]. D'un point de vue technique, l'idée suppose que la machine dispose d'une mémoire de taille suffisante, mais elle présente de gros avantages. Tout d'abord,

1. Cf. René Moreau, *Ainsi naquit l'informatique. Histoire des hommes et des techniques,* Paris, Dunod, 1987, p. 39-51, et Nancy Stern, *op. cit.*

la répétition d'une même suite d'instructions ne suppose plus que cette séquence soit réintroduite autant de fois dans la machine, comme lorsqu'il s'agit de cartes perforées. Ensuite, les instructions peuvent être numériquement codées, et placées ainsi en mémoire. Elles apparaissent donc comme des données particulières sur lesquelles rien n'empêche d'effectuer des opérations programmées (le programme peut donc se modifier lui-même) ; la vitesse d'exécution des calculs est ainsi augmentée, et la commodité d'usage de la machine accrue. Enfin, la commande du déroulement des programmes peut elle-même faire l'objet d'un programme enregistré (appelé « programme de contrôle » ou « programme de commande »), et elle peut ainsi être aisément modifiée.

L'idée de programme enregistré apparaît donc comme un progrès technique important, bien qu'elle ne modifie pas le principe selon lequel à chaque programme correspond une procédure de calcul, une fonction calculable, ou encore une machine bien particulière. Ce qui change est que cette machine-programme (capable de calculer, par exemple, l'extraction d'une racine, ou de trier une liste de données) est entièrement caractérisée par un code enregistré. Ce code est communiqué à une machine physique dotée d'un programme de commande qui est chargé de contrôler l'exécution de tout programme qu'on pourra lui confier. Un seul et même dispositif physique peut ainsi devenir, selon le programme qui est enregistré, n'importe quelle machine particulière. Or si la suite d'instructions qui constitue un programme correspond à la liste de quintuplets qui forme une machine de Turing, la machine de von Neumann (équipée d'organes de commande permettant d'assurer la bonne exécution des instructions inscrites en mémoire) peut être rapprochée de la machine universelle au sens de Turing. Celle-ci est en effet capable de recevoir la représentation standard de n'importe quelle procédure singulière définie par une liste de quintuplets, tout comme les données initiales auxquelles cette procédure pourra être appliquée ; capable également d'effectuer cette application, simulant de cette manière le travail de la machine codée par le programme. On comprend ainsi en quel sens la machine de

Turing universelle peut être considérée comme le modèle théorique des calculateurs universels à programmes enregistrés[1] : par la correspondance que l'on vient d'expliciter entre machine abstraite et machine réelle, les propriétés logiquement démontrables au sujet de la première s'appliquent à tout mécanisme ou dispositif physique qui répond à la définition de l'ordinateur.

Machines de Turing et programmes informatiques

L'idée de programme enregistré est un pas décisif accompli sur la voie d'une dématérialisation des machines, qui ne correspond pas seulement à la disparition progressive des mouvements mécaniques internes au profit de l'électronique[2], mais également à l'écriture de logiciels derrière lesquels le fonctionnement de la machine physique finit par devenir invisible, et entièrement inaccessible à l'utilisateur. Les premiers ordinateurs étaient programmés dans un langage où chaque instruction correspondait à une opération que la machine pouvait directement effectuer. Lire et déchiffrer un programme dans le « langage machine » revenait donc à dérouler mentalement la suite des opérations physiquement réalisées par la machine. Mais dès que les tâches ou les fonctions ainsi programmées ne se réduisent plus à

1. A condition de supposer, ce qui n'est pas immédiatement évident, que les opérations nécessaires à l'exécution du programme que constitue par elle-même la machine de Turing universelle puissent être effectuées par un dispositif physiquement réalisé.
2. Selon Moreau, *op. cit.*, le premier calculateur universel fut l'IBM ASCC *(Automatic Sequence Controlled Calculator)*, machine « construite par la compagnie IBM en collaboration avec l'Université d'Harvard », « plus connue sous le nom de machine Harvard-IBM ». Sa construction fut achevée en 1944. Selon le même auteur, le premier ordinateur, construit entre 1945 et 1948, est l'IBM SSEC *(Selective Sequence Electronic Calculator)*. Aucune de ces deux machines, cependant, n'étaient entièrement électronique. L'ENIAC, ou *Electronic Numerical Integrator And Computer*, construit en 1946 par la Moore School, à Philadelphie, est le premier calculateur universel électronique. Enfin, l'EDSAC ou *Electronic Delay Storage Automatic Calculator*, réalisé en 1949 par l'Université de Cambridge en Grande-Bretagne, est le premier ordinateur entièrement électronique.

une procédure triviale, le travail de déchiffrement des instructions codées devient rapidement un véritable casse-tête, d'une complexité rédhibitoire pour l'entendement humain. Pratiquement, la communication homme-machine est ainsi trop malaisée, et le temps passé au travail de la programmation beaucoup trop long, d'autant qu'il se faisait le plus souvent dans l'alphabet binaire, plus adapté à la machine qu'à son utilisateur. Dans les années cinquante, une grande partie des efforts fournis par les informaticiens fut consacrée à la conception de langages dits plus « évolués », c'est-à-dire moins éloignés de l'usage courant, et donc plus facilement compréhensibles : les instructions reçurent des dénominations symboliques qui ne furent plus des codes binaires ésotériques ; l'adressage des données et des instructions put lui aussi devenir symbolique, grâce à une gestion automatisée de la mémoire[1] ; des bibliothèques de sous-programmes furent constituées, qui assemblaient chacun, sous une dénomination symbolique, une suite d'instructions permettant le calcul d'une fonction ou d'une procédure fréquemment utilisée ; les programmes prirent alors la forme d'une séquence de sous-programmes, qu'ils fussent empruntés à une bibliothèque logicielle préalablement enregistrée ou écrits spécialement pour la réalisation d'une tâche singulière ; et les langages plus évolués se dotèrent même d'une capacité d'analyse syntaxique des expressions formelles que sont les programmes. Se constitua ainsi, au dessus du langage machine primitif, une hiérarchie de langages plus évolués, dans lesquels les programmes s'écrivirent beaucoup plus facilement[2]. Des programmes spéciaux furent aussi conçus, dévolus à la traduction automatique des procédures écrites dans un langage de haut niveau en une suite de codes directement interprétables par la machine. Selon leur mode de fonctionnement, on appelle soit « inter-

1. On n'indique plus à la machine l'adresse physique d'un espace-mémoire où une donnée doit être enregistrée. On se contente de choisir un nom désignant symboliquement un espace mémoire disponible que la machine recherche automatiquement pour enregistrer la donnée. Dans un programme, l'indication de ce nom suffit alors pour que la machine retrouve d'elle-même ce qui a été enregistré.
2. L'un des premiers langages évolués fut Fortran. A titre d'exemples, on peut citer aussi Algol, Lisp, Pascal, Prolog, Ada, ML, etc., qui sont, les uns et les autres, des langages de conceptions très différentes.

préteurs » soit « compilateurs » ces programmes de traduction. Ainsi se comprend mieux, du point de vue de l'utilisateur, la « disparition » progressive du fonctionnement de la machine physique derrière une hiérarchie logicielle de langages de plus en plus évolués. Pour prendre un exemple précis, par l'instruction suivante :

compteur : = 7 × N

on demande de conserver en mémoire, à une adresse nommée symboliquement « compteur », le résultat du produit de 7 par le nombre enregistré en mémoire à une adresse symboliquement dénommée N. En écrivant cela, le programmeur ne se soucie ni de l'endroit physique où sont enregistrées les valeurs des deux identificateurs « compteur » et « N », ni de savoir quelle procédure est utilisée par la machine pour effectuer la multiplication, ou pour affecter la valeur de son résultat à l'adresse « compteur ».

On voit ainsi en quel sens la machine réelle s'efface aux yeux de l'utilisateur. Il n'en demeure pas moins que chaque programme écrit dans un langage évolué peut être considéré comme une machine, concrète dans la mesure où il existe un dispositif physique capable d'exécuter chacune des instructions, mais abstraite au sens où son fonctionnement peut être décrit au niveau du langage, en ignorant le détail des processus physiques sous-jacents. *Les programmes informatiques peuvent donc être considérés comme des machines qui prennent la forme d'un texte écrit dans un langage artificiel, ou encore d'une suite de signes régis par des règles morphologiques et une syntaxe, comparables à cet égard aux formules et aux preuves d'un système formel en logique.* On appelle précisément « logiciels » ces machines qui se réduisent à des entités *linguistiques.*

La même question d'universalité qui se posait au sujet des machines de Turing et des machines physiquement réalisées, se présente maintenant relativement aux programmes écrits dans un langage évolué : quelle est la classe des fonctions calculables par ces machines logicielles ? quelles sont les procédures effectives qu'elles sont capables de définir ? Pour apporter une réponse à ces questions, l'informatique théorique donne une nouvelle preuve d'équivalence, entre le dispositif des machines de Turing et la plupart des langages

évolués conçus comme des langages de programmation (notamment tous ceux qui ont été cités à titre d'exemples : Fortran, Algol, Lisp, Pascal, Prolog, Ada, ML). En d'autres termes, ces langages peuvent être considérés comme des machines logicielles permettant la programmation de n'importe quelle fonction Turing-calculable.

En outre, dans la mesure où il existe également des programmes spéciaux permettant de contrôler l'exécution des instructions que l'on a pu écrire pour la réalisation d'une tâche ou d'une fonction quelconque, l'utilisateur d'un ordinateur dispose d'un ensemble d'outils logiciels[1] qui constituent l'exact équivalent de la machine de Turing universelle. Du point de vue de cette équivalence, à chaque programme particulier écrit dans un langage évolué correspond la représentation standard d'une machine de Turing, c'est-à-dire le code que l'on inscrit sur le ruban de la machine universelle. On voit par là que les machines abstraites de Turing sont autant le modèle théorique de ces machines logicielles, ou programmes, que des machines physiques, dont le fonctionnement reste invisible et non essentiel pour l'utilisateur de l'ordinateur dans la plupart des cas.

La notion de *machine informatique* peut donc être définie d'un point de vue abstrait et purement logique : qu'elle prenne la forme matérielle d'un calculateur physiquement réalisé ou celle d'une machine logicielle (un programme susceptible d'être mis en œuvre indifféremment par de nombreux dispositifs physiques distincts), elle est dans un rapport d'équivalence aux machines de Turing. On appellera *machine (informatique) universelle* tout dispositif, qu'il soit abstrait ou concret, matériel ou logiciel, syntaxique ou physiquement réalisé, pourvu qu'il soit susceptible, au moins en principe, de calculer toutes les fonctions Turing-calculables ; et plus généralement *machine informatique* tout dispositif équivalent à une ou plusieurs machines de Turing[2].

1. C'est-à-dire d'un ensemble de programmes.
2. Cette définition très large fait de la Pascaline, par exemple, une machine informatique. Elle pourra même sembler trop large si l'on considère qu'à la limite, toute chose peut être interprétée comme une machine de Turing. Le plus souvent, cependant, on ne parlera de machine informatique qu'au sujet des dispositifs physiques expressément conçus pour le calcul.

Par cette définition, on se donne la possibilité d'une application extrêmement large des résultats obtenus au sujet des machines abstraites en logique. Ce qui suppose que l'on ne fasse pas seulement abstraction des limites de toute machine physique, du double point de vue de l'espace mémoire disponible pour l'écriture des calculs intermédiaires et de la durée des calculs, mesurée par le nombre des opérations élémentaires effectuées. Car en établissant l'équivalence de plusieurs dispositifs ou formalismes, on efface également les différences et les singularités des procédures utilisées pour calculer une seule et même fonction. Or il existe une infinité de machines de Turing différentes pour l'addition de deux entiers, et, *a fortiori*, autant de machines informatiques possibles, sous la forme de calculateurs physiquement réalisables ou de programmes. Si une théorie générale de la calculabilité peut bien faire abstraction de toutes ces différences, il n'en va plus de même lorsqu'elle se donne pour objet la complexité des calculs, ou la comparaison des différentes procédures permettant le calcul d'une seule et même fonction. Alors se pose le problème d'une équivalence plus fine entre les algorithmes, pour la solution duquel le choix du dispositif ou du formalisme utilisé n'est plus indifférent. Aussi importe-t-il de savoir que d'autres caractérisations logiques des fonctions calculables sont possibles, équivalentes aux machines de Turing du point de vue abstrait de la calculabilité, mais dont la singularité syntaxique peut également être considérée pour elle-même.

En examinant l'exemple du lambda-calcul, on se limitera cependant, au moins provisoirement, au point de vue le plus abstrait qui offre un argument supplémentaire pour la thèse de Turing ; et dans le même temps, on montrera une nouvelle fois comment un formalisme élaboré pour la résolution de problèmes purement logiques peut être interprété en termes de machines et utilisé comme un langage de programmation.

IV

LES MACHINES A RÉDUCTION DU LAMBDA-CALCUL

LES ORIGINES LOGIQUES DU LAMBDA-CALCUL

L'expression du problème de la décision en termes de machines est l'une des singularités de la pensée de Turing en 1936. Elle doit certainement beaucoup à l'intérêt que le jeune Alan Mathison Turing avait porté aux questions relatives aux machines et à leurs rapports avec le cerveau[1]. Cette circonstance n'enlève évidemment rien ni à la valeur ni à la portée de ses travaux, mais signifie seulement qu'une analyse reposant sur la notion de machine ne se présentait pas comme la seule manière possible de résoudre le problème de la décision. On en prendra pour preuve la solution de Church élaborée à partir d'un formalisme très différent des machines de Turing, le lambda-calcul, dont certains principes se trouvent déjà chez un autre logicien qui travaille lui aussi sur les fondements de la logique.

Au début des années vingt, Schönfinkel s'efforce de réduire toutes les formules générales de la logique à la combinaison de quelques opérateurs primitifs. Il est à la recherche d'une syntaxe permettant d'exprimer ces formules en faisant l'économie des notions de proposition, de fonction propositionnelle, et de variable.

1. Cf. Andrew Hodges, *Alan Turing : the enigma of intelligence,* Londres, Unwin Paperbacks, 1985, trad. franç., *Alan Turing : l'énigme de l'intelligence,* Paris, Payot, 1988.

Un exposé de ses travaux est publié dans un article de 1924, *Über die Bausteine der mathematischen Logik*[1]. Les fonctions sont conçues comme des opérateurs et font l'objet d'une analyse dont Church s'inspirera pour la conception du lambda-calcul. Schönfinkel développe l'idée suivante qu'avait déjà aperçue Frege[2] : une fonction binaire, F, peut être pensée comme une fonction unaire dont la valeur est elle-même une autre fonction unaire. En d'autres termes, $F(x,y)$ est conçue comme $(Fx)\,y$. Par exemple, si F est l'opérateur qui effectue le produit de deux entiers, en appliquant F à 2, on obtient la fonction (F2) qui à tout x associe $2x$. La notation (F2)7 signifie que l'on applique d'abord F à 2, puis la fonction unaire ainsi obtenue à 7. Et cette remarque se généralise aisément au cas d'une fonction n-aire[3]. Cette analyse est en accord avec une conception des fonctions bien différente de la conception ensembliste. Dans les années vingt et trente, pour des logiciens comme Schönfinkel et Church, une fonction n'est pas conçue comme un ensemble de paires ordonnées mais comme un opérateur qui transforme un objet en un autre, ou dont l'application a pour effet de produire une valeur[4]. Dans l'exemple précédent, F est un opérateur qui produit la fonction unaire (F2) lorsqu'on l'applique au nombre 2, et (F2) un opérateur qui produit la valeur 14 lorsqu'on l'applique à 7. Or de ce point de vue, rien n'empêche *a priori* d'appliquer une fonction à une autre, ou même un opérateur à lui-même. Par exemple, si F représente un opérateur multiplicatif et G un opérateur additif, on peut donner un sens opératoire aux expressions FG, FF, G(FG), si l'on ne restreint pas l'application de F et G aux nombres. Ainsi (GG) pourrait représenter un opérateur ternaire

1. Moses Schönfinkel, Über die Bausteine der mathematischen logik, *Mathematische Annalen*, 92, 1924 ; trad. angl. *in* Jean van Heijenoort, *From Frege to Gödel. A Source Book in Mathematical Logic, 1879-1931*, Cambridge, Mass., Harvard University Press, 1967.
2. Sur ce point, lire l'introduction que Quine rédigea pour la traduction anglaise de l'article de Schönfinkel.
3. On utilise la convention de parenthésage à gauche. Ainsi, Fxy est une écriture simplifiée de $(Fx)y$, et de la même manière, $Gxyzt$ représente $((Gx)y)z)t$, qui signifie que la fonction G est appliquée à x, la fonction (Gx) à y, $((Gx)y)$ à z, etc.
4. Cf. les remarques de Roger Hindley et Jonathan Seldin dans *Introduction to Combinators and λ-Calculus*, Cambridge, Cambridge UP, 1986, § 3E.

qui, appliqué successivement aux entiers y, z et t produirait la somme de y, z et t.

Dans l'article de 1924, Schönfinkel utilise trois opérateurs primitifs, C, S et U, ainsi que la possibilité de les combiner en les appliquant les uns aux autres, et montre que ces moyens suffisent à exprimer n'importe quelle formule de logique qui ne comporte aucune variable libre[1]. Quelques années plus tard, à partir de la fin des années vingt, cette tentative pour fonder la logique sur la combinaison de quelques opérateurs primitifs est reprise et développée par H. B. Curry et quelques autres sous le titre de « logique combinatoire ».

Cette analyse des constituants et des opérations primitives de la logique et des mathématiques (Curry parle d'une étude « prélogique ») vise évidemment la question des fondements et la résolution des paradoxes, bien que son intérêt vienne aussi de ce qu'elle met en lumière certaines propriétés des fonctions lorsqu'elles sont conçues comme des opérateurs. Or à l'origine, le lambda-calcul s'inscrit dans le même contexte que la logique combinatoire. Dans deux articles publiés en 1932 et 1933, Church énonce un ensemble de postulats pour le fondement de la logique, dans le but avoué de démontrer la consistance de la théorie des nombres, par construction d'un système auquel les théorèmes de Gödel ne puissent pas être appliqués[2]. Du travail de Schönfinkel, Church retient l'idée d'une réduction des fonctions n-aires aux fonctions unaires, et la possibilité d'une construction combinatoire par application d'une fonction à une autre. Mais il ne cherche pas à faire l'économie des variables, et introduit la notation suivante : M étant une combinaison de fonctions et d'arguments contenant la variable libre x, le terme $\lambda x.M$ représente la fonction qui à x associe M. L'*application*

1. Bien plus, si $A[x_1, \ldots, x_n]$ est une formule logique qui comporte $x_1, \ldots, x_n$ comme seules variables libres, elle peut être représentée par une combinaison de U, C, S et $x_1, \ldots, x_n$.
2. Alonzo Church, A set of postulates for the foundation of logic, *Annals of Math.*, 2 s., 34, 1932, p. 346-366, et Alonzo Church, A set of postulates for the foundation of logic (second paper), *Annals of Math.*, 2 s., 34, 1933, p. 839-864. Sur la possibilité d'une preuve de consistance après 1931, cf. le second de ces deux articles, p. 842-843.

de cette fonction au point N se note alors $(\lambda x.M)N$, et l'on appelle *« abstraction »* la construction de $(\lambda x.M)$ à partir de M[1]. Les expressions *x*, M, N, $(\lambda x.M)$ et $(\lambda x.M)$ N peuvent être considérées comme autant de termes d'un calcul sur les fonctions. On verra ci-dessous que ces notations permettent de donner une représentation codée des entiers et de plusieurs fonctions numériques, qui sont alors insérées dans un système logique plus complet, comprenant un ensemble de postulats pour la théorie des nombres.

Dans l'esprit de Church, cette formalisation de l'arithmétique devait être ensuite étendue aux nombres rationnels et aux nombres réels[2]. Mais l'ensemble des postulats se révéla contradictoire, comme le prouvèrent Kleene et Rosser, deux élèves de Church, dans un article publié dès 1935. Si le système dans son ensemble perdait par là toute valeur pour le fondement des mathématiques, les résultats obtenus par Kleene avant la découverte d'une contradiction étaient suffisamment importants pour qu'il ne fût pas entièrement abandonné. Sa syntaxe intégrait les deux idées d'*abstraction* et d'*application* dont Church souligna l'intérêt pour la représentation et l'étude des fonctions. Ces idées furent donc séparées du système logique dans lequel elles étaient insérées, et servirent de base pour l'élaboration du lambda-calcul.

Rosser mit en évidence une correspondance étroite entre ce calcul et la logique combinatoire, alors que Kleene étudiait la possibilité de définir et de représenter dans ce formalisme les fonctions connues en théorie des nombres. Le travail de Kleene put ainsi révéler que le lambda-calcul possédait des ressources définitoires si importantes que Church fut conduit, en 1935, à énoncer la fameuse thèse selon laquelle toutes les fonctions calculables peuvent être représentées par un terme du lambda-calcul, et calculées dans ce formalisme.

C'est en termes de fonctions récursives que Church énonça publiquement sa thèse pour la première fois, bien que Kleene eût

1. De même que M, N doit être ici considéré comme une variable syntaxique du métalangage : elle représente un lambda-terme quelconque et n'appartient pas au langage formel lui-même.
2. Cf. Church, *op. cit.*, 1933, p. 864.

démontré dès 1935 que toute fonction lambda-définissable est une fonction récursive, ainsi que la proposition inverse, avant même que l'invention par Turing de ses machines abstraites ne le conduise indépendamment, dans l'article publié en 1936, à soutenir une thèse équivalente[1]. Dans un appendice du même article, Turing esquissait une preuve d'équivalence similaire entre le lambda-calcul et les machines de Turing.

Pour préciser le sens de cette équivalence, il faut maintenant définir ce qu'est un terme du lambda-calcul, et comment une fonction numérique peut être à la fois représentée et calculée dans ce formalisme. Comme pour l'analyse des machines de Turing, on ne s'attachera pas, dans ce qui suit, à rapporter les singularités du lambda-calcul tel qu'il fut initialement élaboré par Church, ni à en détailler les variantes, ni à suivre les points sur lesquels Kleene fut amené à le modifier. Une présentation sommaire de sa syntaxe et de ses propriétés suffit en effet à comprendre pourquoi ce formalisme purement logique à l'origine peut être interprété en termes de machines, et considéré en informatique comme un langage de programmation.

LA SYNTAXE DU LAMBDA-CALCUL ET QUELQUES-UNES DE SES PROPRIÉTÉS

Les termes du lambda-calcul se construisent à partir de variables (x, y, z, etc.) au moyen de deux opérations appelées « application » et « abstraction ». Formellement, on donne la définition inductive suivante :

(i) les variables sont des termes du lambda-calcul ;
(ii) si t est un terme, et x est une variable, alors l'opération d'ab-

1. Cf. Davis, *op. cit.*, 1982. Selon Davis, la thèse de Church fut conçue dès 1934, sans avoir été publiée à cette date.

straction permet de construire le terme $\lambda x.t$ (qui se lit : « lambda x point t ») ;

(iii) si t et u sont deux termes, l'opération d'application de t à u permet de construire le terme $(t)u$ qui se lit : « t appliqué à u »[1] ;

(iv) seules les trois règles précédentes servent à construire les termes.

Comme on l'a vu, $\lambda x.t$ peut être compris intuitivement comme représentant la fonction qui à x associe t, et $(t)u$, comme l'application d'une fonction t à un argument u. Par exemple, si le terme t est une représentation de l'expression algébrique $x^2 + 3x$, alors la fonction habituellement notée $f : x \to x^2 + 3x$ pourra être représentée par le terme $\lambda x.t$. Dans les mathématiques usuelles, l'application de f à 7 se note souvent $f(7)$, et pour évaluer cette expression, on remplace les occurrences de x par 7 dans $x^2 + 3x$ (on obtient ainsi $7^2 + 3 \times 7$). Dans le lambda-calcul, si le terme u représente 7, l'application de f à 7 s'écrit $(\lambda x.t)u$, et l'évaluation de ce terme s'obtient en remplaçant les occurrences libres de x dans t par u. On note $t[u/x]$ le résultat de cette évaluation, c'est-à-dire de la substitution de u à x dans t.[2] L'évaluation de $f(7)$ correspond dans le lambda-calcul à une opération appelée *β-réduction*, qui à $(\lambda x.t)u$ associe $t[u/x]$. La β-réduction se note $\to_\beta$ et l'on écrit donc $(\lambda x.t)u \to_\beta t[u/x]$, ce qui se lit : « Le terme $(\lambda x.t)u$ se réduit, par β-réduction, à $t[u/x]$. » Un terme de forme $(\lambda x.t)u$, qui peut être réduit de cette manière, est appelé β-redex, ou simplement redex[3].

1. Cette notation peut être ambiguë, par exemple si u n'est rien d'autre que $(x)\,x$, puisque l'on peut comprendre soit $((t)\,x)\,x$ soit $(t)\,((x)\,x)$. Il faut, dans ce cas, ajouter une paire de parenthèses, en écrivant $(t)\,(u)$. Church utilisait la notation suivante : $\{t\}(u)$, qui élimine, au prix de quelque lourdeur, toute ambiguïté.
2. Dans cette présentation informelle, nous ignorons les problèmes de conflit de variables, liés à l'opération de substitution. On notera seulement que dans le terme $(\lambda x.t)$, la variable x est considérée comme étant liée, comme dans la formule logique $\forall x$A. On considère donc comme équivalents les termes $(\lambda x.t)$ et $(\lambda y.t[y/x])$, lorsque y ne figure pas dans t, de même que l'on écrit indifféremment $\forall x$A ou $\forall y$A$[y/x]$, pourvu que y ne figure pas dans A. Les variables libres du lambda-calcul sont celles qui ne sont pas liées par une abstraction.
3. Le mot « redex » vient de l'expression anglaise *reducible expression*. En français, le substantif « radical » est également usité dans le même sens.

Toutefois, les termes qu'il est possible de construire formellement ne peuvent pas tous recevoir l'interprétation qui vient d'être proposée. Par exemple, $(x)x$ ou $(x)\lambda x.(yy)$ sont des termes dont la construction syntaxique est correcte mais pour lesquels l'interprétation intuitive précédente n'a aucune signification claire. Ajoutons aussi que la β-réduction ne s'applique pas seulement à un redex mais également à tout terme qui contient un redex. Par exemple :

$$(x)\ ((\lambda y.y)\ (\lambda z.(z)\ z)) \rightarrow_\beta (x)\ (\lambda z.(z)\ z).$$

Dans ce cas, le redex $(\lambda y.y)\ (\lambda z.(z)\ z)$ se trouve « à l'intérieur » du terme initial. Plusieurs β-réductions peuvent se suivre, et la réduction $t \rightarrow_\beta u$ se fait donc en une ou plusieurs étapes. En réalité, la relation $\rightarrow_\beta$ est la clôture réflexive et transitive d'une relation plus simple appelée « β-réduction en une étape ». De plus, un seul et même terme peut comporter plusieurs redex ; dans ce cas, il y a plusieurs chemins de réduction possibles.

Un terme est dit *normal* (ou en forme normale) s'il ne comporte aucun redex. Un terme peut être transformé en un terme normal par β-réduction ; mais il est important de remarquer que certains termes ne peuvent pas être réduits en un terme normal[1]. Lorsqu'il existe un chemin de réduction qui réduit t à un terme normal, on dit que t est *normalisable.* Si de surcroît tous les chemins de réduction conduisent à un terme normal, t est *fortement normalisable.* De plus, selon le théorème de Church-Rosser, démontré en 1936, si $t \rightarrow_\beta u$ et si $t \rightarrow_\beta v$, alors il existe un terme w tel que $u \rightarrow_\beta w$ et $v \rightarrow_\beta w$. Une conséquence de ce théorème est que si un terme se réduit en un terme normal, celui-ci est unique. Il donne aussi une preuve de consistance du lambda-calcul, au sens où tous les termes ne se réduisent pas à un terme unique.

Pour donner un sens à la thèse de Church, il faut encore montrer comment les entiers et les fonctions numériques peuvent être

1. Par exemple, le terme $(\lambda x.(x)\ x)\ (\lambda x.(x)\ x)$. Si l'on appelle t ce terme, la seule réduction possible est $t \rightarrow_\beta t$. Dans ce cas, le seul chemin de réduction possible est $t \rightarrow_\beta t \rightarrow_\beta t \rightarrow_\beta t...$

représentés dans ce formalisme. Suivant l'idée de Church, on représente les entiers de la manière suivante :

— 0 est représenté par le terme $\lambda x.\lambda y.y$;
— 1 par le terme $\lambda x.\lambda y.xy$ (abréviation de $\lambda x.\lambda y.(x)\,y$) ;
— 2 par le terme $\lambda x.\lambda y.x(x(y))$ (abréviation de $\lambda x.\lambda y.(x)\,((x)\,y)$) ;
— et plus généralement l'entier n est représenté par $\lambda x.\lambda y.x\,(x(\ldots x(y)\ldots))$ où l'on trouve n occurrences de x après λy.

Notons maintenant t_0, t_1, t_2, ..., t_n ces lambda-termes qui servent à représenter les entiers 0, 1, 2, ... n. Ils sont tous en forme normale. On dit d'un certain terme (convenons de l'appeler Add), qu'il représente l'addition dans le lambda-calcul si, n et m étant deux entiers quelconques, $((\text{Add})\ t_n)\ t_m \rightarrow_\beta t_{n+m}$[1]. L'addition est donc lambda-définissable s'il existe un terme Add du lambda-calcul qui vérifie cette propriété. Plus généralement, la fonction numérique f, k-aire, est lambda-définissable s'il existe un terme F du lambda-calcul qui vérifie la propriété suivante pour tous entiers t_{n1}, ... t_{nk} :

$$\mathrm{F}\,t_{n1}\ldots t_{nk} \rightarrow_\beta t_{f(n1,\ldots nk)}.$$

Par exemple, le terme $\lambda n.\lambda f.\lambda u.(f)\,(((n)\,f)\,u)$ est une lambda définition de la fonction successeur. Si l'on convient d'appeler S ce terme, on peut vérifier que le successeur de 1 est 2, en prouvant, par le calcul suivant, que $(\mathrm{S}) \times t_1 \rightarrow_\beta t_2$.

$(\mathrm{S})\times t_1$ s'écrit : $(\lambda n.\lambda f.\lambda u.(f)(((n)\,f)\,u))\ (\lambda x.\lambda y.(x)\,y)$. On montre que ce terme peut être réduit à une représentation de l'entier 2 par trois β-réductions successives :

$$\begin{aligned}
&(\lambda n.\lambda f.\lambda u.(f)\ (((n)\,f)\ u))\ (\lambda x.\lambda y.(x)\,y)\\
&\rightarrow_\beta \lambda f.\lambda u.(f)\ (((\lambda x.\lambda y.(x)\,y)\,f)\ u)\\
&\rightarrow_\beta \lambda f.\lambda u.(f)\ ((\lambda y.(f)\,y)\ u)\\
&\rightarrow_\beta \lambda f.\lambda u.(f)\ ((f)\ u).
\end{aligned}$$

Or $\lambda f.\lambda u.(f)\ ((f)\ u)$ est bien une représentation de l'entier 2.[2]

1. Écrire le terme $((\text{Add})\ t_n)\ t_m$ revient à poser une opération d'addition que l'on va effectuer (l'opération $n+m$) ; la β-réduction est le processus de calcul, et t_{n+m} est le résultat.
2. Puisque les termes $\lambda f.\lambda u.(f)\ ((f)\ u)$ et $\lambda x.\lambda y.(x)\ ((x)\,y)$ sont équivalents (voir ci-dessus la n. 2, p. 68).

Lorsque Kleene travaillait sur la définition des fonctions numériques dans le lambda-calcul, il était clair que toute fonction lambda-définissable était calculable. La thèse de Church pose qu'en outre, toute fonction calculable est définissable par un lambda-terme. Et Church s'appuie sur cette thèse, comme Turing sur la thèse correspondante, pour apporter au problème de la décision une solution négative. En 1935, dans l'article qui porte sur *Un problème insoluble de la théorie élémentaire des nombres*[1], Church identifie les fonctions numériques effectivement calculables et les fonctions numériques récursives. L'équivalence entre fonctions récursives et fonctions lambda-définissables étant connue au moment où l'article est publié, Church précise (au § 7) que sa thèse pourrait aussi bien être énoncée dans les termes du lambda-calcul :

> La notion de calculabilité effective. Nous définissons maintenant la notion, qui a déjà été discutée, d'une fonction *effectivement calculable* des entiers positifs en l'identifiant avec la notion de fonction récursive des entiers positifs (ou de fonction λ-définissable des entiers positifs)[2].

Et Church ajoute en note :

> La question de la relation entre calculabilité effective et récursivité (à laquelle on propose ici de répondre en identifiant les deux notions) fut soulevée par Gödel dans une conversation avec l'auteur. La question correspondante de la relation entre calculabilité effective et λ-définissabilité avait été proposée auparavant par l'auteur, de manière indépendante.

Dans le même article, Church démontre qu'il n'existe aucune fonction récursive qui appliquée à un terme C quelconque, prend la valeur (par exemple) 1 ou 2 selon que le terme C a ou n'a pas de forme normale, c'est-à-dire, par la thèse de Church, qu'il n'existe aucune procédure effective pour déterminer si un terme est normalisable. L'article se termine par une solution négative apportée au problème de la décision, dans le cas de certains systèmes formels particuliers. C'est en 1936, dans *Une note sur l'Entscheidungsproblem*,

1. An unsolvable problem of elementary number theory, *The American Journal of Mathematics*, vol. 58, 1936, p. 345-363, reproduit dans Davis, *op. cit.*, 1965.
2. Davis, *op. cit.*, p. 100.

texte qui complète l'article de 1935, que Church prouve l'insolubilité du problème de la décision tel qu'il avait été formulé par Hilbert[1].

Les lambda-termes sont des machines

Le lambda-calcul étant ainsi compris, la correspondance avec les machines de Turing se conçoit aisément. Chaque lambda-terme t appliqué à un ou plusieurs autres termes u_1, u_2, etc., est l'équivalent d'une machine à laquelle on communiquerait une ou plusieurs données initiales ; t représente une procédure de calcul, mise en œuvre par l'opération de la β-réduction. Le résultat du calcul, s'il existe, est obtenu lorsque le terme initial $(t)u$ (ou $(t)u_1u_2...$) est réduit à un terme en forme normale. Cependant, de même que la machine de Turing n'entre pas nécessairement dans l'état final après un nombre fini d'étapes de calcul, de même, la procédure de réduction ne conduit pas nécessairement à un lambda-terme en forme normale. En revanche, le théorème de Church-Rosser garantit que le résultat du calcul est unique, s'il existe, et qu'en ce sens, la β-réduction est bien une forme de calcul[2]. De surcroît, la classe des fonctions numériques lambda-définissables est exactement celle des fonctions numériques Turing-calculables. En ce sens, le lambda-calcul peut être considéré comme un langage de programmation universel. Dans ce langage, les procédures de calcul ne prennent pas la forme d'une liste de quintuplets (comme pour les machines de Turing) ou d'une suite d'instructions (comme dans un

1. A Note on the Entscheidungsproblem, *Journal of Symbolic Logic*, vol. 1, n° 1 et vol. 1, n° 3, 1936, reproduit dans Davis, *op. cit.*
2. La β-réduction n'est pas entièrement déterministe puisqu'un terme peut comporter plusieurs redex. Dans ce cas, il y a plusieurs chemins de réduction possibles. Mais on peut prouver qu'il existe une stratégie de réduction qui conduit nécessairement à la forme normale si le terme est normalisable.

grand nombre de langages informatiques), mais celle d'un lambda-terme. Et c'est en construisant un tel terme que l'on peut, en théorie[1], programmer l'addition, le produit, le tri d'une liste d'entiers, ou n'importe quelle tâche qu'un ordinateur est capable de réaliser. Remarquons encore que dans l'article de 1935, Church pose au sujet de ces termes une question similaire au problème de l'arrêt pour les machines de Turing : demander s'il existe une procédure effective permettant de déterminer si un terme quelconque est normalisable revient en effet à poser l'équivalent du problème de l'arrêt pour la β-réduction. A maints égards, les lambda-termes peuvent donc être considérés comme des *machines à réduction*[2]. En outre, exactement comme dans l'article de Turing, c'est l'insolubilité de ce problème qui le conduit à une solution négative de l'*Entscheidungsproblem*.

Ce qui fait la singularité des termes du lambda-calcul est leur syntaxe, plus abstraite que celle des machines de Turing puisque l'on n'y retrouve pas l'image d'un mécanisme capable de lire, d'écrire, d'effacer, de se déplacer ou de changer d'état. Il n'en reste pas moins que ces termes peuvent être considérés comme des machines informatiques, au même titre que les programmes écrits dans des langages comme Fortran, Lisp ou Pascal, les machines de Turing elles-mêmes ou les ordinateurs. *Abstraits* dans la mesure où leur définition est purement logique, et parce qu'ils ne se présentent pas sous la forme d'un mécanisme ou d'un dispositif physique, les lambda-termes sont pourtant de véritables machines au sens où un tel dispositif pourrait être réalisé, capable d'effectuer la β-réduction, et donc d'évaluer ainsi n'importe quelle fonction calculable.

1. En pratique, la programmation dans le lambda-calcul pur tel qu'il vient d'être présenté est aussi peu aisée que dans le langage-machine d'un ordinateur. En revanche, il existe des langages évolués qui sont construits à partir des principes du lambda-calcul.
2. L'expression « machines à réduction » est généralement utilisée en parlant de certains formalismes de *lambda-calcul typé*, dans lesquels on restreint les possibilités de construction syntaxique des termes afin de pouvoir démontrer plusieurs propriétés particulièrement intéressantes du calcul (par exemple la normalisation forte, qui garantit que toutes les réductions de termes conduisent à un résultat – c'est-à-dire à un terme normal – après un nombre fini d'étapes). On reviendra au chapitre VIII sur la notion de lambda-calcul *typé*.

Le lambda-calcul se présente donc comme un nouvel exemple d'expression universelle possible pour les procédures de calcul, au même titre qu'un langage de programmation, les machines de Turing, ou certains dispositifs physiquement réalisés. La liste des formalismes équivalents à une machine universelle n'en est pas close pour autant : la théorie de la calculabilité décline toutes les variantes syntaxiques d'une caractérisation des fonctions calculables, et l'histoire des calculateurs universels permettrait de comparer entre eux les multiples procédés de calcul dans leur réalisation physique sous la forme de machines réelles. Un examen plus complet montrerait donc l'équivalence entre les fonctions récursives, les machines à registres, les systèmes de Post, plusieurs variantes des machines de Turing, d'innombrables langages de programmation, l'engin analytique de Babbage, des machines physiques non séquentielles[1], ainsi que d'autres dispositifs logiciels ou matériels, abstraits ou concrets. Cet examen permettrait de mesurer avec une précision accrue les conditions minimales qui font de chacun de ces calculateurs une machine universelle. Et par une suite de démonstrations d'équivalence, se trouverait ainsi confirmée, au-delà des variations de ces conditions minimales selon le style ou la syntaxe propre à chaque machine, l'invariance de la classe des fonctions calculables.

On pourrait aussi compléter cet examen des calculateurs universels par la théorie de dispositifs plus faibles que les machines de Turing, ne permettant de calculer qu'une classe plus réduite de fonctions, et dont les capacités peuvent être mesurées par les langages formels qu'ils sont susceptibles de reconnaître ou d'engendrer. La théorie des automates construit en effet une hiérarchie de machines aux capacités précisément mesurables, des automates finis aux machines de Turing, en passant par les automates à piles et les automates linéairement bornés. Notre propos n'est cependant ni d'exposer la théorie de ces machines ni d'en faire l'his-

1. Contrairement aux machines de von Neumann, elles sont capables d'effectuer plusieurs opérations de calcul en parallèle.

toire[1], mais de comprendre ce qui en fait des objets de la logique. Or à cet égard, les machines informatiques précédemment considérées suffisent à dégager plusieurs conclusions.

En 1936, la machine de Turing pouvait bien apparaître comme une singularité contingente, propre à l'une des solutions possibles de l'*Entscheidungsproblem*, et le modèle *mécanique* de la calculabilité comme une curiosité[2]. Mais les développements ultérieurs de la logique et de l'informatique montrent plutôt que les formalismes élaborés de manière indépendante par d'autres logiciens pour la résolution du même problème peuvent également recevoir une interprétation en termes de machines, interprétation qui est particulièrement éclairante pour la théorie de la calculabilité, et pour la compréhension des calculateurs universels. Le lambda-calcul en est un exemple frappant et tout à fait significatif. En sorte que les machines abstraites de Turing, comme un moyen terme entre deux extrêmes, permettent de construire une correspondance entre la logique et l'informatique. Examiner le sens de cette correspondance suppose que l'on montre comment le moyen terme permet le passage d'un extrême à l'autre, de problèmes portant sur la vérité ou la démonstration à des questions relatives aux capacités des machines et aux programmes. Dans ce qui précède, ce parcours nous a effectivement conduit du programme de Hilbert, purement logique, à la définition des machines de Turing, d'une solution de l'*Entscheidungsproblem* aux calculateurs universels physiquement réalisés, des ordinateurs matériels aux programmes logiciels, jusqu'au lambda-calcul, considéré à la fois comme un langage de programmation et un formalisme élaboré pour la solution de problèmes

1. Sur ces questions, on pourra consulter John Hopcroft et Jeffrey Ullman, *Introduction to automata theory, languages and computation*, Reading, Mass., Addison-Wesley, 1979 ; Marvin L. Minsky, *Computation. Finite and Infinite Machines*, Englewood Cliffs, NJ, Prentice-Hall, 1967 ; et Mosconi, *op. cit.*
2. Comme le souligne Jean Mosconi, « les notions de calculabilité *mécanique*, procédure *mécanique* n'ont joué aucun rôle directeur dans l'élaboration de la théorie de la récursivité ; l'absence presque totale de telles expressions dans les textes de l'époque frappe aujourd'hui d'autant plus qu'elles sont largement répandues dans la littérature actuelle (en partie certes sous l'influence de l'informatique) » (Mosconi, *op. cit.*, p. 17).

logiques ou pour la construction de machines qui effectuent les calculs par β-réduction. Ce parcours conduit aussi à l'idée que d'un point de vue théorique, les machines informatiques peuvent être conçues comme des constructions de signes caractérisées par une syntaxe, et qu'à ce titre, elles ne sont pas essentiellement différentes des systèmes formels de la logique. Les machines de Turing, les termes du lambda-calcul et les programmes informatiques sont en effet tous des systèmes de signes syntaxiquement définis, comparables à cet égard aux différents systèmes formels élaborés en logique. Or, si la logique, depuis Hilbert, procède par construction de systèmes formels déductifs et si, d'autre part, les machines informatiques se réduisent à des constructions syntaxiques, on comprend mieux que la logique puisse prendre ces machines pour objet, et devenir un instrument d'analyse des programmes, des machines et des algorithmes.

Toutefois, si les analyses qui précèdent mettent en évidence l'importance de la machine en théorie de la calculabilité, elles n'en font pas encore pour autant un objet de la logique dans son ensemble. Elles indiquent bien la possibilité de mettre en rapport les systèmes logiques et les machines informatiques, théoriquement réductibles à des constructions formelles régies par une syntaxe, mais ne permettent pas de dépasser l'opposition entre les systèmes conçus pour la formalisation des preuves et ceux qui modélisent une capacité de calcul. L'idée de machine apparaîtrait beaucoup plus clairement comme un objet logique si cette différence entre une démonstration et un calcul pouvait être réduite. Dans ce qui précède, la possibilité d'un tel rapprochement a bien été suggérée par la manière dont Turing résout (négativement) le problème de la décision puisque le fonctionnement d'une machine qui effectue un calcul y est représenté par une formule de logique[1] ; mais ce rapport ressemble plus à un ingénieux artifice de démonstration qu'à la mise en évidence d'une correspondance profonde entre les

1. Voir ci-dessus, chap. II, p. 43-44.

démonstrations formelles d'un système de logique et les calculs d'une machine informatique.

On esquisse ainsi le chemin qu'il reste à parcourir pour comprendre l'idée de machine comme un objet logique : montrer à quelles conditions il est possible de concevoir un calcul comme une preuve, et certaines démonstrations comme l'expression d'un programme. Ce pas ne pourra cependant pas être franchi sans une clarification préalable de la notion polysémique de calcul, ce qui ne suppose pas une simple définition du terme, mais la compréhension d'une problématique, celle du calcul logique et de ses rapports à la démonstration comme à l'idée de machine.

V

L'IDÉE DE CALCUL LOGIQUE LEIBNIZ, BOOLE, FREGE

CALCUL ARITHMÉTIQUE ET RAISONNEMENT LOGIQUE

L'idée de machine fut introduite en logique par une recherche portant sur les procédures de calcul et sur les fonctions calculables ; elle est donc, à cet égard, directement liée à celle de calcul. Cependant, si la notion de machine apparaît en logique dans les années trente, au moment où se constitue la théorie de la calculabilité, il s'en faut de beaucoup que l'idée de calcul logique soit aussi récente. On trouve chez Leibniz le projet d'un *calculus ratiocinator*, et chez Boole la conception d'un calcul algébrique appliqué à la résolution de problèmes logiques. Cette expression s'entend, ici et là, en des sens très différents, de sorte que si l'idée de machine suppose bien celle de calcul, elle ne résulte pas nécessairement de n'importe quelle conception du calcul logique. Bien plus, il est loin d'être évident que la théorie de la calculabilité, lorsqu'elle étudie les machines de Turing, les termes du lambda-calcul ou les fonctions récursives, puisse être située simplement dans le prolongement d'une tradition séculaire qui conçoit la logique comme visant la réalisation d'une forme de calcul.

De notre point de vue, l'enjeu du problème est le suivant : si l'idée de machine est introduite par la théorie de la calculabilité, il reste à montrer quelle est la place de cette théorie au sein de la logique et quel est son rapport à l'idée générale de calcul logique, afin de com-

prendre pourquoi la machine a pu devenir l'une des notions centrales de la logique. La question peut être posée dans une perspective synchronique ou diachronique. Premièrement : quelle est la place de la théorie de la calculabilité au sein de la logique contemporaine ? quel est, notamment, son rapport à la théorie de la démonstration ? En examinant ces points, on montrera pourquoi l'idée de machine, dans la sphère de la logique actuelle, n'est ni marginale, ni restreinte à un sous-domaine particulier, puisqu'elle touche autant à la théorie de la démonstration, voire à la sémantique, qu'à la théorie de la calculabilité. Deuxièmement, du point de vue diachronique, on demande quel est le rapport entre l'idéal d'un calcul logique tel qu'il apparaît au cours de l'histoire, fût-ce sous des formes différentes, en logique et en philosophie, et, d'autre part, les questions qui relèvent de la théorie de la calculabilité actuelle. S'agit-il, ici et là, de la même notion de calcul ? Le fait que dans les années trente Church, Gödel, Turing ou Kleene aient surtout travaillé sur les fonctions calculables *numériques* laisse en effet penser que l'on n'entend plus le mot « calcul » dans le même sens. Ou encore, on demande pourquoi les recherches logiques portant sur le calcul ne conduisent qu'en 1936 à l'introduction, puis à la généralisation d'une idée de machine. Les problèmes posés de ces deux points de vue, synchronique et diachronique, relèvent tous en définitive de la même interrogation : comment la notion de machine, initialement introduite dans le contexte particulier d'une théorie des calculs possibles, peut-elle devenir objet de la logique en général, et quelle conception de la logique faut-il former pour faire des machines un objet de cette discipline ? La question appelle une réflexion sur le calcul et sur sa place, tant dans la logique actuelle que dans celle du passé.

Au sens littéral, élémentaire et premier, le calcul est une opération arithmétique sur des nombres, qui furent jadis représentés par de petits cailloux *(calculi)*. Sa principale caractéristique, on l'a vu, est de pouvoir être effectué en suivant un ensemble de règles rigoureusement déterminées, selon un ordre non moins parfaitement défini. Si une erreur de calcul est possible, elle n'est pas imputable au défaut des règles, mais à l'inattention du calculateur. A cet égard, le calcul se dis-

tingue du raisonnement, suite d'idées ou d'énoncés dont l'enchaînement n'obéit bien souvent qu'à des règles implicites et des intuitions de signification, moins précisément définies, ou dont l'application est rendue problématique par les ambiguïtés ou l'usage variable du langage que nous parlons. En outre, le cours rapide de la pensée répugne à l'application de règles pesantes et contraignantes, qui embarrassent l'esprit lorsqu'il procède par analogies, par allusions à des présupposés, ou en laissant inexprimées nombre de connaissances auxquelles il se réfère pourtant implicitement. Plusieurs prémisses étant posées, la possibilité d'en déduire par le raisonnement des conclusions contradictoires ne découle donc pas nécessairement de l'inadvertance du raisonneur. Mais de ce fait, la correction d'un raisonnement est souvent problématique, alors que celle d'un calcul ne l'est presque jamais. Lorsqu'elle se donne pour tâche l'analyse des inférences, la logique s'efforce de réduire cet écart, et de rendre explicites et parfaitement déterminées les règles du raisonnement correct. En ce sens, l'idéal du logicien se conçoit assez naturellement comme celui d'un *calcul*, applicable aux énoncés d'une pensée qui enchaîne successivement prémisses et conclusions. Le problème de la logique ainsi conçue est donc celui du hiatus qui existe entre cet idéal hypothétique et la manière dont les raisonnements se présentent habituellement dans la langue ou sous la plume de l'*homo ratiocinator*. L'usage courant n'offre en effet que très exceptionnellement l'exemple d'une suite d'énoncés respectant strictement les règles de la logique, comme un calcul numérique celles de l'arithmétique. Entaché ou enrichi d'ambiguïtés, mis au service du mensonge ou de fins déterminées par nos affects, incertain dans son usage et en perpétuelle évolution, le langage que nous parlons et dans lequel nous raisonnons ne se plie guère aux impératifs d'une logique idéalement référée au calcul et visant par là l'expression ou la transmission de la vérité. D'où l'exigence de sa rectification, celle d'un inventaire, pour analyse, des parties et des formes du discours, ou celle, plus radicale, de l'invention d'une langue expurgée de toute ambiguïté, explicite et logique de part en part, soumise aux lois d'un calcul infaillible et permettant l'application de règles purement formelles. D'où, également, les rapports problématiques entre

un raisonnement tenu dans la langue usuelle, et sa version rectifiée ou transcrite : s'agit-il encore du même raisonnement ? où se trouve la logique ? dans l'usage et la langue des locuteurs ou dans une langue artificielle et soigneusement élaborée ?

Cette analogie avec l'arithmétique donne à l'idée de calcul logique un premier sens, élémentaire, et qui demeure insuffisant pour au moins deux raisons. Premièrement, parce qu'on ne voit pas encore clairement ce que cette analogie pourrait faire correspondre, dans l'ordre logique, au *résultat* recherché par une opération de calcul numérique. Deuxièmement et surtout, parce qu'elle présente une conception de la logique qui semble la réduire à une technique de contrôle du raisonnement. Or le philosophe n'attend pas seulement de la logique une étude de la correction des inférences, mais plus généralement une analyse des conditions de possibilité de notre connaissance, en tant qu'elle dépend d'une forme d'adéquation entre le langage et la réalité. Et il n'est pas évident *a priori* qu'une telle exigence puisse être satisfaite par un calcul logique. Les deux questions ne sont d'ailleurs pas sans rapport puisque l'on demande, dans les deux cas, de préciser ce que la logique peut attendre d'un calcul. On ne pourra pas mieux y répondre qu'en se référant aux philosophes et logiciens qui ont explicitement conçu le projet d'un calcul logique, même s'il est conçu très différemment selon qu'il émane de Leibniz, de Boole, de Frege, ou de Hilbert. En considérant quelques-uns des sens qui furent donnés au calcul logique, on pourra mieux déterminer comment la théorie de la calculabilité s'inscrit dans l'histoire de cette idée, et comment le rapport entre machine et calcul logique doit être conçu.

LE *CALCULUS RATIOCINATOR* DE LEIBNIZ

Chez Leibniz, le *calculus ratiocinator* est indissociable d'un projet logique beaucoup plus vaste, qui se présente à la fois sous les espèces d'une science générale présidant à l'ordre encyclopédique

des connaissances, d'une écriture ou même d'une langue universelle dite « caractéristique », et permettant l'expression logique de ces connaissances, et enfin, d'une analyse des idées, des vérités et des démonstrations, visant leur réduction à des pensées et des propositions primitives. Loin d'être un simple procédé de contrôle des raisonnements, ce calcul logique engage la théorie leibnizienne de la connaissance dans son ensemble, et par conséquent de manière indirecte, le système philosophique tout entier. Selon Leibniz, il doit pourtant remplir l'office de tout procédé de calcul, à savoir la production d'un résultat en fonction de données initiales posées comme une question dans l'attente d'une réponse. Et il doit ainsi substituer à l'incertitude des raisonnements construits dans la langue usuelle pour la solution d'un problème particulier la détermination aveugle ou symbolique[1] d'un calcul infaillible, capable de répondre à la question posée, ou d'évaluer avec exactitude la probabilité de telle ou telle réponse possible :

> ... j'ai commencé à avoir certaines vues toutes nouvelles, pour réduire tous les raisonnements humains à une espèce de calcul, qui servirait à découvrir la vérité, autant qu'il se peut faire *ex datis* ou par ce qui est donné ou connu, et lorsque les connaissances données ne suffisent pas à résoudre la question proposée, cette méthode servirait comme dans les Mathématiques, à approcher autant qu'on le peut sur le donné et à déterminer exactement ce qui est le plus probable[2].

Leibniz ajoute qu'il serait possible « de raisonner en calculant, de sorte qu'au lieu de disputer, on pourrait dire : comptons ». Le

1. L'importance de la pensée « aveugle ou symbolique » est explicitement soulignée par Leibniz : « Soit ma pensée d'un chiliogone, ou polygone de mille côtés égaux ; je ne considère pas toujours en elle la nature du côté, ni de l'égalité, ni du nombre mille ou cube de dix ; mais ce sont ces mots-là (...) que j'emploie mentalement, au lieu des idées correspondantes ; je les emploie parce que je me rappelle posséder leur signification, et que, de plus, je juge que leur conception explicite n'est pas à ce moment nécessaire. Cette pensée, j'ai coutume de l'appeler aveugle, ou encore symbolique ; c'est celle dont nous usons en algèbre et en arithmétique, et même presque en toutes choses » (*Meditationes de Cognitione, Veritate et Ideis, in* Gottfried Wilhelm Leibniz, *Die philosophischen Schriften von G. W. Leibniz,* Éd. C. I. Gerhardt, 7 vol., 1875-1890, vol. IV (on adoptera ci-dessous les abréviations *G.*I..., *G.* VII pour les volumes de cette édition), p. 423, trad. franç., Leibniz, *Œuvres,* éd. par L. Prenant, Paris, Aubier-Montaigne, 1972, p. 152-153.
2. *G.* VII, p. 25. Ce passage est extrait d'une lettre inachevée, écrite en français et destinée au duc de Hanovre (*G.* VII, p. 25-27).

calcul logique leibnizien prend donc la forme d'une procédure opératoire destinée, comme dans le cas paradigmatique du calcul arithmétique, à produire un résultat. Mais en logique, le résultat obtenu est rarement une valeur numérique. Il s'agit plutôt, par exemple, de mettre fin aux controverses en déterminant par le calcul ce qui est logiquement déductible de prémisses posées comme admises ou de connaissances acquises par ailleurs. Le calcul logique montre laquelle des deux thèses opposées est correcte, ou révèle quelles sont les connaissances qui manquent pour l'obtention d'un résultat décisoire. On pourra donc reconnaître dans le texte suivant une sorte de problème de la décision (avec cette différence essentielle, cependant, qu'il n'est pas formulé par Leibniz d'un point de vue métathéorique, au sujet d'un langage objet) :

> Et il se trouverait que les erreurs du raisonnement ne seraient que des erreurs de calcul qu'on découvrirait par des épreuves comme dans l'Arithmétique.
>
> Les hommes trouveraient par là un juge des controverses véritablement infaillible. Car ils pourraient toujours connaître s'il est possible de décider la question par le moyen des connaissances qui leur sont déjà données, et lorsqu'il n'est pas possible de se satisfaire entièrement, ils pourront toujours déterminer ce qui est le plus vraisemblable. Comme dans l'arithmétique on peut toujours juger s'il est possible ou non de deviner exactement le nombre que quelque personne a dans la pensée, sur ce qu'elle nous en a dit, et souvent on peut dire : ce doit être l'un de deux ou de trois etc. tels nombres, et prescrire des bornes exactes à la vérité inconnue. En tout cas il importe au moins de savoir que ce qu'on demande n'est pas trouvable par les moyens que nous avons[1].

Initialement conçu par analogie avec les opérations paradigmatiques de l'arithmétique, le calcul logique est appliqué à des raisonnements quelconques. Or si cette généralisation est possible, c'est également parce que le *calculus* est indissociable du projet leibnizien de caractéristique universelle, c'est-à-dire d'une écriture conceptuelle, ou idéographie, dans laquelle la composition des signes répond exactement à la composition des idées ou des choses :

> Cette sorte de calcul général donnerait en même temps une espèce d'écriture universelle qui aurait l'avantage de celle des Chinois, parce que chacun l'entendrait[2] dans sa langue, mais qui surpasserait infiniment la Chinoise en ce

1. *Ibid.*, p. 26.
2. « L'entendoit » dans le texte publié par Gerhardt.

qu'on la pourrait apprendre en peu de semaines, ayant les caractères bien liés selon l'ordre et la connexion des choses, au lieu que les Chinois ayant une infinité de caractères selon la variété des choses, il faut la vie d'un homme pour apprendre assez leur écriture.

Cette écriture ou langue (si on rendait les caractères énonçables) pourrait être bientôt reçue dans le monde, parce qu'elle pourrait être apprise en peu de semaines, et donnerait moyen de communiquer partout. Ce qui serait de grande importance pour la propagation de la foi, et pour l'instruction des peuples éloignés[1].

Le *calculus ratiocinator* de Leibniz est donc porté par la caractéristique universelle, dont la réalisation suppose une analyse systématique de toutes les idées ou notions en usage dans les différents domaines de la connaissance. En effet, dans l'écriture rationnelle à laquelle pense Leibniz, la composition des signes, ou caractères, ne peut répondre à l'ordre des choses ou des idées si l'on ne connaît, par le moyen de l'analyse, la connexion des pensées et des vérités. Le logicien-philosophe doit donc s'efforcer de réduire les notions complexes à des compositions réglées de notions plus simples, et poursuivre ainsi l'analyse, afin de pouvoir étudier ensuite, à l'aide de l'art combinatoire, les possibilités de combinaison des pensées les plus primitives, et retrouver, par synthèse, les notions composées à partir de ce que Leibniz appelle «l'alphabet des pensées humaines». D'où l'intérêt porté aux définitions, dont Leibniz constitue plusieurs collections[2] :

Pour arriver donc à cette Écriture ou Caractéristique qui contient un calcul si surprenant, il faut chercher des définitions exactes des notions. Car les paroles que nous avons, étant assez obscures et ne nous donnant souvent que des notions confuses, on est obligé de substituer d'autres caractères dont la notion soit précise et déterminée, et les définitions ne sont qu'une expression distincte de l'idée de la chose.

Et comme j'ai étudié avec soin non seulement l'Histoire et les Mathématiques, mais aussi la Théologie naturelle, la Jurisprudence et la Philosophie, j'ai fort avancé ce dessein, et je m'ai fait quantité de définitions. Par exemple la définition de la justice chez moi est : la Justice est la charité du sage, ou une charité conforme à la sagesse. La Charité n'est autre chose que la bienveil-

1. *Ibid.*, p. 25-26.
2. Dans Couturat, *Opuscules et fragments inédits de Leibniz,* Hildesheim, Georg Olms, 1903, sont publiées plusieurs listes de définitions recueillies par Leibniz. L'un des opuscules, intitulé *Table de définitions,* s'étend sur plus de 70 pages dans l'édition de Couturat.

lance générale ; la Sagesse c'est la science de la félicité, la Félicité est l'état de joie durable, la Joie c'est un sentiment de perfection, la Perfection c'est le degré de réalité.

Je prétends de donner des définitions semblables de toutes les passions, vertus, vices et actions humaines, autant qu'il en est besoin. Et par ce moyen on pourra parler et raisonner avec l'exactitude. Et comme les nouveaux caractères envelopperont toujours les définitions des choses, il s'ensuit qu'ils nous donneront moyen de raisonner en calculant, comme j'avais dit ci-dessus[1].

Le calcul suppose donc une analyse de toutes nos connaissances. On voit par là qu'il n'est pas dissociable du projet leibnizien d'encyclopédie, mise en ordre générale de l'ensemble du savoir :

La Caractéristique que je me propose ne demande qu'une espèce d'Encyclopédie nouvelle. L'Encyclopédie est un corps, où les connaissances humaines les plus importantes sont rangées par ordre. Cette Encyclopédie étant faite selon l'ordre que je me propose, la caractéristique serait quasi toute faite (...)[2].

Sans définir plus avant l'ordre logique du savoir tel qu'il est conçu par Leibniz, on peut comprendre son rapport à l'analyse des notions et des connaissances puisque cette analyse mettrait en évidence leur composition à partir de cet ensemble d'idées simples qui constitue « l'alphabet des pensées humaines »[3]. Or dans la langue universelle, les caractères sont liés selon la connexion des idées, la combinaison des caractères répondant à la composition des notions. D'où la possibilité d'un calcul opéré sur les signes, ayant valeur d'un calcul sur les notions et sur les connaissances, pourvu que

1. *G.* VII, p. 26-27.
2. Remarque écrite sur une fiche de Leibniz, publiée dans *G.* VII, p. 40.
3. L'expression se trouve, par exemple, dans le fragment intitulé *De arte inveniendi in genere, in* Couturat, *op. cit.*, p. 165, ou dans *G.* VII, p. 185. Leibniz n'affirme pas pour autant que notre pouvoir d'analyse des notions puisse aller jusqu'au degré ultime que constitue cet alphabet : « La capacité humaine d'analyse des concepts ne semble pas être suffisante pour que nous puissions parvenir jusqu'aux notions primitives, ou jusqu'à celles qui se conçoivent d'elles-mêmes » (Leibniz, *Introductio ad encyclopædiam arcanam, in* Couturat, *op. cit.*, p. 514). Cependant, pour Leibniz, cette limite ne rend pas irréalisable la caractéristique universelle : « (...) quoique cette langue dépende de la vraie philosophie, elle ne dépend pas de sa perfection (...). En attendant, elle sera d'un secours merveilleux et pour se servir de ce que nous savons, et pour voir ce qui nous manque, et pour inventer les moyens d'y arriver, mais surtout pour exterminer les controverses dans les matières qui dépendent du raisonnement. Car alors raisonner et calculer sera la même chose » (Leibniz, *in* Couturat, *op. cit.*, p. 27-28).

l'écriture rationnelle dans laquelle il s'exprime soit, en ce sens, une caractéristique « réelle »[1].

Chez Leibniz, en outre, cette théorie des signes et de la mise en ordre logique des connaissances est prolongée par une philosophie de la pensée qui fait du raisonnement humain lui-même un calcul sur des signes : « Omnis humana ratiocinatio signis quibusdam sive characteribus perficitur. »[2] Cela ne signifie pas que le raisonnement lui-même ne soit rien de plus qu'une combinaison de signes, mais seulement qu'il ne peut s'accomplir sans le support des caractères. « Je suis persuadé que les Ames et les Esprits créés (...) ne sauraient raisonner sans caractères. »[3] Leibniz écrit encore :

B. — Eh quoi, on peut bien penser sans mots ?
A. — Mais pas sans recourir à d'autres signes ! Essayez, je vous prie, d'exécuter un calcul arithmétique quel qu'il soit sans l'aide de signes numériques.
B. — Vous me troublez. Je n'imaginais pas que les signes ou les caractères étaient à ce point nécessaires pour raisonner (...).
(...)
A. — Mieux, en l'absence de caractères, nous ne penserions jamais rien distinctement, nous ne ferions aucun raisonnement[4] !

On remarquera que dans tous ces textes portant sur le *calculus ratiocinator* et sur la pensée par signes, il n'est pas directement question de machines. La logique de Leibniz n'en méritait pas moins l'examen qui précède. Premièrement, parce qu'elle élabore un calcul logique dont la possible mécanisation, si elle n'est certes pas

1. « Car bien que les caractères soient arbitraires, leur usage et leur connexion a pourtant quelque chose qui n'est pas arbitraire, à savoir une certaine proportion entre les caractères et les choses, et les relations mutuelles des divers caractères exprimant les mêmes choses. Et cette proportion ou relation est le fondement de la vérité » (*Dialogus*, in *G.* VII, p. 192).
2. *G.* VII, p. 204. Sur cette question, et sur les rapports entre syntaxe et calcul chez Leibniz, cf. Jean-Pierre Séris, *Langages et machines à l'âge classique*, Paris, Hachette, 1995, qui traduit : « Toute l'activité raisonnante des hommes s'accomplit par le moyen de certains signes ou caractères » (p. 210). Voir aussi le passage extrait des *Méditations sur la connaissance, la vérité et les idées* cité ci-dessus dans la n. 1, p. 83.
3. *Nouveaux essais*, II, XXI, § 73, in *G.* V, p. 197.
4. *G.* VII, p. 191 ; trad. de Séris, *op. cit.*, p. 211.

évoquée par Leibniz, ne serait en rien contraire à l'esprit de la philosophie leibnizienne. Pour celui qui fut l'inventeur de l'une des premières machines à calculer, en effet, « la pensée peut avoir des instruments mécaniques sans rien perdre de ses prérogatives »[1]. Il serait, en revanche, beaucoup plus contestable de vouloir prolonger la pensée de Leibniz jusqu'à l'idée qu'une certaine notion de machine pourrait être *objet* de l'analyse logique, à moins d'attribuer au philosophe de Hanovre une conception de la machine qui lui reste étrangère, même lorsqu'il parle, dans d'autres textes, et d'autres contextes, de l'*automaton spirituale*[2].

Ce pas, qui ne fut franchi qu'à partir de 1936, exigeait vraisemblablement une rupture plus profonde avec la grammaire des langues que nous parlons, un point de vue métalinguistique introduit par Post et Hilbert sur l'étude de la syntaxe des langues logiques, et l'idée, due à Gödel, d'une réduction des questions métathéoriques sur les systèmes formels à des problèmes arithmétiques. A cet égard, les critiques bien connues de la notion de précurseur ne manqueraient pas d'atteindre celui qui se risquerait à user sans discernement de ce titre au sujet de Leibniz. Des logiciens aussi éminents que Frege ou Gödel se sont souvent référés à Leibniz, et l'on ne peut manquer d'admirer la lumière, remarquée par maints commentateurs contemporains, qu'apporte la pensée de Leibniz sur les sciences, les techniques, et certaines conceptions philosophiques de notre époque[3]. Ces rapprochements ne doivent cependant pas faire oublier que Leibniz est à plusieurs égards un penseur préfrégéen, et préhilbertien. La question de la continuité posée dans toute son ampleur dépasse largement le cadre du présent travail. Dans ce qui suit, on s'efforcera seulement de souligner les principales étapes qui, après Leibniz et l'invention d'un

1. L'expression est de Jean-Pierre Séris, *op. cit.*, p. 222.
2. Sur ce point aussi, on pourra se reporter aux analyses de Jean-Pierre Séris, et aux textes cités dans Séris, *op. cit.*
3. On ne peut contester, par exemple, qu'une comparaison entre la philosophie leibnizienne et la conception dite « computationnelle » de l'esprit puisse être éclairante, à la fois par les similitudes et par les différences qu'elle révèle.

calculus ratiocinator, rendront possible, bien plus tard, l'introduction en logique d'une idée de machine[1]. Si la logique leibnizienne a retenu notre attention, c'est donc aussi parce que nous la considérerons comme une référence, pour comparaison avec quelques autres conceptions ultérieures de ce qu'est un calcul logique. De ce point de vue, on retiendra surtout les caractéristiques suivantes du calcul leibnizien :

1 / Il doit prendre la forme d'un véritable *procédé opératoire* permettant la *production de résultats*, conclusions de raisonnements effectués sur ces « pensées aveugles » que sont les caractères de la langue universelle : « Le calcul ou l'opération consiste dans la production de relations, faite par le moyen de transformations des formules selon certaines lois prescrites. »[2] A ce titre, le calcul logique est bien une généralisation des opérations arithmétiques, et algébriques.

2 / Il n'est pas dissociable de la caractéristique universelle, et il engage par là une mise en ordre logique de toutes les connaissances humaines. Le calcul logique de Leibniz est en rapport avec un projet d'encyclopédie.

3 / La logique prend donc, comme le *calculus ratiocinator* lui-même, une dimension métaphysique puisque l'analyse des notions, des vérités et des preuves doit révéler l'ordre et la connexion des choses. La logique leibnizienne enveloppe le projet d'une grammaire rationnelle et philosophique.

4 / Enfin, s'il appartient à une théorie de l'ordre des connaissances, il répond également à une philosophie de la pensée, selon laquelle le raisonnement ne se passe jamais entièrement de l'usage des signes dans un langage. Et à ce titre, il sert également de modèle à ce que nous appellerions volontiers une psychologie cognitive du raisonnement.

1. On pourra lire les arguments discontinuistes de Claude Imbert dans *Phénoménologies et langues formulaires*, Paris, PUF, 1992.
2. *G.* VII, p. 206.

LE CALCUL LOGIQUE SELON GEORGE BOOLE

Le modèle le plus élémentaire du calcul est l'opération arithmétique simple. Mais dans la seconde moitié du XVII[e] siècle, les mathématiciens connaissent depuis longtemps un calcul sur des signes qui ne sont pas des nombres. Lorsqu'il évoque son projet de *calculus ratiocinator*, la référence mathématique de Leibniz n'est donc pas seulement l'arithmétique, mais également l'algèbre :

> Mais pour rendre [la science générale] plus facile et pour ainsi dire sensible, je prétends de me servir de la caractéristique dont je vous ai parlé quelques fois, et dont l'Algèbre et l'Arithmétique ne sont que des échantillons. (...) on a oublié le principal qui est que les caractères de cette écriture doivent servir à l'invention et au jugement, comme dans l'Algèbre et dans l'Arithmétique[1].
>
> (...) cette même écriture serait une espèce d'Algèbre générale et donnerait moyen de raisonner en calculant (...)[2].

Le calcul logique apparaît ici comme une généralisation des cas paradigmatiques que sont pour Leibniz à la fois l'arithmétique et l'algèbre. Au milieu du XIX[e] siècle, lorsque George Boole élabore une autre forme de calcul logique, la référence est pour lui également l'algèbre. Mais si le calcul algébrique sert encore de modèle, c'est en un sens très différent et à certains égards moins ambitieux puisque la logique ne se présente pas chez Boole comme une science générale ayant pour objet l'ordre encyclopédique des connaissances, ni comme une analyse des idées et notions qui permette d'en apercevoir la composition à partir d'une combinaison de pensées plus simples (un alphabet des pensées humaines, selon l'expression de Leibniz). Il s'agit plutôt d'étudier les propriétés d'opérations pouvant être effectuées sur des objets quelconques, afin d'en dégager les lois ; ce qui revient, selon Boole, à mener une investigation des *lois de la pensée.*

1. Lettre à Gallois de décembre 1678, in *G.* VII, p. 22-23.
2. Lettre au duc de Hanovre, in *G.* VII, p. 26.

L'ouvrage publié en 1854 sous ce titre *(An Investigation of The Laws of Thought)* a pour but « d'étudier les lois fondamentales des opérations de l'esprit par lesquelles s'effectue le raisonnement »[1]. Pour Boole, la recherche d'une méthode générale en logique va de pair avec l'étude des lois de la pensée car l'objet de la logique n'est pas seulement celui d'un calcul des inférences valides. Son intérêt, de même que celui d'une théorie des probabilités, « découle de la lumière qu'elles projettent sur nos facultés intellectuelles »[2]. Il reste alors à comprendre ce que l'algèbre peut apporter à cette double recherche d'une méthode logique et des lois de la pensée. Or selon Boole, les lois du calcul algébrique sont précisément équivalentes à celles de l'esprit dans son activité de raisonnement :

> Il n'existe pas seulement une étroite analogie entre les opérations de l'esprit dans le raisonnement en général et celles qu'il mène dans cette science particulière qu'est l'algèbre, mais, dans une très large mesure, une équivalence exacte des lois auxquelles obéissent les deux classes d'opérations[3].

Pour Boole, la méthode logique n'est donc rien d'autre qu'une application du calcul algébrique à des questions qui ne relèvent pas des objets mathématiques usuels. Si la logique utilise des méthodes mathématiques, c'est pour résoudre des problèmes qui ne concernent pas nécessairement le nombre ou la quantité mais, plus généralement, les relations entre les choses (comme par exemple dans l'énoncé « tous les hommes sont mortels ») ou les relations entre les propositions (« si le soleil connaît une éclipse totale, les étoiles seront visibles »). Boole écrit :

> Que la logique comme science soit susceptible d'applications très diverses est un point admis ; mais il est tout aussi assuré que ses formes et procédures ultimes sont mathématiques. (...) Il n'est pas de l'essence des mathématiques de s'occuper des idées de nombre et de quantité[4].

1. George Boole, *An Investigation of The Laws of Thought, on which are founded the mathematical theories of logic and probabilities,* 1854 ; trad. franç., *Les lois de la pensée,* Paris, Vrin, 1992, p. 21.
2. *Ibid.,* p. 22.
3. *Ibid.,* p. 25.
4. *Ibid.,* p. 31.

Mais comment les symboles et les opérations de l'algèbre peuvent-ils apporter une solution aux questions qui relèvent de la logique ? La question se pose notamment parce qu'à l'époque où écrit Boole, on ne conçoit guère qu'une méthode mathématique puisse être appliquée à une discipline qui suscite beaucoup plus l'intérêt des philosophes que celui des mathématiciens. Plusieurs raffinements récents de l'analyse logique dus notamment à William Hamilton et Auguste de Morgan, comme l'usage systématique d'une quantification sur les prédicats[1] et la référence à un univers du discours, permettent cependant à Boole de représenter les propositions sous une forme équationnelle, et d'utiliser ainsi les moyens de l'algèbre pour le calcul des conclusions déductibles de quelques prémisses.

Considérons un exemple très simple. Boole raisonne en termes de classes. 1 représente l'univers du discours, et 0 la classe vide. Si l'on note + la somme logique, et × le produit logique, la proposition « aucun dieu n'est mortel » peut être représentée par l'équation $x \times y = 0$ (interprétée de la manière suivante : la classe des individus qui sont *à la fois* x, c'est-à-dire dieux, et y, c'est-à-dire mortels, est la classe vide). « Tous les hommes sont mortels » devient $z \times (1 - y) = 0$ (la classe des individus qui sont à la fois z – c'est-à-dire hommes – et $(1 - y)$ – non-mortels – est la classe vide). Or les lois empruntées à l'algèbre permettent de déduire de ce système équationnel, par un simple calcul, à la fois $x \times z = 0$ (aucun dieu n'est un homme) et $z \times x = 0$ (aucun homme n'est un dieu). Cet exemple élémentaire donne une première idée du type de calcul auquel conduit l'application des méthodes algébriques aux questions logiques qui relèvent d'un raisonnement.

1. La quantification sur les prédicats apparaît lorsque l'on explicite la signification d'une proposition comme « tous les hommes sont mortels » alors comprise de la manière suivante : « Tous les hommes sont *quelques* mortels. » En termes de classes : la classe des hommes est incluse dans la classe des mortels. On se donne ainsi le moyen d'exprimer plus facilement, au prix d'une maladresse de style, l'équivalence extensionnelle entre deux notions ; par exemple : « Tous les hommes sont *tous* les bipèdes sans plumes. »

Le calcul logique booléen se présente donc, de même que le *calculus ratiocinator* leibnizien, comme un procédé opératoire capable de produire un ou plusieurs résultats. L'une des différences principales vient pourtant de ce que son usage ne présuppose la réalisation d'aucune caractéristique universelle. Il requiert seulement la détermination de règles empruntées à l'algèbre pour l'usage des signes[1]. Et ces règles doivent comprendre notamment les lois de la pensée, puisque Boole se propose « d'exprimer les lois fondamentales du raisonnement dans le langage symbolique d'un calcul ». Si la logique booléenne a une portée métaphysique, c'est donc seulement au sens où elle prétend mettre en évidence les lois qui régissent nos facultés intellectuelles. Par là, elle n'engage aucun projet encyclopédique, aucune analyse systématique des notions et des connaissances visant, à travers la combinaison des caractères, la connexion des choses. L'appareil de la logique booléenne se présente donc comme un calcul formel d'inspiration algébrique, faisant l'économie de la langue caractéristique et du système métaphysique qu'elle implique.

Or la critique frégéenne du calcul logique booléen porte précisément sur ce point (entre autres), puisqu'en comparant sa propre idéographie, ou écriture conceptuelle (*Begriffsschrift*), au calcul de Boole, Frege souligne les limites d'une méthode algébrique qui ne se présente jamais comme une analyse du contenu des formules, mais qui voit au contraire dans l'ignorance délibérée de ce contenu le principe de sa fécondité. Dans un texte où Frege apporte des éclaircissements sur le but de son idéographie, il se démarque de Boole en faisant appel à la distinction leibnizienne entre une langue caractéristique et un *calculus ratiocinator* :

> Je n'ai pas voulu donner en formules une logique abstraite, mais donner l'expression d'un contenu au moyen de signes écrits, et d'une manière plus précise et plus claire au regard que cela n'est possible au moyen des mots. En fait, je

1. Boole souligne que l'une des règles du calcul logique n'appartient pas à celles de l'algèbre. En effet, contrairement au produit logique, le produit algébrique ne vérifie pas toujours la relation $x \times x = x$.

> n'ai pas voulu créer seulement un *calculus ratiocinator* mais une *lingua characterica* au sens de Leibniz, étant bien entendu que le calcul de la déduction est à mon sens partie obligée d'une idéographie[1].

En précisant le sens de ce passage, on verra que la *Begriffsschrift* fait subir à la notion de calcul logique un déplacement sémantique important, et qu'en se référant à la langue caractéristique, Frege prend ses distances à l'égard des travaux de Boole, sans être entièrement fidèle pour autant à la conception leibnizienne de la caractéristique universelle.

Le calcul logique dans l'écriture conceptuelle de Frege

Dans les textes où il compare son idéographie au calcul booléen, Frege souligne que les deux constructions ne visent pas les mêmes buts. Boole utilise les techniques de l'algèbre pour résoudre des problèmes logiques dans lesquels seule la *forme* des propositions et des raisonnements importe. Et si le calcul algébrique est applicable à des domaines sans rapport avec le nombre ou la quantité, ce n'est pas seulement parce que l'on opère sur des signes abstraits, en ignorant provisoirement, pendant le temps du calcul, ce qu'ils représentent. C'est surtout, selon Frege, parce que l'on refuse délibérément toute analyse du *contenu* conceptuel des formules, pour ne retenir que les relations logiques qui existent entre elles.

> Si je le comprends bien, Boole voulait construire une technique pour résoudre de manière systématique des problèmes logiques, technique similaire à l'élimination des inconnues, enseignée par l'algèbre. (...) La logique symbolique de Boole ne représente que la partie formelle du langage[2].

1. Sur le but de l'idéographie, *in* Gottlob Frege, *Écrits logiques et philosophiques,* Paris, Éd. du Seuil, 1971, p. 71.
2. Booles rechnende Logik und die Begriffsschrift, *in* Gottlob Frege, *Nachgelassene Schriften und wissenschaftlicher Briefwechsel,* vol. 1, Hambourg, Felix Meiner, 1969, p. 13-14.

A première vue, le projet de Frege paraît très différent puisqu'il s'agit de construire une langue dans laquelle puisse être exprimé un *contenu* de pensée :

> *A contrario,* on peut maintenant définir le but de mon écriture conceptuelle. J'avais à l'esprit, depuis le début, l'*expression d'un contenu.* Ce que je recherche est une *lingua characterica* en premier lieu pour les mathématiques, non un *calculus* limité à la logique pure. Mais le contenu doit être rendu plus exactement qu'il ne l'est par le langage verbal[1].

Le but de Frege apparaît donc tout à la fois différent puisqu'il vise une langue et non un calcul, plus ambitieux au sens où il doit permettre l'expression d'un contenu dans une langue caractéristique, mais plus restreint dans la mesure où il concerne exclusivement, au moins dans un premier temps, le cadre des mathématiques[2]. Avant de montrer comment les deux projets peuvent pourtant faire l'objet d'une comparaison, il convient de préciser le sens des exigences que Frege veut satisfaire.

Tout d'abord, c'est la volonté d'une représentation logique de la pensée qui requiert la conception d'une langue différente de celles que nous parlons habituellement. Dans l'article intitulé *La science justifie le recours à une idéographie,* reprenant plusieurs idées chères à Leibniz, Frege souligne à la fois l'importance des *signes* pour la pensée, celle d'une *écriture,* par opposition aux signes audibles émis par un locuteur, et l'imperfection logique des langues usuelles. Sur ce dernier point, il écrit :

> La langue n'est pas régie par des lois logiques telles que l'observance de la grammaire puisse suffire à garantir la rigueur formelle du cours de la pensée. Les formes où s'exprime la déduction sont si diverses, si lâches, si mal définies, que des hypothèses peuvent être introduites sans qu'on y prenne garde, et on omet de les compter quand on récapitule les conditions nécessaires à la validité de la conclusion[3].

1. *Ibid.,* p. 13.
2. Dans la préface à la *Begriffsschrift,* Frege évoque la possibilité d'un usage ultérieur de l'idéographie dans d'autres domaines que l'arithmétique comme la géométrie, l'*analysis situs,* la théorie du mouvement, la mécanique, et la physique.
3. Que la science justifie le recours à une idéographie, *in* Frege, *Écrits logiques et philosophiques,* p. 64-65.

Toutefois, la difficulté ne concerne pas seulement les liens déductifs entre des propositions, mais également l'expression des concepts :

> Il n'y a qu'une correspondance imparfaite entre la façon dont les mots sont concaténés et la structure des concepts. (...) Une *lingua characterica* devrait, comme dit Leibniz *peindre non pas les paroles, mais les pensées*. Les langues formulaires des mathématiques s'approchent de ce but, et l'atteignent même en partie. Mais (...) celle de l'arithmétique elle-même est inadéquate pour son propre domaine ; car précisément aux points les plus importants, lorsque de nouveaux concepts doivent être introduits, de nouveaux fondements posés, elle doit abandonner le champ au langage verbal (...) [1].

Malgré ces limites, le langage formulaire de l'arithmétique sert de modèle à Frege[2], comme l'indique le sous-titre de la *Begriffsschrift* : *« Une langue formulaire de la pensée pure construite sur le modèle de l'arithmétique »* ; mais il doit être complété par un formalisme capable d'en combler les lacunes logiques. Or la logique de Boole se révèle être inadéquate à cette fin pour deux raisons. Premièrement, parce qu'étant purement formelle, elle fait entièrement abstraction des contenus de pensée, et deuxièmement, parce qu'elle utilise des signes arithmétiques (+, 1, 0) dans un sens qui ne s'accorde pas avec leur signification mathématique. Frege se propose donc de concevoir une langue caractéristique nouvelle, capable de compléter le formulaire arithmétique :

> Si l'on prend une vue d'ensemble du langage formulaire de Boole, on voit qu'il consiste à habiller la logique abstraite du vêtement des signes algébriques ; il n'est pas propre à l'expression d'un contenu et tel n'est pas non plus son but. Or, c'est là précisément mon intention. Je veux fondre les quelques signes que j'ai introduit avec les signes mathématiques en un seul formulaire. Les signes existants correspondraient à peu près aux racines

1. Boole rechnende Logik und die Begriffsschrift, *in* Frege, *Nachgelassene Schriften und wissenschaftlicher Briefwechsel*, p. 14.
2. Frege se sert notamment de la distinction entre deux types de signes, « ceux sous lesquels on peut se représenter des choses différentes et ceux qui ont un sens tout à fait déterminé », comme les lettres d'une part (*a, b*, etc.) et +, −, 0, 1 d'autre part, cf. *Begriffsschrift*, § 1.

> des mots, tandis que les signes introduits sont à comparer aux terminaisons et aux particules qui établissent les rapports logiques entre les contenus des racines[1].

Les buts de Boole et de Frege apparaissent donc comme étant très différents, puisque l'un conçoit un procédé de calcul pour la solution de questions logiques, et l'autre une idéographie pour la formalisation de l'arithmétique. On pourrait alors penser que Frege renonce à l'idéal d'un calcul logique proprement dit s'il ne précisait, comme on l'a vu : « Le calcul de la déduction est à mon sens partie obligée d'une idéographie. » De surcroît, la référence à Leibniz, dans *Le calcul logique de Boole et l'idéographie*, rappelle explicitement que la *lingua characterica* leibnizienne devait rendre possible un *calculus ratiocinator*. Et dans le passage de Leibniz cité par Frege, le calcul apparaît bien comme un véritable procédé permettant la résolution de problèmes :

> Si daretur vel lingua quædam exacta, vel genus scripturæ vere philosophiæ..., omnia quæ ex datis ratione assequi, inveniri possent quodam genere calculi, perinde ac resolvuntur problemata arithmetica aut geometrica[2].

Que reste-t-il de cet idéal d'un procédé logique de calcul dans l'idéographie frégéenne ? Dans le calcul de Boole,

> les lois logique prennent la forme d'algorithmes (...) [et] l'usage de signes arithmétiques à des fins logiques a l'avantage que l'on s'épargne la nécessité d'apprendre un algorithme entièrement nouveau[3].

Sur ce point précis, l'écriture conceptuelle ne semble pas disposer des mêmes capacités. Frege minimise cependant l'avantage que semble présenter le calcul booléen, en montrant comment son idéographie permet, sans faire usage d'un algorithme inspiré de l'arithmétique ou de l'algèbre, de résoudre un problème de logique

1. Sur le but de l'idéographie, *in* Frege, *Écrits logiques et philosophiques*, p. 73.
2. *De Scientia universali seu calculo philosophico*. On trouvera le texte intégral dans *G*. VII, p. 198-199. « Si l'on disposait de quelque langue exacte, ou d'un genre d'écriture vraiment philosophique, (...) toutes les conséquences qui suivent rationnellement de certaines données pourraient être trouvées par une sorte de calcul, de même que l'on résout des problèmes arithmétiques ou géométriques. »
3. Frege, *Nachgelassene Schriften und wissenschaftlicher Briefwechsel*, p. 13.

bien précis, traité par Boole, Schröder et Wundt à l'aide de procédés de calcul[1]. Au moment de rappeler les raisons pour lesquelles son idéographie doit être jugée supérieure au calcul de Boole, il écrit :

> On peut l'utiliser pour résoudre le type de problèmes traités par Boole, et même le faire avec moins de règles préliminaires pour le calcul. C'est le point auquel j'attache le moins d'importance, car de tels problèmes se présenteront rarement, s'ils se présentent un jour, en science[2].

La capacité à déduire de manière systématique certaines conséquences d'un ensemble d'énoncés connus, en suivant un algorithme de calcul et afin de résoudre tel ou tel problème de logique formelle, n'est donc pas considérée par Frege comme un objectif important de l'idéographie. Pour lui, si le calcul booléen manifeste cette capacité, c'est uniquement en choisissant des problèmes *ad hoc* dans lesquels les difficultés d'analyse logique des contenus conceptuels sont supposés résolus : le contenu des pensées est ignoré et seule la forme logique abstraite est considérée. L'essentiel n'est donc pas d'emprunter à l'algèbre des algorithmes de calcul, car leur efficacité est illusoire ou limité à des cas bien particuliers :

> Je crois que presque toutes les erreurs faites dans une inférence ont leur racine dans l'imperfection des concepts. Boole présuppose donnés des concepts logiquement parfaits, et ainsi, que la partie la plus difficile de la tâche est déjà réglée ; il peut alors tirer ses inférences des hypothèses données au moyen d'un procédé mécanique de calcul. Stanley Jevons a d'ailleurs inventé une machine pour le faire. Mais si nous avons des concepts parfaits, au contenu desquels nous n'avons plus besoin de nous référer, nous pouvons aisément nous prémunir de l'erreur, même sans calcul[3].

Ainsi, en majorant l'importance de l'expression logique d'un contenu grâce à une *lingua characterica*, Frege minore le rôle du *calculus ratiocinator* compris comme un procédé de calcul destiné à pro-

1. Le problème est posé puis résolu à la fin de l'article Booles rechnende Logik und die Begriffsschrift, *in* Frege, *op. cit.* Certaines relations entre les propriétés A, B, C, D, E d'une classe de phénomènes étant connues, on demande de « calculer » d'autres relations entre ces propriétés, déductibles de celles qui sont connues.
2. *Ibid.*, p. 52.
3. *Ibid.*, p. 39.

duire un résultat. L'expression adéquate des contenus de pensée rend quasiment inutile la recherche d'un algorithme permettant de calculer les conséquences logiques. Mais comment comprendre, alors, l'affirmation frégéenne selon laquelle « le calcul de la déduction est partie obligée d'une idéographie » ?

Si la recherche d'une procédure algorithmique n'est pas considérée par Frege comme un problème essentiel, l'exigence d'une expression explicite et purement logique des rapports de déduction est évidemment maintenue : elle se présente comme l'une des *fonctions* de la langue caractéristique (qui doit en outre être capable de représenter les contenus de pensée)[1], et donc, en ce sens, comme une partie obligée de l'idéographie. Deux composantes de l'idée de calcul logique doivent donc être distinguées.

1 / Dans un raisonnement, on doit pouvoir vérifier que la séquence des énoncés ou des formules répond strictement aux règles de la logique déductive, comme chaque étape d'une opération numérique répond aux règles du calcul arithmétique.

2 / Un procédé doit permettre de produire un ou plusieurs résultats, conséquences logiques d'un ensemble d'énoncés initialement donnés, de même qu'un algorithme pour l'addition produit la somme de plusieurs nombres posés au départ.

On a vu que chez Leibniz, la langue caractéristique doit rendre possible un *calculus ratiocinator* compris en ce deuxième sens. Chez Frege, en revanche, l'idéal d'un calcul comme procédé opératoire passe au second plan ; le calcul logique exprime bien plutôt une exigence de formalisation des constructions déductives, à laquelle doit satisfaire l'idéographie. Par là, on aperçoit l'infléchissement sémantique subi par l'expression « calcul logique » dans l'idéographie frégéenne : on dira du raisonnement lui-même qu'il est un calcul lorsqu'il est entièrement formalisé, ce qui, chez Frege, suppose qu'il puisse exprimer les contenus de pensée.

1. Sur les fonctions de l'idéographie, lire les remarques de Claude Imbert dans Frege, *Écrits logiques et philosophiques*, p. 53.

Or en minimisant la composante opératoire du calcul logique, Frege semble écarter toute possibilité de rapprochement avec la machine. Même si Leibniz n'envisage pas explicitement cette possibilité, une mécanisation du *calculus ratiocinator* ne serait pas contraire à l'esprit de la philosophie leibnizienne ; par ailleurs, le calcul booléen et les développements d'une algèbre de la logique sont effectivement à l'origine des machines logiques que l'on construit au XIX[e] siècle, comme celles de Jevons et Marquand[1]. *A contrario,* le « calcul de la déduction » au sens de Frege est trop lié aux caractéristiques syntaxiques de la *Begriffsschrift,* l'idéal frégéen de la *lingua characterica* trop éloigné de la recherche d'un algorithme opératoire, pour que cette idéographie suscite un intérêt pour la mécanisation des raisonnements ou pour une quelconque machine logique.

Pourtant, l'œuvre de formalisation accomplie par Frege n'en est pas moins décisive pour notre propos. Car l'idée de machine en logique ne doit pas être confondue avec la possibilité d'une mécanisation des procédés logiques. On a pu montrer que la problématique des machines logiques, au XIX[e] siècle, ne conduisait pas à l'idée d'un calculateur universel au sens de Turing[2]. En revanche, l'entreprise de formalisation de l'arithmétique a rendu possible le programme de Hilbert, ouvrant ainsi indirectement la voie à l'arithmétisation gödelienne de la syntaxe. Par là, les procédés opératoires de l'arithmétique devaient rejoindre, d'une manière quelque peu inattendue, les exigences de formalisation exprimées

1. Sur cette tradition des machines logiques, cf. Martin Gardner, *Logical machines and diagrams,* New York, McGraw-Hill, 1959, trad. franç., *L'étonnante histoire des machines logiques,* Paris, Dunod, 1964 ; Marguin, *op. cit.* ; et Mosconi, *op. cit.* Certains dispositifs mécaniques utilisés pour la résolution de questions logiques simples ont cependant été construits avant les travaux de Boole.
2. Cf. les analyses minutieuses de Jean Mosconi qui écrit : « On peut comprendre assez aisément que les "machines logiques" dans la lignée du "piano logique" de Jevons ou du démonstrateur de Stanhope n'aient pas eu à cet égard de fécondité théorique. Réalisées dans le cadre théorique de l'algèbre de la logique ou de la syllogistique, elles se révélaient d'un intérêt restreint pour la logique moderne (...). Elles n'apparaissent plus guère que comme des curiosités d'un autre âge, même si leur conception avait pu être l'occasion d'une réflexion théorique intéressante, notamment chez Jevons » (Mosconi, *op. cit.,* p. 30).

par les logiciens, et permettre de résoudre plusieurs problèmes logiques fondamentaux posés par Hilbert. D'où l'étude des fonctions récursives, restreinte au cas des fonctions numériques[1], celle des fonctions lambda-définissables et des machines de Turing. Il nous reste donc à montrer comment les procédés de l'arithmétique permettent de répondre aux problèmes soulevés par l'idéal hilbertien d'une formalisation des mathématiques. Car la puissance opératoire de l'arithmétique a joué un rôle décisif pour l'introduction d'une notion de machine en logique.

1. L'important étant de comprendre que cette « restriction » est sans incidence sur la généralité des résultats obtenus en théorie de la calculabilité.

VI

CALCUL ET DÉMONSTRATION. HILBERT ET GÖDEL

QUELQUES PROLONGEMENTS GÖDELIENS DU PROGRAMME DE HILBERT

En 1879, la *Begriffsschrift* donne le premier exemple d'une langue logique permettant l'écriture de raisonnements formalisés. Or cette œuvre, parfois considérée comme le texte le plus important de toute l'histoire de la logique, semble être aussi éloignée que possible de toute considération relative aux machines. On a vu en effet quel était le but de l'idéographie. Pour Frege, cet instrument de pensée doit libérer l'esprit humain du joug que les langues usuelles font peser sur lui en s'écartant des règles de la logique par leurs ambiguïtés ou leurs constructions idiomatiques. En ce sens, et comme l'indique le sous-titre de l'opuscule, l'idéographie est une langue formulaire de « la pensée pure »[1]. Mais cette référence à un idéal de pureté ne vise pas seulement les imperfections et les singularités grammaticales des langues que nous parlons. Frege veut également prévenir l'usage, tout aussi contraire à la logique, du sous-entendu, du présupposé, ou de la compréhension intuitive dans les raisonnements.

> Afin que rien d'intuitif ne s'insinuât subrepticement dans cette démarche, il importait de tout réduire à une chaîne déductive dépourvue de lacunes[2].

1. L'expression est empruntée à Kant, et comprise également au sens kantien (c'est-à-dire : où ne se mêle rien qui appartienne à la sensation).
2. *Begriffsschrift,* p. 99 (la pagination est celle de la trad. franç. partielle, *in* Rivenc et De Rouilhan, *op. cit.*).

Enfin, la pureté de la pensée qu'exprime l'idéographie s'oppose aussi à la genèse psychologique de nos idées et aux singularités subjectives des représentations. On connaît la critique que Frege opposait à la confusion du logique et du psychologique, et l'on sait quelle importance il accordait à la distinction entre la pensée *(Gedanke),* qui n'a besoin d'aucun porteur, d'aucune conscience pour être vraie, la représentation *(Vorstellung)* propre à un individu, et les processus par lesquels la pensée peut être appréhendée. La pensée qu'exprime l'idéographie est donc également pure de tout ce qui relève en propre de la représentation.

Les caractères de la langue formulaire ne sont donc pas l'expression d'une logique abstraite, dépourvue de tout contenu. Au contraire, ce qui est exprimé par l'idéographie est le contenu conceptuel d'une pensée indépendante des représentations. D'où la comparaison, hautement significative, avec le microscope :

> Je crois pouvoir exprimer la relation de mon idéographie à la langue usuelle de la manière la plus claire possible, si je la compare à la relation du microscope à l'œil. Certes l'œil, par l'étendue de ses usages et la souplesse avec laquelle il s'adapte aux circonstances les plus variées, possède de gros avantages sur le microscope. (...) Mais l'œil se révèle insuffisant dès que s'imposent, à des fins scientifiques, des distinctions plus précises[1].

Le microscope sert à observer ce qui est, et dans l'esprit de Frege, l'idéographie doit être, comme lui, un artifice transparent, dans lequel on puisse penser comme on peut mener une observation à travers l'appareil optique. De ce point de vue, l'écriture conceptuelle est un instrument pour l'expression d'une pensée vraie, et ne fait pas elle-même l'objet d'une analyse logique. La logique frégéenne n'élabore aucune étude métalinguistique de l'idéographie. Comme on l'a déjà souligné précédemment (au chapitre I), l'universalisme logique de Frege implique qu'aucune perspective extérieure ne permet, par un effet de surplomb, d'étudier logiquement l'écriture conceptuelle, d'en faire ainsi un objet d'examen[2]. En ce

1. *Ibid.,* p. 100.
2. Cela ne signifie cependant pas qu'en droit, aucune question métasystématique ne puisse être élaborée de l'intérieur. Sur ce point, lire, ci-dessus, la n. 3, p. 20.

sens, la *Begriffsschrift*, tout comme les *Principia Mathematica* de Russell et Whitehead, permettent l'écriture de raisonnements formalisés sans être pour autant des systèmes formels de signes au sujet desquels leurs auteurs auraient envisagé l'exercice d'une pensée logique et métalinguistique. Or sans ce redoublement du langage en un métalangage, opéré notamment par Hilbert et qui implique un profond changement de perspective – une autre philosophie de la logique et des mathématiques –, l'entreprise de formalisation ne peut recevoir aucune interprétation mécaniste. La mise en correspondance des machines et des constructions formelles est impensable tant que l'on ne se donne aucun système réglé de signes, considéré comme un dispositif autonome, et pouvant faire lui-même l'objet d'une analyse logico-mathématique.

L'optique frégéenne est donc aux antipodes de l'idée de machine. Tout d'abord, on l'a vu, parce qu'en minimisant l'importance de la dimension opératoire du calcul logique, qui était mise en valeur par l'algèbre de la logique, elle s'éloigne de la problématique des machines à la Jevons. Au reste, Frege n'accorde qu'un très faible intérêt à ces dispositifs ingénieux qui, selon l'expression de Jean Mosconi, « reflètent de quelque manière dans leur organisation et leur fonctionnement la structure de la situation logique étudiée et les étapes de la résolution »[1]. Mais d'autre part, si Frege relègue au second plan la possibilité d'une mécanisation des procédures déductives, sa philosophie de la logique et des mathématiques, tout comme celle de Russell au moment de la rédaction des *Principia Mathematica*, ne permet pas encore une interprétation des constructions formelles en termes de machines. Une telle interprétation supposait en effet à la fois l'élaboration d'une analyse métalinguistique et l'orientation des recherches vers les procédures *finies* et vers une exigence d'effectivité ; toutes choses qui demeuraient étrangères aux préoccupations de Frege et Russell au début du siècle.

L'entreprise logiciste de formalisation et de fondation des mathématiques, bien qu'étrangère à l'idée de machine, n'en a pas

1. Mosconi, *op. cit.*, p. 51.

moins joué un rôle décisif, quoique indirect, pour l'introduction de cette idée en logique puisque le programme de Hilbert et les résultats négatifs de Gödel supposent la possibilité d'une formalisation des raisonnements mathématiques. Ce sont d'ailleurs souvent les *Principia Mathematica,* modifiés et réinterprétés, qui ont servi de base ou de modèle pour la construction et l'étude des systèmes formels[1].

Pour qu'une correspondance profonde et féconde pût être élaborée entre les constructions logiques et les machines, il fallait donc premièrement que s'imposât l'idée d'une formalisation des raisonnements mathématiques, au prix d'un abandon des machines logiques, et deuxièmement que les langues formulaires fussent conçues comme des systèmes de signes pouvant faire l'objet d'une étude logico-mathématique extérieure. Le programme de Hilbert accomplit cette seconde condition, nécessaire bien que non suffisante. S'ensuivent alors un ensemble de conséquences dont l'importance n'apparaît, pour certaines d'entre elles, qu'après l'article de Gödel publié en 1931, ou même après la solution de l'*Entscheidungsproblem* par Church et Turing en 1936, mais qui préfigurent plusieurs transformations ultérieures de la logique, dont l'introduction dans cette discipline d'une idée de machine. Les théorèmes d'incomplétude sont souvent présentés, à juste titre, comme une mise en échec du programme de Hilbert. Pourtant, paradoxalement, de notre point de vue, par la démonstration de ces théorèmes, Gödel prolonge et accomplit plusieurs des idées dues à Hilbert plus qu'il ne contredit son programme. On s'en convaincra, ci-dessous, par une étude plus approfondie des travaux de Gödel. Mais on pourra dès maintenant en apercevoir quelques-unes des raisons par l'examen sommaire des prolongements dont il vient d'être question, et qui éclairent la possibilité d'introduire en logique une idée de machine.

1. Cf. le titre de l'article de Gödel, en 1931 : Sur les propositions formellement indécidables des *Principia Mathematica* et des systèmes apparentés.

Logique et opérations numériques

Du point de vue métamathématique, les signes et suites de signes d'un système formel sont des entités finies comparables aux entiers de l'arithmétique. On se souvient qu'en 1922, Hilbert faisait explicitement la comparaison entre les chiffres et les formules ou les démonstrations qui deviennent, dans le cadre d'un système formel, « quelque chose de concret et de tangible »[1]. Or la suggestion est prise au pied de la lettre par Gödel qui invente un codage numérique pour les signes, les expressions bien formées et les preuves, transformant ainsi les propriétés et relations métamathématiques en propriétés et relations arithmétiques, et réduisant les problèmes de décision relatifs aux formules à des questions d'effectivité portant sur des prédicats numériques. De là l'importance logique donnée aux calculs arithmétiques et aux fonctions numériques ; de là, également, l'importance et la signification nouvelles qu'acquiert leur composante opératoire. On devine ou l'on pressent alors les rapports qui seront établis, ultérieurement, entre un problème logique de décision, par exemple, et les capacités d'un calculateur numérique.

Logique et effectivité

Le programme de Hilbert donne à la notion d'*effectivité* une place centrale en logique. D'abord par l'énoncé du problème de la décision et la recherche d'une procédure effective capable de le résoudre positivement. Ensuite, à travers les travaux de Church et de Turing sur la possibilité d'une solution négative. Mais aussi, enfin, parce que l'intérêt des systèmes formels est souvent conçu comme étant relatif à plusieurs exigences d'effectivité (bien que l'on puisse aussi s'intéresser aux systèmes qui ne satisfont pas ces exi-

1. Cf. ci-dessus, chap. I, et Largeault, *op. cit.*, p. 119 et 124.

gences) : on demande qu'il existe une procédure effective permettant de déterminer si un signe appartient à l'alphabet, si une expression est bien formée, si une formule est un axiome, si une formule est conséquence immédiate d'une ou de plusieurs autres formules, et, en définitive, si une suite de formules est bien la démonstration de telle ou telle expression bien formée. Or la notion de fonction récursive introduite par Gödel en 1931[1] est un outil qui permet de préciser le sens de cette exigence d'effectivité, et de vérifier qu'elle est bien satisfaite. En outre, c'est une généralisation de cette notion de récursivité (devenant alors équivalente à celle de machine de Turing ou de fonction lambda-définissable) qui permet à Church de résoudre négativement le problème logique de la décision en 1936.

Démonstration et calcul

Qu'est-ce qu'un calcul logique selon Hilbert, et du point de vue de son programme ? Comme chez Frege, l'idée de calcul n'a plus le sens – d'origine arithmétique et algébrique – d'une opération permettant la production d'un résultat. Il fait seulement référence à l'enchaînement déductif des preuves, sans lacunes et rigoureusement déterminé. Mais chez Hilbert, le calcul logique doit être mis en relation avec la possibilité d'une étude purement morphologique et syntaxique des constructions formelles de signes et des transformations qui permettent de faire progresser la déduction. Le calcul est donc lié à la démonstration, mais en un sens qui reste encore faible puisqu'il ne vise que la forme des preuves. Dans ce contexte, Gödel accomplit un pas décisif lorsqu'il arithmétise la syntaxe : il donne au calcul logique un nouveau sens opératoire, et à son rapport avec la démonstration une signification beaucoup plus forte. Premièrement, parce qu'il aurait suffit, alors, s'il en eût existé, de trouver une fonction calculable adéquate pour apporter une solu-

1. Il s'agit des fonctions que l'on appelle aujourd'hui « primitives récursives ».

tion positive au problème de la décision. Deuxièmement, aussi, parce que la recherche d'une démonstration, dans un système formel, peut être considérée comme un calcul ou l'application d'un algorithme (dont on ne sait pas, en général, s'il conduit à un résultat). Troisièmement, enfin, parce qu'à l'inverse, calculer le résultat d'une opération numérique (ou encore, plus généralement, appliquer un algorithme) revient à démontrer quelles sont les conséquences de sa définition[1]. A cet égard, l'équivalence entre calcul et démonstration prendra un sens précis lorsque la notion de calcul aura pu être réélaborée grâce aux travaux de Gödel, Church, Kleene, Turing et d'autres, c'est-à-dire à travers la théorie de la calculabilité, naissante dans les années trente. Mais ce point ne pourra évidemment pas être éclairci avant que soient examinés plus précisément les travaux de Gödel dans l'article de 1931.

Démonstration et vérité

Malgré toutes les réformes introduites par la *Begriffsschrift*, Frege conserve de la logique classique l'idée d'une présentation des vérités logiques, qui prend, avec l'écriture conceptuelle, une forme systématique et formalisée. Selon le point de vue logiciste de Frege, ces vérités logiques doivent suffire à démontrer les vérités de l'arithmétique, car « chaque proposition arithmétique (est) une loi logique, bien que dérivée »[2]. Selon Hilbert, *a contrario*, l'arithmétique et la logique doivent être construites simultanément et au moyen d'un système formel. Le problème des rapports entre démonstration et vérité se pose alors différemment. En ce qui concerne les vérités logiques, le programme de Hilbert demande

1. Sur ce point, cf. Jean-Yves Girard, La machine de Turing, de la calculabilité à la complexité, *in* Girard et Turing, *op. cit.*, p. 18-20. Il ne s'agit évidemment pas, ici, des moyens psychologiques par lesquels un mathématicien s'efforce de démontrer une proposition mais de la possibilité de rechercher systématiquement et mécaniquement une preuve (indépendamment des limites imposées par les contraintes physiques sur cette recherche).
2. Frege, *Grundlagen der Arithmetik*, Breslau, Max und Hermann Marcus, 1884, § 87, trad. franç., *Les fondements de l'arithmétique*, Paris, Éditions du Seuil, 1969, p. 211.

que l'on conçoive un système dans lequel les propositions démontrables soient exactement les formules universellement valides. Le théorème de complétude montre que cette exigence peut effectivement être satisfaite. Pour les vérités arithmétiques, le programme formaliste se propose de construire un système qui soit à la fois cohérent et syntaxiquement complet : tout énoncé arithmétique serait alors démontrable, ou réfutable par la démonstration de sa négation. Dans cette perspective, la vérité des propositions arithmétiques ne serait pas garantie par leur réduction à des vérités logiques, mais par une démonstration de cohérence. Selon la philosophie des mathématiques de Hilbert, l'absence de contradiction dans un système formel quelconque – et donc en particulier s'il est conçu pour la formalisation de l'arithmétique – est en effet un critère suffisant de la vérité des axiomes :

> Si les axiomes arbitrairement posés avec toutes leurs conséquences ne se contredisent pas, alors ils sont vrais et les choses qu'ils définissent existent. C'est pour moi le critère de la vérité et de l'existence[1].

En niant la possibilité de réduire les lois de l'arithmétique à des vérités *logiques*, Hilbert ne renonçait donc pas à exprimer formellement des *vérités* mathématiques. Et par conséquent, loin d'être mise en question, l'équivalence entre vérités et propositions formellement démontrables demandait à être rigoureusement établie par une preuve métamathématique. Or en soulevant expressément la question de cette équivalence, à laquelle il souhaitait pouvoir donner une réponse positive, Hilbert introduisait aussi la possibilité d'une solution négative, qui fut effectivement apportée par Gödel, au grand dam du programme formaliste. En un sens, cependant, les théorèmes d'incomplétude prolongent la distinction hilbertienne entre le vrai (dans les mathématiques usuelles) et le démontrable (dans un système formel), dans la mesure où ils rendent impossible l'établissement d'une équivalence entre ces deux notions.

1. Lettre de Hilbert à Frege, du 29 décembre 1899, *in* Rivenc et De Rouilhan, *op. cit.*, p. 227. Une telle déclaration suscite les protestations de Frege qui répond, le 6 janvier 1900 : « C'est sûrement à propos de votre critère de la vérité que nos vues sont le plus opposées » (*ibid.*, p. 234).

On pourrait alors penser qu'en prouvant l'incomplétude des systèmes hilbertiens pour l'arithmétique et celle de leurs extensions, Gödel ruinait l'intérêt que suscitaient ces constructions formelles ainsi que la métamathématique qui leur est corrélative. L'histoire ultérieure de la logique montre qu'il n'en est rien, et que le programme d'une théorie de la démonstration demandait seulement à être infléchi et redéfini. Elle montre aussi que la logique peut prendre un autre sens que celui d'un exposé, fût-il systématique, de certaines vérités, puisqu'elle est souvent conçue aujourd'hui comme l'étude d'une relation de déductibilité caractérisée par un ensemble de règles, cette étude pouvant être motivée par tout autre chose qu'une présentation des *vérités* logiques ou mathématiques. Cela ne signifie évidemment pas que la question de la vérité devienne étrangère aux interrogations des logiciens. Mais l'intérêt que présente l'étude des propriétés d'une syntaxe, ou d'un système de règles, émane autant de la linguistique, de l'informatique, de l'intelligence artificielle, voire de la psychologie et des sciences cognitives que de la vérité mathématique. Et de notre point de vue, on comprend d'autant mieux l'intérêt d'une telle conception de la logique que l'on saisit plus profondément les correspondances qui existent entre une machine et un système de signes syntaxiquement défini, ou encore, nous y reviendrons, entre les preuves formelles constructives étudiées en théorie de la démonstration et les programmes élaborés en informatique. A la lumière de ces quelques remarques, on peut apercevoir dès maintenant dans le programme de Hilbert et les travaux de Gödel l'origine de cette orientation de la logique vers la notion de machine.

Historiquement, ces quatre prolongements du programme de Hilbert ne prennent tout leur sens que progressivement, à partir de la démonstration des théorèmes d'incomplétude. Ils permettent surtout de comprendre quelles sont les conditions qui ont permis de cristalliser l'idée de machine à l'occasion d'un problème purement logique à l'origine, et montrent pourquoi celle-ci est tout autre chose qu'une métaphore curieuse ou un modèle singulier auquel Turing, seul, aurait pensé pour caractériser la notion d'effectivité.

L'idée de machine en logique est enracinée dans la problématique d'une théorie de la démonstration et touche, à ce titre, des questions centrales de la logique contemporaine. Pour s'en convaincre, on ne saurait se contenter des quatre points qui viennent d'être présentés sommairement. S'ils peuvent être pressentis dans le programme de Hilbert, leur importance décisive n'est pas révélée avant les travaux de Gödel. On se propose donc maintenant d'en approfondir le sens à partir d'un examen des moyens démonstratifs utilisés dans la preuve des théorèmes d'incomplétude.

Les théorèmes d'incomplétude de Gödel et leur démonstration

L'interprétation philosophique des théorèmes d'incomplétude concerne avant tout l'épistémologie des mathématiques et de la logique, et plus précisément le programme de Hilbert et la question des fondements. Mais les philosophes ont souvent estimé que la signification de ces théorèmes n'était pas sans conséquence pour des questions bien différentes comme celle de la comparaison entre l'homme et la machine, et de la possibilité de comprendre la pensée humaine par référence aux machines et à leurs capacités. Dans cette perspective, l'interprétation des théorèmes de Gödel est cependant si délicate qu'elle a conduit les philosophes à reconnaître dans ces résultats logiques des arguments capables d'étayer tantôt des thèses mécanistes, tantôt des positions antimécanistes, qui semblent pourtant contredire les premières. Malgré ces difficultés, ou peut-être bien à cause d'elles, les théorèmes d'incomplétude ont suscité une quantité impressionnante de commentaires qui s'efforcent d'en tirer différentes conséquences relatives à la philosophie de l'esprit, et donc en dehors de la logique proprement dite. Dans les pages qui suivent, en interrogeant les théorèmes de Gödel du point de vue des machines, on se démarque des commentaires philosophi-

ques les plus courants pour une double raison. D'abord, parce que l'on ne cherche pas à transférer hors du champ logique des résultats que l'on croirait pouvoir appliquer à la philosophie de l'esprit ; mais pour autant, ce n'est ni la question des fondements ni la philosophie des mathématiques qui est visée. On voudrait montrer en effet qu'il est possible de poser à l'intérieur du domaine logique une problématique philosophique différente, et relative aux machines. Deuxièmement, c'est moins l'énoncé des résultats obtenus par Gödel qui retiendra notre attention que certaines techniques utilisées pour leur démonstration, qui ont, du point de vue des machines logiques et informatiques, une importance décisive et durable. On y trouve en effet un ensemble d'outils et d'idées qui furent conçus, à l'origine, pour la solution de problèmes purement logiques, et qui ont pu recevoir, ultérieurement, une interprétation en termes informatiques. A cet égard, la lecture des théorèmes de Gödel qui est proposée ici prolonge donc directement les analyses des machines de Turing et du lambda-calcul qui ont été développées précédemment. Elle n'exige pas un examen détaillé de leur démonstration, mais demande que l'on en comprenne les ressorts essentiels, en distinguant les principales étapes de la preuve[1].

Le codage numérique des expressions formelles

Gödel considère un système formel inspiré des *Principia Mathematica*, dans lequel il est possible d'exprimer les démonstrations et les résultats connus de l'arithmétique élémentaire. Conformément à la philosophie hilbertienne des mathématiques, on distingue les expressions formelles, internes au système, et les énoncés métama-

1. Comme pour les travaux relatifs aux machines de Turing et au lambda-calcul précédemment examinés, notre point de vue n'est pas celui d'un historien des mathématiques ou de la logique. Il s'agit moins de restituer précisément les résultats obtenus par Gödel en 1931 que de souligner ce qui importe pour la question des machines, dans les énoncés habituellement appelés « théorèmes d'incomplétude » (et dans leur démonstration), qui sont en réalité, à plusieurs égards, des généralisations de ce que Gödel a publié en 1931.

thématiques, qui portent sur le formalisme. Mais alors que Hilbert suggérait une simple comparaison entre les nombres entiers de l'arithmétique, d'une part, et, d'autre part, les signes et suites de signes du système, objets de la métamathématique (il s'agit, ici et là, d'entités finies), Gödel donne à ce rapprochement un sens littéral puisqu'il conçoit un codage numérique et systématique de toutes les expressions formelles. A chaque signe, à chaque formule, à chaque démonstration du système logique considéré, est associé un nombre entier distinct, appelé « nombre de Gödel » de l'expression. Le codage est effectif au sens suivant :

(i) il existe une procédure effective qui permet de calculer en un nombre fini d'étapes le nombre de Gödel de n'importe quelle expression formelle ou suite d'expressions formelles donnée ;

(ii) il existe une procédure effective permettant de déterminer en un nombre fini d'étapes si un nombre entier quelconque donné est le nombre de Gödel d'une expression formelle (ou d'une suite de telles expressions), et, le cas échéant, de trouver quelle est cette expression (ou cette suite) par un calcul numérique.

L'arithmétisation de la métamathématique

Ce codage permet de concevoir les énoncés et relations métamathématiques comme des énoncés et des relations numériques. Par exemple, au lieu de parler de l'ensemble des formules du système S, ou du prédicat « x est une formule du système S », on considère l'ensemble des entiers qui sont chacun le nombre de Gödel d'une formule de S, ou encore le prédicat « x est le nombre de Gödel d'une formule de S », noté Form (x) (par abus de langage, on dit aussi « n est une formule » lorsque n est le nombre de Gödel d'une formule). On réalise ainsi ce qu'il est convenu d'appeler « l'arithmétisation de la métamathématique », en traduisant sous la forme de prédicats ou de relations numériques des expressions syntaxiques comme « x est une formule du système S », « la formule x est la

négation de la formule y », « x est une démonstration, dans S, de la formule y », ou encore « x est une formule démontrable (c'est-à-dire un théorème) de S ». Les prédicats et relations numériques ainsi obtenus peuvent être notés, respectivement, Form (x), Nég (x,y), Dém (x,y), Th (x). Ils permettent d'écrire, dans le métalangage, une relation *numérique* exprimant l'incomplétude du système S considéré, par exemple à partir de l'énoncé suivant : « Il existe deux entiers x et y tels que Form (x), Form (y), Nég (x,y) et, pour tout entier z, il est faux que Dém (z,x) et faux que Dém (z,y). »

Le plongement des énoncés métamathématiques dans le système formel

Le système S choisi par Gödel permet de formaliser l'arithmétique élémentaire. Si $k_1, \ldots, k_n$ sont des entiers naturels, on notera $\boldsymbol{k_1}, \ldots \boldsymbol{k_n}$ la représentation formelle de ces entiers dans S. Est-il possible, dès lors, d'exprimer par des formules *du* système des relations métamathématiques numériques comme, par exemple, Form (x), qui, via le codage, portent *sur* le système ? Gödel montre que l'on dispose effectivement de moyens suffisants pour plonger les énoncés métalinguistiques dans le langage objet lui-même, à condition de donner à ce plongement un sens précis. A cette fin, on donne la définition suivante : une *relation* numérique n-aire R est représentable dans le système par une formule $A[x_1, \ldots, x_n]$ (où $x_1, \ldots, x_n$ sont les variables libres de A) lorsque l'on a :

i) si $R(k_1, \ldots, k_n)$, alors $\vdash A[\boldsymbol{k_1}, \ldots, \boldsymbol{k_n}]$
(c'est-à-dire : si les entiers $k_1, \ldots, k_n$ vérifient la relation R, alors la formule $A[\boldsymbol{k_1}, \ldots, \boldsymbol{k_n}]$ est démontrable dans le système) ;

ii) si non-$R(k_1, \ldots, k_n)$, alors $\vdash \neg A[\boldsymbol{k_1}, \ldots, \boldsymbol{k_n}]$
(c'est-à-dire : si les entiers $k_1, \ldots, k_n$ ne vérifient pas la relation R, alors la formule $\neg A[\boldsymbol{k_1}, \ldots, \boldsymbol{k_n}]$ est démontrable dans le système).

De manière similaire, une *fonction* numérique (par exemple la fonction unaire f) est représentable dans le système S lorsqu'il existe une formule à deux variables libres, $A[x,y]$ qui vérifie la condition suivante : pour deux entiers n et m quelconques, on a $f(m) = n$ si et seulement si la formule $\forall y(A[\boldsymbol{m},y] \leftrightarrow y = \boldsymbol{n})$ est démontrable dans le système[1].

Gödel considère alors une liste de 46 relations ou fonctions numériques (un grand nombre d'entre elles servent à construire les suivantes), parmi lesquelles on trouve Form (x) et Dém (x,y), et dont la dernière est Th (x). Il prouve que toutes sont représentables dans S, excepté la dernière, Th (x)[2].

Le rôle des fonctions et relations (primitives) récursives

Pour prouver qu'une fonction ou relation est représentable dans S, Gödel utilise une méthode indirecte. Il définit d'abord la classe des *fonctions* numériques primitives récursives (qu'il appelle simplement « récursives » dans l'article de 1931) ainsi que la notion de *relation* primitive récursive[3] et prouve ensuite que toute fonction ou relation primitive récursive est représentable dans le système S. Pour établir qu'une relation numérique qui exprime une propriété métamathématique (par exemple Form (x)) est représentable *dans* S, il suffit alors de montrer qu'elle est primitive récursive. Et Gödel prouve effectivement que les 45 premières fonctions ou relations énumérées sont bien primitives récursives, établissant ainsi, notamment, qu'il existe deux expressions formelles (qu'on notera respectivement **Form** $[x]$ et **Dém** $[x,y]$) capables de repré-

1. Ou, si l'on préfère, $f(m) = n$ si et seulement si les formules $A[\boldsymbol{m},\boldsymbol{n}]$ et $\exists!yA[\boldsymbol{m},y]$ sont l'une et l'autre formellement démontrables. Dans les deux cas, on veut exprimer que pour un entier m donné, il existe un entier n et un seul tel que $A[\boldsymbol{m},\boldsymbol{n}]$.
2. Le prédicat Th (x) peut être *traduit* en une formule de S (qu'on pourra noter **Th** $[x]$) sans être *représentable dans* S au sens précis qui vient d'être défini.
3. Ici encore, « primitive récursive » selon l'usage actuel, correspond à « récursive » selon l'usage de Gödel en 1931.

senter dans S les prédicats numériques Form(x) et Dém(x,y). Parmi les relations considérées, seul le prédicat Th(x) n'est pas primitive récursif[1].

Le premier théorème d'incomplétude

Gödel peut alors prouver qu'il existe une formule A formellement indécidable, c'est-à-dire telle que ni A ni sa négation, ¬ A, ne soient démontrables dans le système, la démonstration ne valant pas seulement pour le système initialement considéré, mais pour tout système $S_\varkappa$ résultant de S par l'ajout d'un ensemble primitif récursif d'axiomes. Le premier théorème d'incomplétude énonce que si $S_\varkappa$ est consistant, alors A est formellement indécidable[2]. Dans la démonstration, l'un des moments cruciaux est la preuve qu'il existe une relation (numérique) primitive récursive, qu'on notera R(x, k), exprimant la propriété métamathématique sui-

1. La classe PR des fonctions primitives récursives de N^k dans N (où N est l'ensemble des entiers naturels) peut être définie par les clauses suivantes :
(i) les fonctions constantes et la fonction successeur sont (dans l'ensemble) PR ;
(ii) les fonctions définies par composition de fonctions PR sont PR (par exemple la fonction f définie par $f(x_1, x_2, x_3) = g(h_1(x_2, x_2), h_2(x_3), h_3(x_3, x_2, x_1))$, où g, h_1, h_2 et h_3 sont PR ;
(iii) les fonctions définies selon le schéma de récursion primitive à partir de deux fonctions PR sont PR.

On dit qu'une fonction $f(x_1, \ldots, x_n)$ est définie selon le schéma de récursion primitive à partir des fonctions $g(x_1, \ldots, x_{n-1})$ et $h(x_1, \ldots, x_{n+1})$ si l'on a :

$$f(0, x_2, \ldots, x_n) = g(x_2, \ldots, x_n) \text{ et}$$
$$f(n+1, x_2, \ldots, x_n) = h(n, f(n, x_2, \ldots, x_n), x_2, \ldots, x_n).$$

On dit qu'une relation est primitive récursive, ou qu'un ensemble est primitif récursif, si sa fonction caractéristique est primitive récursive.

2. Dans le mémoire de 1931, Gödel supposait une condition plus forte que la consistance de $S_\varkappa$, à savoir « $S_\varkappa$ est ω-consistant ». En 1936, Rosser montrait qu'en choisissant une autre formule que A, plus complexe, on pouvait se contenter de la condition la plus faible, savoir la consistance simple du système. La formule A qui est considérée dans les lignes qui suivent suppose donc en réalité l'ω-consistance de $S_\varkappa$. Pour un exposé plus détaillé et plus précis de la preuve, cf. par exemple Blanché et Dubucs, *op. cit.*, Gödel, *op. cit.*, 1986, ou Stephen C. Kleene, *Introduction to metamathematics*, Amsterdam, North-Holland, 1952, § 42.

vante : si k est le nombre de Gödel d'une formule à une variable libre, qu'on note $F_k[\dot{l}]$, x n'est pas le nombre de Gödel d'une preuve de $F_k[\boldsymbol{k}]$, où $\boldsymbol{k}$ est la représentation formelle de l'entier k. $R(x, k)$ étant une relation primitive récursive, elle est représentable dans S_x par une formule $\mathbf{R}[x, k]$. Considérons maintenant la formule $\forall x\mathbf{R}[x, k]$, et appelons u son nombre de Gödel. Alors, la formule $\forall x\mathbf{R}[x, \boldsymbol{u}]$ exprime dans le système l'idée métamathématique suivante : il n'existe aucune preuve de $\forall x\mathbf{R}[x, \boldsymbol{u}]$, c'est-à-dire : je suis une formule indémontrable. Il reste alors à prouver que cette formule est formellement indécidable, c'est-à-dire que ni $\forall x\mathbf{R}[x, \boldsymbol{u}]$ ni $\neg\forall x\mathbf{R}[x, \boldsymbol{u}]$ ne sont démontrables. Gödel le fait en montrant que dans l'un et l'autre cas, l'hypothèse de (ω-) × consistance serait contredite.

ω - consistance

Le second théorème d'incomplétude

L'énoncé métamathématique exprimant la (ω-)consistance de S peut être traduit par une formule du système S lui-même, que l'on notera **Cons**[1]. De plus, Gödel remarque que les moyens démonstratifs utilisés pour la preuve du premier théorème d'incomplétude ne dépassent pas ceux dont on dispose dans l'arithmétique élémentaire. Il est donc possible de formaliser dans S_x la preuve de la formule suivante : $\mathbf{Cons} \rightarrow \forall x\mathbf{R}[x, \boldsymbol{u}]$, qui est l'expression d'une partie du premier théorème d'incomplétude (puisque $\forall x\mathbf{R}[x, \boldsymbol{u}]$ signifie : $\forall x\mathbf{R}[x, \boldsymbol{u}]$ n'est pas démontrable). Par conséquent, si **Cons** était démontrable, $\forall x\mathbf{R}[x, \boldsymbol{u}]$ le serait aussi, contrairement à ce qu'énonce le premier théorème. Le second théorème d'incomplétude s'énonce donc : si S_x est (ω-)consistant, alors la formule **Cons** qui affirme cette propriété de S_x est formellement indémontrable dans ce système.

1. Pour la consistance, cette formule exprime, via le codage numérique des expressions formelles, l'idée suivante : il existe une formule qui n'a pas de preuve dans S_x, ce que l'on peut écrire $\exists x(\mathbf{Form}[x] \wedge \forall y \neg \mathbf{Dém}[y, x])$. L'expression de l'ω-consistance est un peu plus complexe.

Que résulte-t-il de ces deux théorèmes et de leur démonstration ? Ici, la question ne vise ni la problématique des fondements des mathématiques, ni la philosophie de l'esprit. On demande plutôt quels sont les liens qui se nouent entre le calcul et la démonstration et donc, indirectement, entre les calculateurs ou machines informatiques et la logique. La preuve des théorèmes d'incomplétude donne en effet aussi un premier éclairage sur la question posée au début du chapitre précédent, celle des rapports entre théorie de la calculabilité et théorie de la démonstration. On reprendra donc maintenant, pour les lire à la lumière des théorèmes d'incomplétude et de leur preuve, les idées présentées ci-dessus comme des prolongements du programme de Hilbert, explicités par Gödel en 1931.

LES THÉORÈMES D'INCOMPLÉTUDE COMME PRÉALABLES A L'INTRODUCTION EN LOGIQUE D'UNE IDÉE DE MACHINE

Logique et opérations numériques : le codage des expressions formelles et l'arithmétisation des énoncés métamathématiques

En arithmétisant les énoncés métamathématiques, Gödel met en œuvre de manière rigoureuse l'idée qu'un certain nombre de problèmes logiques peuvent être réduits à des questions de calcul numérique. Par exemple, pour déterminer si l'énoncé A est un théorème du système S, il suffit de calculer son nombre de Gödel, n, et de trouver un entier k vérifiant la relation numérique Dém (k, n). Dans certains cas particuliers, on peut montrer que l'un des deux énoncés A et ¬ A est démontrable. Notons m le code de A (son nombre de Gödel) et n celui de ¬ A. Pour déterminer lequel des deux énoncés est un théorème, il suffit d'énumérer les entiers k qui codent les démonstrations formelles, et de déterminer par un

calcul numérique, pour chacun d'eux, s'il vérifie Dém (k, m) ou Dém (k, n). Ou encore, les énoncés syntaxiquement dérivables à partir d'un ensemble de formules, $\varkappa$, peuvent être connus par un calcul arithmétique en énumérant de manière systématique les nombres de Gödel des démonstrations du système $S_{\varkappa}$, afin d'en déduire par un autre calcul quels sont les théorèmes de $S_{\varkappa}$. Un calculateur numérique convenablement programmé ou, par exemple, une machine de Turing, sont donc capables de résoudre ces problèmes logiques, lorsqu'ils possèdent une solution[1].

Gödel met donc en vedette l'idée de codage, dont on connaît la fortune en informatique et en théorie de la calculabilité : les tableaux de commande des machines de Turing sont codés avant d'être inscrits sur le ruban de la machine universelle, les entiers et les fonctions numériques sont codés par des termes du lambda-calcul, les données, quelles qu'elles soient, sont codées avant d'être enregistrées dans un système informatique et traitées selon les instructions d'un programme, etc. C'est aussi par codage que l'on établit l'équivalence entre les machines de Turing, les lambda-termes, les fonctions récursives, les systèmes canoniques de Post, les calculateurs universels à programmes enregistrés, et autres machines abstraites ou physiquement réalisées. L'importance de l'idée de codage vient donc aussi de ce qu'il est toujours possible, pour effectuer un calcul, de se placer dans un autre système de signes, au moins tant que l'on néglige les propriétés par lesquels se distinguent les différents choix syntaxiques possibles (tel système formel pour l'arithmétique, le lambda-calcul, les machines de Turing, etc.) ; le seul véritable privilège que possède un codage *numérique* vient de ce qu'il bénéficie des nombreux résultats connus en arithmétique.

1. « (...) étant donné la représentation par les nombres de Gödel et les idées qui lui sont associées, la logique symbolique en général peut être considérée, mathématiquement, comme une branche de la théorie élémentaire des nombres. Cette remarque est due à Hilbert (...) mais c'est dans les termes de la représentation par les nombres de Gödel qu'elle est formulée le plus clairement » (Church, An Unsolvable Problem of Elementary Number Theory, *in* Davis, *The Undecidable,* p. 94).

Mais en utilisant la puissance technique du codage, Gödel marque du même coup les limites de toute tentative pour mécaniser le raisonnement par la voie des systèmes formels ou par un quelconque moyen équivalent. Car le codage et la réduction de certaines questions logiques traditionnelles à un calcul arithmétique se payent au prix fort des contraintes qu'ils imposent. Ou plutôt, ils révèlent les limites inhérentes à toute conception semblable du calcul logique. Et de ce point de vue, la preuve des théorèmes d'incomplétude se présente comme la toute première étape d'une étude des capacités et incapacités des machines, menée dans le cadre logique de la théorie de la démonstration. Mais voyons plutôt quelles sont ces conséquences contraignantes que les travaux de Gödel révèlent ou laissent présager.

(i) La réduction d'un problème logique de raisonnement à une opération de calcul numérique suppose que le problème en question puisse passer l'épreuve de la formalisation, et donc que ses termes puissent être exprimés par une suite finie de signes dans un système défini par son ordre syntaxique. En effet, si l'on ne se limite pas aux questions abstraites de logique pure, il faut bien formaliser le contenu des raisonnements que l'on espère réduire à un calcul, conformément à l'exigence frégéenne de l'expression d'un *contenu conceptuel* dans l'idéographie. Or un grand nombre des tentatives menées pour construire des machines capables de raisonner buttent précisément sur la formalisation du contenu conceptuel des raisonnements. C'est surtout à partir du moment où l'on tenta de donner une forme concrète à l'ambitieux projet de l'intelligence artificielle que l'on put mesurer l'ampleur des obstacles qui se présentent lorsque l'on s'efforce de formaliser des raisonnements quelconques, hors des domaines logique et mathématique. Leibniz avait, à sa façon, bien mesuré la difficulté puisqu'il posait, comme condition préalable à la réalisation du *calculus ratiocinator*, la conception d'une langue caractéristique universelle, et par là également, d'une encyclopédie conçue comme la mise en ordre logique de toutes nos connaissances. C'est pourquoi, au moment où Gödel ouvre la voie de la réduction de questions logiques (c'est-à-dire, ici, relatives au

raisonnement et à la déduction) à des calculs arithmétiques (et donc susceptibles d'être effectués par une machine à calculer), il introduit aussi les moyens d'évaluer les difficultés d'une telle réduction lorsque l'on prétend lui donner une réalisation effective et la généraliser à des raisonnements quelconques. Alors seulement pouvait être posée la question d'une mesure des capacités de calcul en général. C'est pour répondre à cette question que se constitue progressivement une théorie de la calculabilité, et par là, un examen systématique des capacités des machines.

(ii) La seconde conséquence découle de l'incomplétude des systèmes formels pour l'arithmétique, qui impose une nouvelle définition du calcul. Supposons que l'on veuille déterminer laquelle des deux formules, A et sa négation ¬ A, est démontrable dans un système donné. S'il est vrai que pour ce problème la recherche d'une solution peut prendre la forme d'un calcul numérique, il convient de prendre en considération, dans le cas général, la possibilité que A soit formellement indécidable, et par conséquent, que le calcul se poursuive indéfiniment sans jamais produire le résultat recherché, et sans que rien ne permette de dire si ce résultat pourra être trouvé après un nombre fini d'opérations de calcul. Si la réduction de questions logiques à un calcul numérique est possible, elle suppose donc que l'on étende la notion de calcul à la recherche indéfinie, quoique parfaitement déterminée dans son procès, d'un résultat inexistant. En 1931, on pouvait encore espérer qu'il existât une procédure générale, à laquelle personne n'avait encore pensé, permettant de déterminer au sujet de A et de ¬ A si l'une ou l'autre est démontrable. Or c'est précisément cette possibilité qui est fermée en 1936 par les solutions négatives qu'apportent Church et Turing au problème de la décision.

(iii) Enfin, si l'arithmétisation du calcul logique établit un rapport entre des questions logiques de raisonnement, de déduction ou de décision et les calculateurs numériques, il faut bien dire que la possibilité de découvrir un théorème mathématique ou de résoudre une question logique dignes d'intérêt en utilisant de cette manière les ressources opératoires d'une machine reste, même aujourd'hui,

et *a fortiori* dans les années trente, hautement théorique et très limitée. En 1931, les problèmes considérés par Gödel étaient purement logiques et théoriques : le codage numérique des expressions formelles n'avait pas plus que les machines de Turing en 1936 vocation à être appliqué à des cas concrets. Dans cette perspective, la théorie de la calculabilité pouvait déterminer s'il existe ou non un algorithme permettant de résoudre tel problème, si telle propriété était décidable, ou si deux formalismes caractérisaient ou non la même classe de fonctions calculables ; mais il n'était jamais question de savoir si l'on pouvait attendre de tel ou tel algorithme bien déterminé la solution de problèmes concrets par des moyens humainement ou même physiquement réalisables. En d'autres termes, la théorie de la calculabilité, comme l'arithmétisation du calcul logique par Gödel, faisait entièrement abstraction de contraintes physiques comme la durée des calculs et l'espace mémoire requis pour qu'une machine puisse les effectuer. L'idée d'un tel usage des calculateurs devait attendre la réalisation physique des machines universelles ; elle devait aussi rencontrer les problèmes logiques et informatiques relatifs à la complexité des preuves et des programmes. A tout le moins, le principe d'une étude logique des calculs et des calculateurs était posé. Elle pouvait, plus tard, se doter d'outils plus fins pour la comparaison de différents algorithmes capables de calculer une seule et même fonction, et se muer en une théorie de la complexité ; celle des programmes, en informatique, et, parallèlement, celle des preuves en théorie de la démonstration, affirmant ainsi plus nettement encore l'importance des rapports entre algorithmes et preuves, et donc celle de la notion de machine en logique.

Logique et effectivité : la définition des fonctions récursives

Gödel introduit la classe des fonctions primitives récursives, qu'il nomme simplement « récursives » dans l'article de 1931. Même si l'idée de définition récursive d'une fonction n'est pas nouvelle (on la

trouve par exemple chez Dedekind en 1888, Skolem en 1923, Hilbert en 1925 ou Ackermann en 1928), cet aspect de la démonstration fait pourtant lui aussi des théorèmes d'incomplétude un moment décisif pour l'élaboration à venir d'une théorie de la calculabilité, et pour l'introduction ultérieure de la notion de machine en logique.

Premièrement, parce que quelques années plus tard, dans un nouvel exposé des théorèmes d'incomplétude, Gödel suggère une généralisation de la classe des fonctions primitives récursives qui se révèle être équivalente à celle des fonctions lambda-définissables ou Turing-calculables. En effet, dans les conférences qu'il donna en 1934 à l'Institute for Advanced Study, Gödel remarquait que

> les fonctions récursives ont la propriété importante que, pour chaque ensemble de valeurs donné aux arguments, la valeur de la fonction peut être calculée par une procédure finie[1].

Et il ajoutait en note :

> L'inverse semble être vrai si, outre les récursions conformes au schéma (2), on admettait d'autres formes de récursion (par ex., sur deux variables simultanément)[2].

Or au § 9 du même texte, Gödel introduit la notion de fonction *générale* récursive, qui étend la classe des fonctions primitives récursives, et dont Kleene montra, dans un texte publié en 1936, qu'elle définit une classe de fonctions équivalente à celle des fonctions lambda-définissables. On sait en outre que c'est dans les termes de cette classe de fonctions récursives généralisées que la fameuse thèse de Church fut énoncée pour la première fois. Bien qu'en 1934 Gödel n'ait pas été lui-même convaincu par l'équivalence entre fonctions calculables par une procédure finie et fonctions récursives[3], on voit que les travaux de Gödel sur les théorèmes d'incom-

1. On undecidable propositions of formal mathematical systems, 1934, *in* Gödel, *op. cit.*, 1986, p. 348.
2. *Ibid.* Le schéma (2) correspond à la définition récursive classique, sur une variable (voir le schéma de récursion primitive, ci-dessus, n. 1, p. 117).
3. Dans la citation qui précède, il suggère seulement l'équivalence entre les fonctions calculables « par une procédure finie » et « d'autres formes de récursion », sans affirmer que les fonctions générales récursives qu'il introduit au paragraphe 9 couvrent effectivement tous les cas de définition par récursion. Sur ce point, cf. Davis, *op. cit.*, 1982.

plétude ont joué un rôle décisif pour la caractérisation mathématique de la notion de procédure effective.

Deuxièmement, parce que inversement, les théorèmes d'incomplétude ne pouvaient pas être énoncés dans toute leur généralité avant qu'une caractérisation satisfaisante de l'effectivité n'eût été énoncée. Et l'on sait qu'en ce qui concerne la thèse de Church, ce sont les machines de Turing qui emportèrent l'adhésion de Gödel[1], comme l'indique le post-scriptum suivant ajouté, trente ans plus tard, aux conférences de 1934 :

> En conséquence de progrès ultérieurs, en particulier du fait que, grâce au travail de A. M. Turing, une définition précise et indiscutable du concept général de système formel peut maintenant être donnée, l'existence de propositions arithmétiques indécidables et la non-prouvabilité de la consistance d'un système dans ce même système peuvent maintenant être démontrées rigoureusement pour *tout* système formel consistant contenant une certaine quantité de théorie élémentaire [*finitary*] des nombres[2].

En 1931, Gödel démontrait les théorèmes d'incomplétude pour un système formel particulier, ainsi que pour tous les systèmes obtenus par ajout d'un ensemble récursif d'axiomes, et indiquait comment les mêmes résultats pouvaient être obtenus au sujet d'autres systèmes connus, comme ceux de Zermelo-Frænkel et de von Neumann pour la théorie des ensembles[3]. Au § 6 du texte de 1934, il précise quelles sont les conditions générales qu'un système doit satisfaire pour que sa démonstration des théorèmes d'incomplétude puisse lui être appliquée. Mais on ne pouvait pas encore affirmer pour autant l'incomplétude de tous les systèmes formels possibles répondant aux conditions requises dans l'énoncé des deux théorèmes. Pour qu'une preuve aussi générale pût être donnée, il fallait disposer d'une caractérisation des systèmes formels qui ne dépendît d'aucun trait syntaxique singulier. La solution de cette difficulté vint de Kleene, en 1943, qui s'appuyait sur l'idée suivante : pour

1. Sur ce point, la référence essentielle est encore Davis, *ibid.*
2. Gödel, *op. cit.*, p. 369. Voir aussi la suite du post-scriptum citée ici même dans l'Introduction, p. 3.
3. Cf. Gödel, *op. cit.*, p. 181.

qu'une construction syntaxique soit un système formel, il suffit que l'on puisse reconnaître par une *procédure effective* si une suite de signes quelconque est ou n'est pas la démonstration d'une formule A donnée[1]. D'où le post-scriptum de Gödel, qui attribue à Turing la caractérisation mathématique de l'effectivité et, par suite, la possibilité de définir le concept général de système formel.

On voit donc en quel sens les théorèmes d'incomplétude peuvent être considérés comme les prolégomènes d'une théorie de la calculabilité qui, en retour, confère à leur énoncé la plus grande généralité.

Ajoutons aussi que la récursivité fut amenée, ultérieurement, à jouer un rôle majeur en programmation. En effet, la définition récursive d'une fonction peut être considérée comme une manière, souvent très concise, d'écrire un programme pour le calcul de cette fonction[2].

Démonstration et calcul : les fonctions et les relations numériquement représentables

Il reste alors à considérer un troisième outil logique utilisé par Gödel pour la démonstration des deux théorèmes, savoir le plongement des énoncés métamathématiques dans le système formel lui-même, par la définition de ce qu'est une fonction *numériquement représentable*. Car cette technique définit un rapport précis entre calcul et démonstration.

Tout d'abord, pour les fonctions primitives récursives qui sont considérées par Gödel en 1931, le calcul est exactement équivalent à la recherche d'une démonstration, les deux démarches pouvant

1. Sur l'élaboration technique de cette idée, cf. Gödel, *op. cit.*, p. 344, et Kleene, *op. cit.*, § 60.
2. Soient par exemple les deux équations suivantes : (i) $\text{plus}(n, 0) = n$ et (ii) $\text{plus}(n, \text{succ}(m)) = \text{succ}(n, \text{plus}(n, m))$. Elles définissent récursivement l'addition à partir de la fonction successeur, et peuvent être utilisées comme un programme pour calculer, par exemple, 3 + 2. Si « p » est une abréviation de « plus » et « s » de « successeur », on écrira : $p(3, 2) = p(3, s(1)) = s(p(3, 1)) = s(p(3, s(0))) = s(s(p(3, 0))) = s(s(3)) = s(4) = 5$.

être achevées après un nombre fini d'opérations. En effet, si f est numériquement représentable par la formule $A[x,y]$, on a, pour tous les entiers m et n :

(i) $f(n) = m$ si et seulement si $\vdash A[\boldsymbol{m}, \boldsymbol{n}]$;
(ii) $f(n) \neq m$ si et seulement si $\vdash \neg A[\boldsymbol{m}, \boldsymbol{n}]$.

Calculer la valeur d'une fonction revient à *démontrer*, pour chacun des points de son domaine, qu'elle prend telle valeur.

Mais plus généralement, quelles sont les fonctions numériques qui peuvent être définies par la prouvabilité d'une formule dans un système formel pour l'arithmétique ? Considérons par exemple les fonctions unaires pour lesquelles il existe une formule $A[x,y]$ telle que, pour deux entiers m et n quelconques :

(i) $f(m) = n$ si l'on a à la fois $\vdash A[\boldsymbol{m}, \boldsymbol{n}]$ et $\vdash \exists!y A[\boldsymbol{m}, y]$;
(ii) $f(m)$ n'est pas défini s'il n'existe aucun n tel que $\vdash A[\boldsymbol{m}, \boldsymbol{n}]$ ou s'il existe plusieurs entiers n_i tels que $\vdash A[\boldsymbol{m}, \boldsymbol{n}_i]$.

On peut montrer que la classe des fonctions numériques ainsi définissables par la prouvabilité d'une formule dans un système formel est exactement la classe des fonctions partielles[1] qui sont calculables par une machine de Turing[2]. La notion de preuve formelle sert donc à définir l'idée logique de calcul, celle-là même qui donne une mesure des capacités des machines.

Mais inversement, la prouvabilité peut être comprise à partir de la notion de calcul, au sens des machines de Turing. Car, si l'on entend le mot « calcul » au sens général suivant : un algorithme qui, lorsqu'on l'applique à une donnée, peut ne conduire à aucun résultat après un nombre fini quelconque d'opérations, alors le prédicat $\mathrm{Th}(n)$ (c'est-à-dire : n est le nombre de Gödel d'un théorème) peut être calculé par une machine de Turing. Dans les termes des fonctions récursives, on dit que $\mathrm{Th}(n)$ est *semi-récursif* : les théorèmes

1. Voir chap. II, n. 2, p. 36.
2. Cf. Girard et Turing, *op. cit.*, p. 18-20 où Girard précise que la classe des fonctions partielles ainsi représentées dans un système formel T pour l'arithmétique auquel s'applique le théorème de Gödel « ne dépend pas (sauf cas controuvés en vue de thèse) du choix de T ».

peuvent être énumérés de manière systématique (on dit aussi que l'ensemble des théorèmes est récursivement énumérable), et si la formule A est un théorème, on le prouvera après un calcul fini ; en revanche, si A n'est pas un théorème, le même calcul ne permettra jamais de le savoir[1]. En ce sens, « on peut voir l'acte de démontrer comme le calcul d'une propriété semi-récursive »[2].

Démonstration et vérité : l'incomplétude

Les théorèmes d'incomplétude mettent en échec le programme de Hilbert. Premièrement, parce qu'il existe des énoncés formellement indécidables qui sont *élémentaires* au sens que Hilbert donnait à ce terme. Ainsi, la distinction entre des méthodes mathématiques *idéales* d'une part, et celles qui sont finitaires, élémentaires ou réelles d'autre part, perd beaucoup de son intérêt puisque l'incomplétude touche ces dernières, pourtant réputées les plus sûres. Il existe aussi des énoncés arithmétiques démontrables, mais dont la démonstration requiert nécessairement des méthodes abstraites, qui dépassent le cadre de l'arithmétique élémentaire. De ce fait, à moins, comme le souligne Gödel lui-même, que l'on conçoive des méthodes finitaires non formalisables dans l'arithmétique élémentaire, c'est l'idéal de conservativité ou de pureté des méthodes qui est réduit à néant. Deuxièmement, parce que le second théorème d'incomplétude prouve que l'idéal hilbertien d'une preuve de consistance de l'arithmétique dans l'arithmétique (élémentaire) est vain lui aussi. Enfin, troisièmement, parce qu'il existe des propositions formellement indécidables dont on peut reconnaître la vérité dans le métalangage, comme la formule $\forall x \mathbf{R}[x, \boldsymbol{u}]$ par exemple. De ce fait, les résultats de Gödel viennent rompre l'équilibre entre les expressions formellement démontrables et les énoncés vrais, niant l'équivalence

1. Ce qui n'exclut pas qu'on puisse le savoir d'une autre manière, par exemple si l'on prouve $\neg$ A et que le système est consistant.
2. Girard et Turing, *op. cit.*

entre ces deux notions, que le programme de Hilbert se proposait d'établir.

Pour autant, tous les effets de cette mise en échec du programme de Hilbert ne sont pas nécessairement négatifs. Plutôt qu'un abandon, les théorèmes d'incomplétude demandaient une réorientation de la théorie de la démonstration[1]. En niant la possibilité de réduire à des méthodes élémentaires toutes les preuves d'énoncés élémentaires, ils introduisent l'idée – qui se révélera extrêmement féconde en logique – d'une hiérarchie des formules et des méthodes, et corrélativement, d'une théorie de la complexité logique, les différents systèmes formels possibles étant ordonnés selon leur puissance démonstrative et leurs propriétés remarquables, syntaxiques ou sémantiques[2]. Ainsi, sachant que l'idéal hilbertien de conservativité ne saurait être entièrement satisfait, on peut demander quels sont les énoncés des mathématiques abstraites qui sont susceptibles de recevoir une preuve « élémentaire » ; ou plutôt, parce que l'opposition hilbertienne entre « idéal » et « élémentaire » ne suffit plus, comment caractériser les preuves d'un système donné qui peuvent être *converties* en démonstrations d'un système de moindre complexité, et *a contrario*, celles qui résistent à une telle transformation. Cet idéal de convertibilité, qui existe déjà chez Hilbert, peut donc être réinterprété[3]. On pourra demander, en outre, si cette convertibilité peut être effectuée par des procédures effectives, et à quelles conditions. Par là, des questions de transformation, ou de manipulation de démonstrations posent des problèmes de calcul ; mieux : ces opérations sur les preuves peuvent prendre elles-mêmes la forme de calculs, comme on le verra en examinant la correspondance de Curry-Howard entre les preuves et les programmes.

Ces quelques remarques suffisent à montrer en quel sens les résultats obtenus par Gödel rendent possible une orientation de la

1. Sur l'interprétation philosophique et les conséquences des théorèmes d'incomplétude en mathématiques, voir le dernier chapitre de Blanché et Dubucs, *op. cit.*
2. Cf. Girard, *Proof theory and logical complexity*.
3. En 1935, Gentzen lui donnera une signification précise dans le cadre de la déduction naturelle et du calcul des séquents. Ce point sera précisé au chapitre VIII.

logique qui abandonne l'idée d'une présentation, fût-elle systématique, des vérités appartenant à une certaine catégorie, dite « logique ». Se trouve ainsi mise en valeur l'étude de systèmes syntaxiques de preuves dont on s'efforce de montrer qu'ils jouissent de telles ou telles propriétés. De là l'usage largement répandu de l'article indéfini devant le mot « logique », au singulier ou au pluriel, qui ne signifie pas pour autant, loin s'en faut, que l'on renonce à concevoir *la* logique. Il indique seulement que le travail des logiciens n'est pas nécessairement orienté par l'idéal d'une *vérité* logique, et que l'on peut s'intéresser aux propriétés d'*une* logique, c'est-à-dire d'un système formel particulier. Et si l'on se souvient maintenant de l'équivalence entre machines, programmes et certains formalismes abstraits définis par des règles de syntaxe, on ne doit plus du tout trouver étrange que de nombreux aspects de la logique contemporaine puissent être considérés comme l'étude des machines, autant que celle des calculs ou des preuves. Malgré cela, on ne pourra saisir suffisamment bien la portée des relations entre logique et informatique qu'après avoir examiné la correspondance de Curry-Howard. Mais avant d'aborder ce point, qui justifiera la recherche ultérieure d'une problématique philosophique prenant pleinement en considération l'idée de machine en logique, considérons maintenant l'usage bien particulier que la philosophie de l'esprit et la recherche d'une science des phénomènes psychiques peut faire des notions et des résultats qui ont été analysés jusqu'ici.

VII

CALCUL, MACHINE ET PHILOSOPHIE DE L'ESPRIT

Perspective historique

Le principal intérêt que les philosophes reconnaissent aux notions de machine et de calcul telles qu'elles sont élaborées en logique vient de l'usage qu'ils jugent légitime d'en faire pour l'examen de certains problèmes relevant de la philosophie de l'esprit. Cet intérêt s'est surtout développé à partir des années cinquante, au début de la révolution informatique qui met progressivement à la disposition d'un plus grand nombre d'usagers des ordinateurs aux performances techniques croissantes et des langages ou des programmes plus aisément soumis aux tâches qu'il est possible de leur confier. La réalisation de machines douées de capacités qui étaient naguère l'apanage de l'esprit humain suggère aux yeux de certains qu'elles pourraient aider à comprendre nos facultés de connaissance et à modéliser les phénomènes psychiques ou cognitifs. Nombreux sont ceux qui pensent même apercevoir dans les progrès de l'informatique et ses fondements logiques la possibilité de concevoir une philosophie de l'esprit capable d'éclairer les rapports entre l'esprit et le corps ou le problème de la nature des états mentaux. Pour naïf que puisse paraître l'espoir de résoudre des problèmes philosophiques aussi aporétiques par de tels moyens, il fut parfois nourri par ceux qui crurent réalisables les projets d'intelligence artificielle conçus en 1956 et au cours des années qui suivirent. On sait aujourd'hui que la volonté de programmer une forme générale d'intelligence, susceptible d'être appliquée à

la résolution de problèmes très divers, rencontra des obstacles qui se révélèrent insurmontables, et qu'il fallut réorienter la recherche vers des travaux moins ambitieux comme la programmation de facultés plus limitées et de tâches restreintes à des domaines précis et soigneusement circonscrits. Pourtant, les réussites étaient parfois spectaculaires[1], et plusieurs philosophes pensaient que les capacités de l'esprit pourraient être réduites à des opérations de calcul comparables au traitement informatique des données. Les références à Hobbes (« Per Ratiocinationem autem intelligo computationem », *De corpore*) ou à Leibniz :

> Quo facto, quando orientur controversiae, non magis disputatione opus erit inter duos philosophos, quam inter duos Computistas. Sufficiet enim calamos in manus sumere sedereque ad abacos, et sibi mutuo (accito si placet amico) dicere : *calculemur.*[2]

confirmaient que l'idée d'une réduction des raisonnements et des opérations de l'intelligence à des calculs pouvaient avoir un intérêt majeur pour la réflexion philosophique. Les idées de Hobbes et de Leibniz (ou du moins celles que l'on retenait) paraissaient d'autant plus fortes que l'on pouvait maintenant se recommander d'une reformulation de la thèse de Church-Turing en termes de machines informatiques réelles (n'importe quel calcul peut être effectué par un ordinateur – au moins en principe)[3] pour soutenir une forme nouvelle de philosophie mécaniste qui réduit les opérations de l'esprit (ou certaines d'entre elles) à des calculs, et les calculs eux-mêmes aux opérations d'une machine.

1. On trouvera quelques exemples de ce que l'intelligence artificielle est capable de faire aujourd'hui dans l'ouvrage de Jean-Gabriel Ganascia, *L'intelligence artificielle,* Paris, Flammarion, 1993. Une réflexion philosophique générale sur les limites de l'intelligence artificielle est développée par Hubert Dreyfus dans *What Computers Can't Do. The limits of Artificial Intelligence,* 1972, 2e éd., 1979, trad. franç., *Intelligence artificielle. Mythes et limites,* Paris, Flammarion, 1984 ; nouv. éd., *What Computers* Still *Can't Do. A Critique of Artificial Intelligence,* Cambridge, Mass., The MIT Press, 1992.
2. « Cela étant fait, lorsque des controverses naîtront, la dispute ne sera pas plus nécessaire entre deux philosophes qu'entre deux calculateurs. Il suffira en effet de prendre la plume, de s'asseoir devant une abaque, et de se dire mutuellement (en présence d'un ami si l'on veut) : *calculons* » (*G.* VII, p. 200).
3. C'est-à-dire si l'on ignore les limites physiques et techniques que rencontre la complexité des calculs.

L'échec, sur le plan empirique, des premiers projets d'intelligence artificielle et, *a contrario*, la mécanisation réussie de certaines formes de raisonnement, la théorie de la calculabilité et des machines abstraites récemment élaborée en logique, les thèses mécanistes et antimécanistes relatives aux phénomènes de l'esprit et à la pensée, tous ces éléments appelaient une réflexion philosophique générale sur les capacités des machines comparées à celles de l'esprit, et sur l'usage paradigmatique, en philosophie de l'esprit, de la notion de calcul telle qu'elle est comprise en logique et en informatique. Loin d'être aujourd'hui résolues, les questions posées par cette problématique « computationnelle » fondée sur le calcul, le modèle informatique et les machines, ne laissent pas de susciter la réflexion philosophique : Une machine peut-elle penser ? Les machines informatiques offrent-elles un modèle mécaniste pour la compréhension de la pensée, des capacités cognitives ou des phénomènes de l'esprit en général ? Une psychologie cognitive fondée sur de tels modèles est-elle possible ? Le rapport entre machines abstraites et machines réelles pourrait-il éclairer la question philosophique des relations entre les états mentaux et les états physiques du corps ? Que nous apprennent les théorèmes d'incomplétude de Gödel sur les capacités des machines, et, par comparaison, sur les capacités humaines de penser ? etc.

Toutes ces difficultés apparurent progressivement à partir des années cinquante[1] et suscitèrent une réflexion philosophique dont les termes se sont parfois modifiés à la lumière des progrès ou des échecs enregistrés par l'intelligence artificielle sur les plans théorique et empirique, même si l'évolution des machines, des programmes et des idées scientifiques ou techniques est souvent d'une interprétation très

1. Le texte inaugural de cette problématique est certainement l'article que Turing publia en 1950, *Computing Machinery and Intelligence* (voir ci-dessous, n. 1, p. 135). Il existe avant 1950 d'autres textes philosophiques sur les machines (comme l'article de Turing, Intelligent Machinery, *Report to the National Physical Laboratory*, 1947, reproduit dans Meltzer et Michie (ed.), *Machine Intelligence*, vol. 5) mais ils appartiennent plutôt à d'autres problématiques, comme celle de la cybernétique. Ce qui caractérise en propre la philosophie de l'informatique et de l'intelligence artificielle sera précisé dans les pages qui suivent.

délicate. On voit ainsi comment les machines de Turing, la thèse de Church ou les travaux de Gödel peuvent être sollicités par le philosophe lorsqu'il s'interroge sur l'esprit, la pensée, la psychologie. De ce point de vue, ce qui suscite sa réflexion n'est pas du tout l'importance accordée à la notion de machine par la logique, discipline traditionnellement conçue comme étant relative à la vérité, la pensée, la connaissance, ou le langage ; il s'agit plutôt de savoir quel usage légitime on pourra faire de certains outils et résultats logiques dans le domaine de la philosophie de l'esprit[1]. On voudrait maintenant donner un aperçu de la manière dont sont utilisés, dans cette perspective, la thèse de Turing, l'équivalence entre machines abstraites, programmes et ordinateurs, ainsi que les théorèmes d'incomplétude. Les réflexions philosophiques relatives à l'esprit, et fondées sur l'idée de machine que l'on trouve en logique et en informatique, ne feront pas ici l'objet d'un examen détaillé. Il s'agit seulement de dégager certains traits caractéristiques généraux de cette problématique bien particulière afin de préciser en quel sens et pourquoi une autre perspective philosophique sur les machines est possible[2].

1. Dans ce domaine, l'importance de la logique ne se limite cependant pas à l'usage des notions conçues et des résultats établis en théorie de la calculabilité. Bien que nous ne développions pas ici cet aspect du problème, des logiques non classiques comme la logique modale et les logiques dites « déviantes » ou « logiques pour l'intelligence artificielle » sont importantes pour cette perspective computationnelle sur la philosophie de l'esprit. Cf. par exemple Dov Gabbay et F. Guenthner (ed.), *Handbook of Philosophical Logic,* Dordrecht, D. Reidel, vol. 1 à 4, 1983-1988, et R. Turner, *Logics for Artificial Intelligence,* Chichester, NY, Ellis Horwood, 1984, trad. franç., *Logiques pour l'intelligence artificielle,* Paris, Masson, 1986.
2. Ce chapitre n'étudie de manière approfondie ni l'histoire de l'informatique et de l'intelligence artificielle dans son rapport à la philosophie, ni les arguments mécanistes et antimécanistes en philosophie de l'esprit. Pour une analyse plus précise de ces différents points, on pourra consulter, par exemple, les textes suivants : Alan Ross Anderson, *Minds and machine,* Englewood Cliffs, NJ, Prentice-Hall, 1964, trad. franç., *Pensée et machine,* Seyssel, Champ Vallon, 1983 ; Daniel Andler (dir.), *Introduction aux sciences cognitives,* Paris, Gallimard, 1992 ; Margaret Boden (ed.), *The Philosophy of Artificial Intelligence,* Oxford, Oxford University Press, 1990 ; Dreyfus, *op. cit.* ; Ganascia, *op. cit.* ; Jean-Gabriel Ganascia, *L'âme-machine. Les enjeux de l'intelligence artificielle,* Paris, Éditions du Seuil, 1990 ; John Haugeland, *Artificial Intelligence. The Very Idea,* Cambridge, Mass., Londres, The MIT Press, 1985, trad. franç., *L'esprit dans la machine. Fondements de l'intelligence artificielle,* Paris, Odile Jacob, 1989 ; Pierre Wagner, *Machine et pensée : l'importance philosophique de l'informatique et de l'intelligence artificielle,* thèse de l'Univ. de Paris I, 1994 ; Judson C. Webb, *Mechanism, Mentalism, and Metamathematics. An essay on finitism,* Dordrecht, D. Reidel, 1980.

MACHINES LOGIQUES ET MACHINES INFORMATIQUES EN PHILOSOPHIE DE L'ESPRIT ET EN PSYCHOLOGIE COGNITIVE

La thèse de Turing et le jeu de l'imitation

En 1950, dans un article sur les ordinateurs et l'intelligence[1], six ans avant la création de l'expression « intelligence artificielle » et l'élaboration du programme qu'elle désigne, c'est Turing lui-même qui fonde sur ses machines abstraites une comparaison entre l'homme et la machine, introduisant ainsi l'usage de quelques outils forgés en théorie de la calculabilité dans une réflexion philosophique relative à la pensée. Considérant que la question « une machine pourrait-elle penser ? » a trop peu de sens pour mériter une discussion, Turing la remplace par un test fondé sur « le jeu de l'imitation ». Supposons qu'un interrogateur utilise un téléscripteur pour communiquer avec un ordinateur et un être humain qu'il ne distingue l'un de l'autre, au début du jeu, que par deux dénominations conventionnelles, A et B. En posant à A et à B les questions qu'il juge pertinentes et en considérant les réponses qu'il reçoit, il cherche à déterminer lequel des deux est l'être humain. L'ordinateur est programmé de manière à tromper l'interrogateur en lui faisant croire qu'il n'est pas une machine mais un homme, alors que la personne interrogée cherche au contraire à faire connaître la vérité à celui qui pose les questions. En 1950, Turing estime qu'il suffira d'une cinquantaine d'années pour que des machines réelles fassent bonne figure à ce jeu et

> pour les faire si bien jouer (...) qu'un interrogateur moyen n'aura pas plus de 70 % de chances de procéder à l'identification exacte après cinq minutes d'interrogation[2].

1. Computing Machinery and Intelligence, *Mind*, vol. LIX, 236, 1950. L'article est reproduit dans Anderson, *op. cit.* ; il est également traduit dans Girard et Turing, *op. cit.*
2. Anderson, *op. cit.*, p. 49 (trad. franç.).

Il répond aussi successivement à neuf arguments opposés à son opinion, selon laquelle à la fin du siècle, « on pourra parler de machines pensantes sans s'attendre à être contredit »[1]. L'objet de l'article dans son ensemble est à la fois de montrer qu'aucune impossibilité de principe ne vient s'opposer à l'idée d'une machine capable de penser et de rendre vraisemblable la possibilité d'une telle machine. Sur cette dernière question, l'argumentation fait fond sur trois points :

I / La thèse de Turing appliquée aux machines réelles : tout calcul peut être effectué par une machine de Turing, et donc par un ordinateur (pourvu qu'il dispose d'un espace mémoire et d'un temps suffisants).

II / Le remplacement du problème initial (« une machine pourrait-elle penser ? ») par une question jugée moins obscure : « Peut-on imaginer des ordinateurs numériques qui fassent bonne figure dans le jeu de l'imitation ? », étant entendu qu'une réponse positive à la seconde question justifierait l'*usage* du mot « penser » au sujet des machines.

III / La possibilité d'écrire des programmes permettant à une machine de jouer avec succès au jeu de l'imitation.

Les trois points sont liés de la manière suivante : Turing cherche à convaincre le lecteur qu'il n'y a aucun obstacle de principe pour programmer la capacité de jouer au jeu de l'imitation de manière à ce que le programme réussisse à tromper l'interrogateur (III). Or, quels que soient les calculs requis pour réaliser une telle performance, ils pourraient être effectués par un ordinateur (I). Turing estime que la complexité de ces calculs ne serait pas rédhibitoire. Mais dans ce cas, il n'y aurait plus aucune raison de ne pas *dire* au sujet de certaines machines qu'elles pensent (II).

Les objections philosophiques opposables à cette argumentation portent davantage sur les deux derniers points que sur le premier. Tout d'abord, à supposer qu'une machine puisse faire bonne figure au jeu de l'imitation, comment interpréter ce succès ? Dans un

1. *Ibid.*

article aujourd'hui célèbre, Searle élabore une machinerie argumentative, celle de « la chambre chinoise »[1], visant les thèses de ce qu'il appelle « l'intelligence artificielle forte », mais qui peut s'appliquer au jeu de l'imitation. Selon Searle, le succès d'une machine à ce jeu prouverait qu'elle est capable de *simuler* une certaine forme de comportement intelligent (celle qu'on peut manifester par une communication verbale à distance), mais il n'établirait certainement pas que la machine *pense* au sens où elle *comprendrait* ce qu'on lui dit, comme un être humain peut comprendre les questions qu'on lui pose et les réponses qu'il donne. Si l'argument s'applique à certaines thèses philosophiques inspirées par les travaux en intelligence artificielle, il n'est pas évident que ce type d'objection atteigne l'opinion soutenue dans l'article de 1950 puisque Turing ne prétend rien affirmer au sujet de la conscience ou d'une quelconque intériorité de la pensée. Il semble plutôt vouloir soutenir que les progrès prévisibles des machines réduiront tellement les différences de capacité entre un homme et une machine que « l'usage, les mots et l'éducation de l'opinion générale » seront conduits à banaliser une expression comme « machine pensante » : car si une machine pouvait passer le test de Turing avec succès, on ne pourrait plus invoquer aucune *raison* pour rejeter une telle expression comme étant absurde (les *raisons* en question prenant la forme de capacités manifestées par un comportement humain, mais qui seraient inaccessibles aux machines – par exemple écrire une symphonie, émouvoir un homme par l'expression verbale de sentiments, etc.).

Pourtant, même si les objections de Searle n'atteignent pas les idées que Turing soutient en 1950, le jeu de l'imitation n'en reste pas moins d'une interprétation difficile : A quelles conditions pourra-t-on dire que la machine gagne une proportion significative des parties jouées ? Que prouverait un éventuel succès de la

1. John Searle, Minds, Brains and Programs, *The Behavioral and Brain Sciences,* vol. 3, p. 417-458, 1980 ; trad. franç. partielle dans Hofstadter et Dennett (dir.), *Vues de l'esprit,* Paris, InterÉditions, 1987. Les principaux points de l'argumentation sont repris dans un article de Searle, L'esprit est-il un programme d'ordinateur ?, *Pour la science,* n° 149, mars 1990.

machine ? Ne dépend-il pas de l'habileté des candidats humains en jeu ? Les règles du jeu ne sont-elles pas trop restrictives pour ce que Turing voudrait établir (il s'agit d'un interrogatoire, non d'un dialogue, le comportement observable est strictement verbal[1], etc.) ? Au total, il n'est pas évident que la question de substitution soit beaucoup plus claire que la question initiale : « Une machine pourrait-elle penser ? ».

Une seconde catégorie d'objections porte cette fois sur le troisième point (III) : il semble aujourd'hui beaucoup plus difficile de programmer une machine pour le jeu de l'imitation que Turing ne l'avait imaginé. Comme l'écrit Daniel Andler,

> [L']objection concerne la nature particulière du comportement linguistique : (...) la désinvolture de Turing sur ce chapitre semble aujourd'hui incroyable : rien n'indique qu'il entrevoie la difficulté pour la machine (ou son programmeur) de passer d'une « pensée » ou d'une « intention » communicative ou informative à une expression linguistique correcte et pragmatiquement adéquate. (...) C'est peut-être là que l'intuition de Turing, un demi-siècle plus tard, semble le plus manifestement prise en défaut[2].

Même si l'on admet qu'il existe un programme permettant de produire par un calcul des réponses comparables à celles qu'un être humain donnerait aux questions quelconques d'un interrogateur, il est loin d'être évident que ce programme puisse être implémenté sur une machine réelle. Autrement dit, Turing sous-estime certainement de beaucoup les difficultés liées à la complexité des calculs (mesurée en espace mémoire et en temps requis pour effectuer les opérations). La thèse de Turing, énoncée au sujet de machines abstraites en 1936, est appliquée aux dispositifs physiquement réalisés que sont les ordinateurs en 1950. Or cette transposition pose un problème de complexité. Ce type d'objection doit être rappro-

1. Dans le jeu de l'imitation, l'interrogateur n'appréhende l'intelligence et la pensée de ses interlocuteurs qu'à travers un échange de questions et de réponses à distance, par l'intermédiaire d'un téléscripteur. Selon Turing, cela ne constitue pas une objection sérieuse à la validité de son test.
2. Daniel Andler, Turing : pensée du calcul, calcul de la pensée, *Rapport n° 9611*, CREA, février 1996, p. 19-20, à paraître dans Frédéric Nef *et al.*, *Les années 1930 : réaffirmation du formalisme*, Paris, Vrin.

ché d'une critique plus générale visant toute tentative pour programmer un comportement intelligent[1] dans des circonstances qui ne sont pas strictement délimitées : les difficultés que rencontre l'intelligence artificielle au niveau empirique s'expliqueraient par une impossibilité de principe et ne devraient donc rien attendre d'éventuels progrès techniques à venir. Mais avant d'exposer la raison de cette critique, présentons les thèses philosophiques inspirées des travaux en intelligence artificielle, qui sont directement visées. Elles donnent une autre illustration de la manière dont les machines logiques et informatiques peuvent être mises au service d'une réflexion philosophique relative à l'esprit.

Les machines, les programmes et l'explication en psychologie

L'intérêt porté par différents philosophes aux travaux d'intelligence artificielle vient en grande partie de ce qu'ils permettent d'introduire un nouveau type d'explication des phénomènes de l'esprit (ou de certains d'entre eux) en posant l'existence d'un niveau de description du comportement intelligent, intermédiaire entre le niveau physique et le niveau dit « psychologique » ou « intentionnel ». Par analogie avec les machines informatiques, ce niveau peut être conçu comme celui des programmes ou du traitement de l'information (c'est-à-dire des calculs effectués sur des données conformément à certains programmes enregistrés). Pour le comprendre, considérons un exemple simple comme celui du jeu d'échecs. Au niveau intentionnel ou psychologique, on décrit le joueur en disant qu'il élabore une stratégie, imagine celle de l'adversaire, évalue les risques de tel ou tel coup, manifeste par son jeu sa volonté de gagner la partie, etc. Si l'on cherche à concevoir un programme d'intelligence artificielle pour rendre une machine capable de jouer

1. Sur l'objection classique selon laquelle un comportement programmé ne saurait être intelligent puisqu'il est prévisible, on pourra consulter, par exemple, Wagner, *op. cit.*, chap. 1, ou la plupart des ouvrages philosophiques relatifs à l'intelligence artificielle.

aux échecs, un ensemble de stratégies devra être défini et traduit sous la forme de règles générales qui détermineront le comportement de la machine pour une situation de jeu quelconque, sans prévoir explicitement pour autant un coup précis pour chaque disposition possible des pièces sur l'échiquier[1]. Ces règles sont ensuite exprimées dans un langage informatique évolué. On programme ainsi le comportement intelligent d'une machine dans le cadre strictement circonscrit du jeu d'échecs.

Le comportement de la machine au cours d'une partie peut alors, au moins en principe, être décrit d'un triple point de vue. Au niveau intentionnel, on dira de la machine : « elle cherche à mettre l'adversaire en échec », « elle sacrifie sa reine », « elle opère un retrait stratégique », etc. Au niveau du programme, on pourrait suivre chaque étape de l'algorithme et comprendre à l'issue de quelle suite de calculs tel coup est joué dans telle situation de jeu. Enfin, bien que cela soit irréalisable en pratique, on peut concevoir comme théoriquement possible de donner une explication du comportement de la machine à un niveau et dans des termes purement physiques. Or à partir de ce cas paradigmatique, on aperçoit une nouvelle stratégie explicative des comportements et des capacités de l'esprit, par réduction à un programme pour le traitement d'informations. Considérons la description psychologique ou intentionnelle d'une de ces formes d'intelligence qui se manifestent par la capacité de résoudre tel type de problèmes, d'effectuer telle sorte de performance, ou d'agir avec succès dans telles circonstances (jouer aux échecs, décrire une image, traduire un texte, établir un diagnostic médical, interpréter les résultats d'une analyse chimique, assembler les pièces d'un appareil, etc.). On s'efforce de réduire cette description à la combinaison de plusieurs capacités plus élémentaires, et l'on poursuit ainsi l'analyse jusqu'à obtenir des capacités qui puissent être programmées. L'explication peut être considérée comme achevée ou complète lorsqu'une combinaison de programmes permet de conférer à une machine la capacité initialement décrite. De ce point de vue, la psychologie et

1. Cela ne serait pas réalisable à cause du trop grand nombre de situations de jeu possibles.

l'intelligence artificielle effectuent des travaux complémentaires puisque l'une s'efforce de réduire les comportements intelligents et les capacités de l'esprit à des règles et que l'autre élabore des programmes qui expriment formellement les règles caractéristiques d'une forme d'intelligence particulière[1].

Cette stratégie explicative s'inspire du modèle informatique et notamment de la possibilité de décrire le comportement d'une machine aux niveaux intentionnel, computationnel et physique. Elle s'appuie aussi sur la thèse de Church-Turing, reformulée en termes de programmes et de machines réelles : « Tous les calculs peuvent être programmés, et donc effectués par une machine physiquement réalisée. » Enfin, elle pose, au moins à titre d'hypothèse méthodologique, l'idée suivante : « Toutes les capacités de l'esprit que l'on peut caractériser au niveau intentionnel, tous les comportements qui manifestent une forme d'intelligence, peuvent être réduits à un ensemble de règles non intentionnelles, c'est-à-dire à une forme de calcul. » Ce dernier énoncé peut être considéré comme une idée directrice pour la recherche en psychologie, une hypothèse qu'il convient de mettre à l'épreuve. Mais il est parfois reçu dogmatiquement, et doit alors être rapproché des thèses de ce que Searle appelle « l'intelligence artificielle forte », selon lesquelles l'esprit est au corps ce que le programme est à la machine physique. En d'autres termes, toutes les capacités de l'esprit pourraient être expliquées par un ensemble de programmes, susceptible d'être caractérisé indépendamment du corps, celui-ci n'étant donc qu'une « implémentation » de l'esprit-programme parmi d'autres possibles.

C'est ici que se présente une objection de principe, distincte de celle de Searle, et qui vise à la fois les thèses philosophiques inspirées de l'intelligence artificielle, la possibilité d'expliquer les phénomènes de l'esprit en général en termes de programmes et de traitement de l'information, et l'opinion de Turing selon laquelle une machine

1. Cf. Daniel Dennett, *Brainstorms. Philosophical Essays on Mind and Psychology,* Montgomery, Bradford, 1978, nouv. éd., Brighton, Harvester Press, 1981, chap. 5 et chap. 7, ainsi que Robert Cummins, *The nature of psychological explanation,* Cambridge, Mass., Londres, The MIT Press, 1983.

pourrait faire bonne figure au jeu de l'imitation. Elle est formulée notamment par Dreyfus[1] et peut être résumée de la manière suivante.

L'intelligence que l'on manifeste dans une situation déterminée dépend essentiellement de la connaissance précise d'innombrables détails qui caractérisent cette situation. Nous autres humains agissons toujours à l'intérieur d'un monde déjà donné comme le fond de notre représentation, alors qu'une machine ne dispose que des connaissances qui lui sont explicitement communiquées et des conséquences qu'elle est capable d'en inférer. En outre, rien ne permet d'affirmer que l'intelligence manifestée par nos attitudes pratiques puisse être expliquée ou simulée à partir d'un ensemble de connaissances. Or si nous possédons d'une part certaines aptitudes proprement corporelles et d'autre part un nombre virtuellement infini de connaissances implicites grâce à ce que nous appelons notre sens commun, la machine, de son côté, ne peut faire usage que de connaissances formalisées qui doivent être intégrées dans un programme, et mises à jour en temps réel en fonction d'une situation ou d'un problème qui peuvent évoluer. Ses mouvements et déplacements éventuels ne peuvent être réglés que par des connaissances auxquelles elle doit pouvoir accéder d'une manière ou d'une autre. On se heurte alors à une difficulté qui semble tout à fait rédhibitoire car les différents contextes possibles d'une action ou d'un comportement sont d'une diversité illimitée ; en outre, ils ne dépendent pas seulement d'un environnement objectivement descriptible, mais également de notre culture, de notre humeur, de nos désirs, etc. Alors que l'intelligence et le corps humains savent s'adapter à une infinité de tels contextes, on ne voit pas du tout comment communiquer à une machine programmable la représentation qu'un individu pourrait se faire du monde dans son ensemble. Or dans un jeu comme celui de l'imitation, le nombre de situations auxquelles l'interrogateur est susceptible de se référer, et qu'un être humain est capable de comprendre, est arbitrairement élevé. Et au problème de la quantité des connaissances qu'il fau-

1. Dreyfus, *op. cit.*

drait pouvoir communiquer à l'ordinateur s'ajoute celui de leur représentation dans la machine et des moyens qui lui permettraient d'y accéder suffisamment rapidement pour réagir, comme disent les informaticiens, « en temps réel ». Un auteur comme Dreyfus estime que le développement d'une telle argumentation suffit à rendre vain et caduc tout projet d'intelligence artificielle qui viserait à mettre sur un pied d'égalité les hommes et les machines, la critique valant en particulier pour le test de Turing.

La difficulté vient essentiellement d'un problème de complexité qui apparaît dès que l'on cherche à transposer la thèse de Church-Turing en termes de programmes ou de machines réelles. En 1936, les logiciens qui s'intéressaient à la classe des fonctions calculables ne se préoccupaient pas de savoir si la solution algorithmique de tel ou tel problème était compatible avec les contraintes physiques de l'univers ou la durée moyenne de vie d'un être humain. Or les spéculations sur la possibilité de réduire l'esprit à un programme perdraient beaucoup de leur intérêt si l'exécution de ce programme requérait, comme c'est le cas de certains algorithmes connus, un temps de calcul astronomique et définitivement rédhibitoire.

L'esprit humain à l'aune des théorèmes d'incomplétude

Contrairement à la thèse de Turing, l'énoncé des théorèmes d'incomplétude ne concerne pas directement les machines. On a vu cependant que leur généralisation à la classe de tous les systèmes formels possibles suppose une caractérisation de l'effectivité qui est précisément donnée par les machines de Turing. D'un point de vue très général, et selon l'expression de Gödel lui-même, un système formel peut être défini comme une procédure mécanique pour la production de formules qu'on appelle « formules démontrables », et l'essence du système formel est que « le raisonnement est complètement remplacé par des opérations mécaniques sur des formules »[1].

1. Voir ci-dessus, Introduction, p. 3.

Un système formel peut donc être assimilé à une machine à produire des théorèmes, et toute machine informatique capable de produire des formules peut être considérée comme un système formel particulier. Bien entendu, un grand nombre de ces systèmes ou machines ne présentent aucun intérêt pour la formalisation de la logique et des mathématiques parce qu'ils ne jouissent pas des propriétés que les logiciens cherchent précisément à satisfaire lorsqu'ils définissent une construction formelle (possibilité d'exprimer formellement certaines notions, correction, complétude, consistance, normalisation, etc.). Mais si l'on considère une machine quelconque qui répond aux conditions énoncées par les théorèmes d'incomplétude (en particulier, qu'elle soit consistante et qu'elle permette d'exprimer une certaine partie de l'arithmétique élémentaire), alors il existe une formule formellement indémontrable par cette machine. Pourtant, si l'énoncé considéré par Gödel est indécidable dans le système formel lui-même (c'est-à-dire pour la machine), il peut être reconnu par le logicien comme étant vrai, d'un point de vue métalinguistique. En effet, la signification de l'énoncé en question est « je ne suis pas démontrable » et l'on peut prouver qu'il est effectivement indémontrable. En ce sens, l'être humain qu'est le logicien semble être supérieur à la machine dans la mesure où il est capable de donner la preuve d'une formule que la machine ne peut démontrer. Ce type d'argument permet-il pour autant d'établir une comparaison convaincante entre l'homme et la machine ? Qu'est-il légitime de conclure des théorèmes d'incomplétude à cet égard ? Ces interrogations ont suscité de très nombreuses réflexions logico-philosophiques, mais il n'est pas question ici d'examiner les différentes interprétations que l'on a pu donner des résultats publiés par Gödel en 1931[1]. Pour comprendre quel genre d'usage la philosophie peut faire de ces théorèmes dans le but d'alimenter une

1. Sur ce point, l'une des réflexions les plus approfondies est celle de Webb, *op. cit.* On pourra aussi se référer aux nombreuses indications bibliographiques données dans cet ouvrage.

réflexion relative à l'esprit, il suffira d'évoquer sommairement quelques-unes de ces interprétations.

Pour Turing, l'argument ne suffit certainement pas à prouver la supériorité de l'homme sur la machine en général :

> Nous donnons nous-mêmes trop souvent des réponses fausses à des questions pour que nous ayons le droit de nous réjouir d'une telle preuve de la faillibilité des machines. Nous ne pouvons de plus, en de telles occasions, ressentir notre supériorité que par rapport à la machine particulière sur laquelle nous avons remporté un succès insignifiant. Il ne serait pas question de triompher simultanément de *toutes* les machines. En bref il se pourrait qu'il y ait des hommes plus intelligents que n'importe quelle machine donnée, mais il se pourrait aussi qu'il y ait d'autres machines encore plus intelligentes, et ainsi de suite[1].

Turing distingue donc deux raisons de ne pas recevoir l'argument : la première vient de ce qu'il met en évidence une limite de la machine sans prendre en considération les incapacités de l'être humain. La seconde de ce qu'il ne saurait établir globalement la supériorité de l'homme sur *toutes* les machines. Le dernier point dépend directement de questions logiques complexes (et dont l'interprétation philosophique est délicate), qui portent sur la possibilité de construire de manière systématique des suites de systèmes formels de plus en plus puissants par adjonction successive d'axiomes choisis parmi les propositions formellement indécidables du système précédent. Cette perspective, qui a fait l'objet de nombreux commentaires, n'est pourtant pas la seule possible. Dans un article intitulé *L'esprit humain, la machine et Gödel*, Lucas écarte la question du rapport de *supériorité* pour soutenir la thèse antimécaniste selon laquelle la machine ne pourra jamais être *similaire* à l'esprit humain. Pour lui, les théorèmes d'incomplétude prouvent que

> nous ne pourrons jamais espérer construire une machine capable de faire tout ce qu'un esprit peut faire : nous ne pourrons jamais, pas même en principe, avoir un modèle mécanique de l'esprit[2].

Les arguments développés par Lucas ne font cependant pas l'unanimité puisque certains auteurs soutiennent que les théorèmes d'in-

1. Turing, *in* Anderson, *op. cit.*, p. 52 (trad. franç.).
2. Lucas, L'esprit humain, la machine et Gödel, *in* Anderson, *op. cit.*, p. 85.

complétude permettent au contraire d'étayer les thèses du mécanisme en philosophie de l'esprit. Les arguments de Webb sont clairement orientés dans cette direction puisqu'il écrit que

> loin de réfuter le mécanisme, comme beaucoup l'ont prétendu, Gödel a découvert une sorte de bouclier protecteur invisible recouvrant le fondement du mécanisme moderne, à savoir la thèse de Turing[1].

Ces quelques citations donnent une première idée des difficultés que l'on rencontre dès que l'on cherche à donner une interprétation correcte et convaincante des théorèmes d'incomplétude en dehors du domaine de la logique mathématique *stricto sensu,* puisque les uns et les autres se servent de ces résultats pour étayer des positions philosophiques littéralement opposées. Il est souvent difficile de distinguer, parmi les conséquences que l'on croit pouvoir tirer des travaux de Gödel, celles qui sont légitimes de celles qui sont douteuses ou erronées. De là l'interminable conflit entre les commentateurs qui aperçoivent dans les théorèmes d'incomplétude une preuve de la supériorité des hommes sur les machines, ceux qui critiquent ce genre d'argument sans avancer aucune thèse ou ceux qui pensent au contraire qu'il convient d'en donner une interprétation mécaniste. La question est examinée par Turing dans l'article de 1950 et elle suscite encore la réflexion philosophique aujourd'hui.

Il n'est pas question ici de prendre position à l'intérieur de ce débat[2]. Il suffira d'en avoir tracé l'orientation générale en montrant comment les théorèmes d'incomplétude peuvent être utilisés, au même titre que certaines idées empruntées à la théorie de la calculabilité, pour alimenter la réflexion philosophique sur les capacités de l'esprit comparées à celles des machines, et sur la possibilité d'une compréhension mécaniste des phénomènes psychiques. Dans

1. Webb, *op. cit.,* p. 9, voir aussi p. 202 et 208. La position de Webb sur cette question est expliquée au chapitre IV de son ouvrage, ainsi que dans Judson C. Webb, Gödel's Theorems and Church's Thesis. A Prologue to Mechanism, *in* R. S. Cohen et M. W. Wartofsky (éd.), *Language, Logic, and Method,* Boston Studies in the Philosophy of Science, vol. XXXIII, Dordrecht, D. Reidel, 1983, p. 309-353.
2. On pourra trouver un examen plus approfondi de la question dans Wagner, *op. cit.*

la perspective qui vient d'être considérée, les résultats obtenus par Gödel en 1931 (et leurs généralisations), tout comme les machines de Turing et la thèse de Church, sont mis au service d'une problématique fondée sur les machines logiques et informatiques, mais beaucoup plus clairement orientée vers la psychologie, le fonctionnement et les capacités de l'esprit que vers la logique proprement dite.

LE COGNITIVISME, PHILOSOPHIE MÉCANISTE DE L'ESPRIT

Les principes du cognitivisme

Les travaux d'intelligence artificielle sont à l'origine d'une forme d'explication des phénomènes de l'esprit qui est fondée sur les notions de calcul, de machine, de traitement de l'information, et qui relève, pour cette raison, de la psychologie cognitive. Cependant, la psychologie inspirée des machines logico-informatiques a pu être comprise en des sens différents et elle a beaucoup évolué au cours des quarante dernières années. Au prix d'une révision des espoirs parfois naïvement conçus dans l'enthousiasme des premiers pas de l'intelligence artificielle, à la lumière des réussites et des échecs sur les plans empirique et théorique, et à partir d'une ouverture des recherches vers d'autres disciplines comme la neurobiologie, la neuropsychologie, la linguistique, l'anthropologie ou la sociologie, le modèle cognitif s'est progressivement imposé comme l'une des principales voies pour la recherche d'une science des phénomènes de l'esprit aujourd'hui, sans donner pour autant à l'intelligence artificielle une place centrale au sein des sciences cognitives. Les sciences de la cognition suscitent l'espoir de pouvoir concilier des méthodes et des points de vue différents, voire opposés. Pour ne citer qu'un exemple, l'approche cognitive pourrait rendre possible une mise en rapport des approches clinique, neurobiologique et

psychologique de certains phénomènes psychopathologiques[1]. L'explication cognitive, ici, se situe encore à un niveau analogue à celui des programmes et du traitement de l'information, niveau intermédiaire entre les points de vue physique et intentionnel lorsqu'il s'agit d'expliquer le comportement intelligent d'une machine ; pour autant, elle ne suit pas le modèle de l'intelligence artificielle et l'idéal d'une programmation informatique des capacités de l'esprit. Si, dans les pages qui précèdent, nous avons abordé la psychologie cognitive à partir des notions de machine et de programme, cette circonstance ne doit pas nous faire oublier que l'intelligence artificielle n'est pas au centre des sciences cognitives : elle ne constitue que l'une des branches d'un mouvement qui reconnaît également la possibilité d'autres approches.

La *science* cognitive des phénomènes psychiques que l'on cherche à élaborer à partir du modèle logico-informatique des machines s'appuie sur une conception générale de l'esprit que l'on dit volontiers *philosophique* dans la mesure où elle se présente aussi comme un début de solution possible pour plusieurs problèmes qui appartiennent à l'histoire de la philosophie, notamment le problème des rapports entre l'esprit et le corps, celui de la nature des états mentaux et celui du fonctionnement de l'esprit. Dans cette perspective, comment les relations entre science et philosophie sont-elles conçues ? Pour le comprendre, précisons au préalable ce qu'il convient d'entendre par une conception « cognitiviste » de l'esprit[2]. Il s'agit d'une forme de fonctionnalisme[3], ce qui signifie, première-

1. Cette position est défendue, par exemple, dans l'ouvrage de Nicolas Georgieff, *La schizophrénie,* Paris, Flammarion, 1995.
2. Cf. Daniel Andler, *op. cit.,* et Daniel Andler, Les sciences de la cognition, *in* Jean Hamburger (dir.), *La philosophie des sciences aujourd'hui,* Paris, Gauthier-Villars, 1986. Les sciences cognitives traitent toutes, chacune d'un point de vue différent, de la cognition, sans que cela implique nécessairement l'adhésion à cette conception de l'esprit qu'est le cognitivisme. On distingue d'autres positions comme l'éliminativisme, le phénoménisme, le connexionnisme, etc., qui sont plus ou moins éloignées du modèle computationnel des machines.
3. Il existe toute une variété de conceptions fonctionnalistes de l'esprit, dont on pourra se faire une idée en consultant, par exemple, William Bechtel, *Philosophy of Mind. An overview for Cognitive Science,* Hillsdale, New Jersey, Lawrence Erlbaum Associates Publishers, 1988.

ment, que l'esprit est conçu comme un appareil dont on cherche à comprendre le *fonctionnement*, notamment à partir d'une analogie avec les machines logico-informatiques ; deuxièmement, que l'explication des phénomènes psychiques repose sur une *analyse fonctionnelle* des capacités et des comportements, de même que les performances d'une machine informatique s'expliquent par les fonctions d'un ensemble de programmes, et non les propriétés physiques de l'ordinateur sur lequel ils sont implémentés ; troisièmement, que le comportement et les phénomènes psychiques sont une *fonction* à la fois de stimuli et, de manière réciproque, d'autres phénomènes psychiques corrélatifs ; globalement, l'esprit est comparable à une machine de Turing dont le fonctionnement dépend à la fois des données initiales inscrites sur le ruban et des états internes. Pour le cognitivisme, la cognition est comprise comme un ensemble d'opérations de calcul effectuées sur des représentations, étant bien entendu que ces représentations ne sont pas nécessairement conscientes, et qu'elles sont définies, au niveau du traitement de l'information, par leur fonction au sein de l'appareil psychique dans son ensemble. L'idée principale du fonctionnalisme est que l'analyse du rôle fonctionnel des représentations doit suffire pour l'explication des phénomènes de l'esprit. Dans cette perspective, l'importance de la thèse de Church-Turing vient de ce qu'elle caractérise l'ensemble des possibilités de calcul. Alors que l'intelligence artificielle n'est que l'une des manières possibles de concevoir la recherche en science cognitive, la thèse de Church-Turing donne une base théorique générale et commune aux différentes approches de la cognition.

L'une des principales conséquences du point de vue qui vient d'être défini est tirée par Putnam, dans un des premiers textes qui présentent une conception fonctionnaliste de l'esprit[1]. A partir d'une analogie entre les états mentaux et les états logiques d'une machine

1. Hilary Putnam, Minds and machines, *in* Sidney Hook (éd.), *Dimensions of Mind : A Symposium*, New York, New York University Press, 1960 ; l'article est reproduit dans Anderson, *op. cit.*

de Turing (l'opposition états mentaux - états physiques du corps est mise en rapport avec l'opposition états logiques - états structuraux d'une machine de Turing physiquement réalisée), Putnam entend montrer que le problème des relations entre le corps et l'esprit est aussi dénué d'intérêt que celui des relations entre les états logiques d'une machine de Turing et les états physiques de sa réalisation matérielle. Quelle que soit la valeur de la thèse défendue par Putnam[1], elle permet d'apercevoir l'une des idées-forces du cognitivisme : bien qu'ils ne puissent pas être réduits à des états du corps ou du cerveau, les états mentaux n'appartiennent pas à une substance immatérielle ; ils doivent être compris comme des fonctions intégrées dans un appareil psychique qui est complexe et entièrement réalisé (ou « implémenté », comme un programme sur une machine) dans le corps. Les états mentaux ne sont par eux-mêmes rien d'autre que des états du corps, bien que les catégories d'états mentaux, celles dont fait usage la psychologie, ne soient pas réductibles à des catégories de la biologie ou de la physique. Ici encore, c'est l'analogie avec l'ordinateur qui est la plus éclairante. Le fonctionnement de l'esprit est comparable aux calculs effectués par une machine selon les instructions d'un programme, et les états mentaux analogues aux états logiques de la machine : ils peuvent correspondre à des états physiques différents, par exemple si le programme est implémenté sur des machines physiques différentes.

On voit maintenant comment la recherche d'une science cognitive des phénomènes psychiques peut faire fond sur cette conception générale de l'esprit que l'on appelle communément « cognitivisme ». On voit aussi quel serait le rôle du philosophe qui voudrait se situer dans la mouvance des sciences cognitives : alors que les spécialistes en neurobiologie, neuropsychologie, intelligence artificielle, psychologie cognitive, linguistique, anthropologie et sociologie travaillent chacun de leur point de vue particulier, fût-ce dans

1. La position de Putnam a évolué puisqu'en 1988, dans *Representation and reality*, Cambridge, Mass., The MIT Press, 1988, trad. franç., *Représentation et réalité*, Paris, Gallimard, 1990, il soutenait des idées opposées au fonctionnalisme dont il avait pourtant été l'un des promoteurs dans l'article de 1960, « Minds and machines ».

l'esprit du paradigme cognitif, il aurait pour tâche de montrer comment les différents niveaux d'explication des phénomènes peuvent être rendus compatibles, d'analyser les concepts fondamentaux et les méthodes de chaque discipline dans un esprit de synthèse, et d'élaborer une conception cohérente de l'esprit capable à la fois d'élargir le cadre théorique des sciences particulières et de clarifier leurs relations mutuelles. Selon Dan Sperber, c'est de la manière suivante que l'on peut concevoir la position du philosophe cognitiviste :

> Les sciences cognitives ne sont pas seulement mécanistes, elles sont aussi matérialistes ou, comme on dit aujourd'hui, « naturalistes ». Être naturaliste, c'est penser que le rapport entre une cause et son effet n'existe qu'en vertu de leurs propriétés physiques : il n'y a pas d'autres pouvoirs causaux dans le monde que ceux de la matière. Dans ces conditions, toute science capable de fournir des explications causales doit décrire le monde physique à un certain niveau de complexité ou d'abstraction. On ne peut affirmer qu'il en va bien ainsi que si le rapport entre le niveau particulier où travaille une science donnée et le niveau physique de base est clairement conçu. Comment les sciences cognitives peuvent-elles satisfaire à cette exigence ? C'est là que les philosophes interviennent[1].

Dans la perspective cognitive, le philosophe est donc loin de jouer un rôle négligeable puisqu'il est investi d'une fonction de clarification des concepts et d'analyse des relations entre différents niveaux d'investigation théorique des phénomènes de l'esprit. Il cherche à élaborer une conception générale du psychisme permettant de colliger en un tout cohérent l'ensemble des connaissances qui relèvent du champ des sciences cognitives. Sa position à l'égard de la science n'est donc ni externe ni normative : il participe lui-même d'un mouvement scientifique en devenir. Par là, on aperçoit aussi une autre caractéristique de la philosophie ainsi comprise : mise au service d'un ensemble de disciplines dont elle a pour charge d'assurer les relations mutuelles ainsi que le fondement, le paradigme et le procès communs, elle devient servante des sciences de la cognition : *ancilla cognitionis scientiarum.*

1. Dan Sperber, Connaître l'acte de connaître, *Le Monde,* 22 octobre 1993, p. 32.

Une nouvelle philosophie mécaniste

Bien que cette manière de concevoir les relations entre science et philosophie soit fort peu conforme à la pensée de la plupart des philosophes du XVII[e] siècle, il arrive fréquemment qu'au sujet du cognitivisme et d'autres conceptions de l'esprit apparentées l'on parle de philosophie *mécaniste*, suggérant ainsi une analogie avec le mécanisme de l'époque classique. Dans ce cas, ce n'est pas nécessairement à la réduction par Hobbes de la *cogitatio* à une *computatio* que l'on pense, ni au « *calculemur* » leibnizien, mais également à la philosophie cartésienne de la nature, dans la mesure exacte où elle a puissamment contribué à l'avènement d'une physique mathématisée.

Chez Descartes, le mécanisme est un modèle explicatif qui se comprend par référence aux machines et qui exprime un idéal d'intelligibilité des phénomènes de la nature. Descartes pense d'abord aux machines simples de la tradition (le levier, le coin, la vis, la poulie, le treuil, le plan incliné)[1], mais également aux montres, horloges, fontaines artificielles, moulins, automates divers ou machines mouvantes qui exemplifient de manière frappante la possibilité d'obtenir un grand nombre d'effets divers, parfois fort élaborés, par des moyens purement mécaniques :

> (...) s'ils approchent d'une Diane qui se baigne, ils la feront cacher dans les roseaux ; et s'ils passent plus outre pour la poursuivre, ils feront venir vers eux un Neptune qui les menacera de son trident ; ou s'ils vont de quelque autre côté, ils en feront sortir un monstre marin qui leur vomira de l'eau contre la face ; ou choses semblables selon le caprice des ingénieurs qui les ont faites[2].

Les machines illustrent ainsi un idéal d'intelligibilité qui écarte tout recours aux qualités occultes et formes substantielles pour l'explication des phénomènes de la nature, y compris, selon la thèse des ani-

1. Lire son *Explication des engins par l'aide desquels on peut avec une petite force lever un fardeau fort pesant*, éd. Alquié, t. 1, Garnier, p. 802-814, texte publié en 1668 par le P. Poisson sous le titre *Traité de la mécanique*.
2. *L'homme*, éd. Alquié, t. 1, p. 390-391. Cf. aussi, entre autres références, *L'homme, passim*, *Discours de la méthode*, 5[e] partie, ou *Principes de la philosophie*, 4[e] partie, § 203.

maux-machines, les phénomènes du vivant. Pourtant, l'essentiel du mécanisme cartésien ne réside pas dans cette référence aux automates et autres artefacts, qui sert surtout à *illustrer* la possibilité d'un autre modèle explicatif[1] ; il doit être cherché dans la critique d'une physique qui, faisant appel aux qualités réelles, aux formes substantielles et aux forces occultes, conduit, selon une caricature célèbre, à expliquer la vertu dormitive de l'opium par une *virtus dormitiva*. Descartes reproche à la physique qualitative d'origine aristotélicienne de n'apporter aucune réelle intelligibilité, et lui substitue sa physique mécaniste, mathématique au sens précis où l'explication des phénomènes naturels repose cette fois sur les seuls principes géométriques que sont la figure, la grandeur et le mouvement. C'est cette idée qui constitue le principal trait caractéristique de la philosophie cartésienne et mécaniste de la nature : seules ces qualités d'origine géométrique sont admises à titre de principes physiques, et elles doivent permettre d'expliquer aussi bien toutes les choses de la nature que les machines et artefacts des ingénieurs[2]. Non que Descartes estime possible une compréhension mécaniste effective et vraie de tous les phénomènes de la nature dans leur détail ; mais « la part d'imagination que comporte sa Physique »[3] n'est pas incompatible avec la vérité de ses principes[4].

Par sa référence aux machines comme illustration d'un nouveau modèle d'intelligibilité, par sa critique de la physique qualitative d'origine aristotélicienne et du recours aux qualités occultes, par la géométrisation de l'espace et le principe d'une explication possible des phénomènes naturels en termes de figures, grandeurs et mouvements, le mécanisme cartésien a certainement joué un rôle décisif

1. Ce point est précisé par Jean-Pierre Séris dans une communication intitulée Descartes et la mécanique, *Bulletin de la Société française de Philosophie,* Paris, Armand Colin, t. LXXXI, avril-juin 1987.
2. Contrairement à ce que pense Aristote, les choses artificielles comme les machines ne sont pas seulement des artifices ; selon Descartes et la conception moderne, elles font aussi partie de la nature. « Descartes n'argumente pas à partir de la thèse : "la nature est machine" mais à partir de la thèse "la machine est nature" (à ceci près que les parties ou les causes des effets sont grossières ou sensibles (...) » (Séris, *ibid.*, p. 51).
3. Séris, *ibid.*, p. 48.
4. Cf. Descartes, *Principes de la philosophie,* 3e partie, § 43-47.

pour la création d'une science mathématisée du mouvement, la mécanique, et pour ce qu'il est convenu d'appeler « la révolution scientifique moderne ». On comprend mieux, dès lors, l'expression « mécanisme contemporain », usitée au sujet de conceptions philosophiques de l'esprit comme le cognitivisme. On pense en effet que par la référence aux machines logiques et informatiques, par la critique des explications psychologiques qui traitent les états mentaux comme des qualités ou des facultés occultes, par la définition d'un niveau computationnel et le principe d'une explication possible des phénomènes de l'esprit en termes de calculs et de traitement de l'information, le mécanisme contemporain en philosophie de l'esprit pourrait jouer un rôle décisif dans le développement d'une science de la cognition. On se gardera toutefois d'une analogie trop facile qui semble suggérer que le cognitivisme et le modèle computationnel rendront possible une connaissance scientifique des phénomènes de l'esprit et du comportement. On ne peut ignorer en effet que si la philosophie cartésienne de la nature a largement contribué à la révolution scientifique moderne, la mécanique, en définitive, n'a pu se constituer et se développer comme science mathématisée du mouvement qu'en renonçant aux principes cartésiens de l'explication scientifique, et donc en s'opposant à la philosophie mécaniste de Descartes. Or nul se saurait dire aujourd'hui avec certitude à quels renoncements une philosophie mécaniste comme le cognitivisme devrait consentir afin de rendre possible une science de la cognition, ni même si le développement des sciences cognitives rendra possible une conceptualisation scientifiquement satisfaisante, durable et féconde des phénomènes de l'esprit.

Les machines logiques et informatiques ont rendu possible la définition d'une problématique philosophique relative à l'esprit, à ses capacités et à son fonctionnement. Elles permettent d'élaborer à la fois un modèle théorique pour l'explication cognitive des phénomènes psychiques et une conception générale de l'esprit, le cognitivisme, qui se présente comme une nouvelle forme de philosophie mécaniste. Cela ne signifie pas que les relations entre la logique et les sciences cognitives concernent uniquement les machines ; elles

concernent également les raisonnements que la logique mathématique usuelle ne permet pas de formaliser. Des logiques spécifiques sont élaborées pour les raisonnements en situation d'incertitude, la formalisation des croyances, l'expression de connaissances variables au cours du temps, etc. Mais en ce qui concerne les machines elles-mêmes, les éléments logiques dont fait usage la philosophie de l'esprit se limitent essentiellement aux machines de Turing, à la thèse de Church et aux théorèmes d'incomplétude (dans la mesure où les systèmes formels sont compris comme des procédures mécaniques pour la production de théorèmes). Par là, l'intérêt philosophique des machines logico-informatiques se trouve défini d'un point de vue qui est à la fois tout à fait singulier et très réducteur dans la mesure où il ne rend pas justice à la place que ces machines occupent *en logique*. Car les rapports les plus étroits et les plus profonds entre logique et machine apparaissent seulement lorsque l'on réussit à établir une correspondance entre les preuves et les programmes, c'est-à-dire lorsque l'on comprend dans quelle mesure et en quel sens une démonstration formelle peut être considérée elle-même comme une machine. Or cela suppose que l'on dépasse le cadre bien particulier de la théorie de la calculabilité telle qu'elle fut élaborée dans les années trente, et que l'on prenne en considération à la fois les questions de complexité logique et algorithmique et certains résultats obtenus en théorie de la démonstration comme l'isomorphisme de Curry-Howard et le *Hauptsatz* de Gentzen. Toutes choses que la problématique des machines *en psychologie* ou *en philosophie de l'esprit* ignore entièrement lorsqu'elle s'appuie sur certains éléments bien précis de la théorie de la calculabilité.

A cet égard, la thèse de Church-Turing offre un exemple tout à fait caractéristique de l'usage qui est souvent fait des machines logiques en philosophie. On donne en effet une expression contestable de cette thèse lorsqu'elle est appliquée au cas des machines réelles : toute fonction calculable est calculable par une machine de Turing ; or les ordinateurs sont des machines universelles ; donc toute fonction calculable peut être calculée par les opérations physiques d'une machine informatique. C'est cette version de la thèse

qui est postulée par Turing lui-même en 1950 et par ceux qui entendent généraliser l'explication de type computationnel en psychologie. Or elle présente un double défaut. Premièrement, elle fait entièrement abstraction des problèmes de complexité relatifs au temps et à l'espace mémoire indispensables pour effectuer un calcul. Comme si l'intelligence artificielle pouvait faire abstraction des limites physiques des machines et, corrélativement, comme si la connaissance de l'esprit pouvait ignorer le corps, ses limites et ses incapacités[1]. En 1950, Turing fonde son argumentation sur l'universalité des machines et sous-estime de beaucoup la complexité d'une machine qui serait capable de faire bonne figure au jeu de l'imitation. Rien ne mettra mieux en lumière, quelques années plus tard, la nature et l'importance de cette difficulté que les obstacles rencontrés par les premiers projets d'intelligence artificielle. Deuxièmement, elle occulte, en l'ignorant, l'étude logique de ce que l'on appelle parfois la « faisabilité » d'un calcul par une machine, c'est-à-dire l'évaluation de sa complexité relativement aux contraintes physiques, ou même à des contraintes comme la durée de vie d'un être humain. Pour ne considérer qu'un exemple, si le plus simple des programmes possibles qui permettrait à une machine de jouer avec succès au jeu de l'imitation était si complexe qu'il devait se heurter aux limites physiques des machines, l'argumentation de Turing perdrait beaucoup de sa valeur[2]. Et une objection similaire pourrait être avancée contre la thèse d'une généralisation de l'explication computationnelle en psychologie. Ce qui importe ici, cependant, est moins la possibilité d'élaborer une argumentation antimécaniste fondée sur l'étude logique de la complexité que de comprendre les rapports qui existent entre ces questions relatives aux machines et les problèmes considérés par les logiciens en théorie de la démonstration. C'est donc le rapport entre les

1. Remarquons bien que dans le jeu de l'imitation, toute comparaison entre la machine physique et le corps humain est mise hors-jeu : la communication entre l'interrogateur et les joueurs passe par un téléscripteur.
2. Ce sont des contraintes de cet ordre qui rendent impossible le calcul d'une stratégie gagnante au jeu d'échec par examen systématique de toutes les parties possibles.

machines, les programmes et les preuves mathématiques qui sera examiné au chapitre suivant.

Les théorèmes d'incomplétude offrent un autre exemple caractéristique de l'usage qui est fait des machines en philosophie de l'esprit. Car pour appliquer les théorèmes de Gödel au problème de l'évaluation des capacités de l'esprit par comparaison avec les dispositifs mécaniques, on commence par interpréter les systèmes formels comme des machines capables de produire des théorèmes. Or cette perspective est profondément réductrice dans la mesure où elle conduit à passer sous silence les questions qui font l'intérêt des systèmes formels pour le logicien théoricien de la démonstration : Quelles sont les caractéristiques propres du système en question ? non seulement : Quelles sont les propositions démontrables ? mais aussi : Quelles sont les preuves qu'il permet d'écrire ? Sont-elles constructives ? Le système jouit-il des propriétés de normalisation, ou, pour reprendre l'une des idées fondamentales du programme de Hilbert, est-il « conservatif » ? Comme le montrera le chapitre suivant, c'est précisément lorsque l'on s'interroge sur ce type de questions qu'apparaissent les rapports les plus profonds entre machine et logique. En explicitant ce point, on s'efforcera de mettre en lumière ce qui fait l'intérêt philosophique de la machine *en logique*.

VIII

LA CORRESPONDANCE DE CURRY-HOWARD

INTRODUCTION : LE CONTENU ALGORITHMIQUE DES PREUVES

Calcul logique et calcul fonctionnel

Le mot « calcul » s'entend en deux sens bien différents. Il désigne tout d'abord une suite d'opérations sur des signes, déterminée par les règles d'une procédure effective qui prend la forme d'une machine de Turing, d'un lambda-terme, d'une fonction récursive ou encore, par exemple, d'un programme écrit dans un langage informatique. Ainsi compris, il est une généralisation de la notion commune de calcul numérique et son étude relève de la théorie de la calculabilité. Le sens du mot est tout autre lorsque l'on parle du « calcul des propositions » ou du « calcul des prédicats ». Il signifie alors que dans le cours d'une démonstration, on considère les relations entre les énoncés ou les expressions quelconques d'un point de vue strictement formel, en prenant en considération seulement les aspects morphologique et syntaxique des suites de signes. En ce dernier sens cependant, le calcul désigne non seulement la formalisation des *démonstrations*, mais également celle des *calculs* au premier sens du terme, c'est-à-dire des opérations effectuées pour l'évaluation d'une fonction, et l'on pense alors surtout

au formalisme du lambda-calcul[1]. On distinguera donc les calculs logiques et les calculs fonctionnels, essentiels en informatique. Tous sont des systèmes de signes syntaxiquement définis ; les premiers formalisent la démonstration d'une formule et les seconds l'évaluation d'une fonction sur le modèle du lambda-calcul.

Extraire un programme d'une preuve

C'est à partir de cette distinction entre calculs logiques et calculs fonctionnels qu'il est possible de mettre en correspondance les preuves et les programmes, les formules et les types de données (au sens informatique du terme)[2], la normalisation des preuves et l'exécution d'un programme par réduction d'un lambda-terme. Cette correspondance fut découverte par Curry dans le cadre de la logique combinatoire en 1958, explicitée par Howard en 1969, et utilisée par de Bruijn dès 1968[3]. Elle met en évidence les liens les plus étroits et les plus profonds qui existent entre la logique et l'informatique puisqu'à travers elle, les preuves formelles, au moins si elles sont constructives, peuvent être identifiées à des programmes, c'est-à-dire à des machines abstraites. Considérons l'exemple d'une formule de la forme suivante : $\forall x \exists y \mathrm{R}[x,y]$. Une preuve constructive de cette formule donne le moyen de construire un y_0 vérifiant $\mathrm{R}[x_0,y_0]$ pour tout x_0 donné. Supposons que x_0 soit la représentation codée d'une liste d'entiers, et que la formule $\mathrm{R}[x,y]$ exprime le fait que y soit la liste

1. Ce sens de l'expression « calcul fonctionnel » ne doit donc pas être confondu avec le *Funktionenkalkül* dont il était question dans la première édition de Hilbert et Ackermann, *Grundzüge der theoretischen Logik*, en 1928. Voir ci-dessus, Introduction, n. 1, p. 4.
2. On précisera ci-dessous.
3. H. B. Curry et R. Feys, *Combinatory Logic I*, Amsterdam, North-Holland, 1958 ; W. A. Howard, *The formulæ-as-types notion of construction, in* J. R. Hindley et J.-P. Seldin (ed.), *To H. B. Curry : Essays on Combinatory Logic, Lambda Calculus and Formalism*, New York, Academic Press, 1980 (cet article, écrit en 1969, ne fut pas publié avant 1980) ; N. G. de Bruijn, The mathematical language AUTOMATH, its usage and some of its extensions, *Symposium on Automatic Demonstration, IRIA, Versailles, 1968*, Berlin, Springer, « Lecture Notes in Mathematics », vol. 125, 1970 ; N. G. de Bruijn, A survey of the project Automath, *in* Hindley et Seldin, *op. cit.*

des entiers de x, triée selon une certaine relation d'ordre. Alors une preuve constructive de la formule $\forall x \exists y R[x,y]$ fournit un algorithme de tri, et peut être considérée à ce titre comme une machine qui prend des listes quelconques en entrée et produit les mêmes listes triées en sortie. En dégageant le contenu fonctionnel d'une preuve, la correspondance (on dit aussi l' « isomorphisme ») de Curry-Howard permet d'en extraire un algorithme. La machine apparaît alors liée aux questions de logique en un sens nouveau : il ne s'agit plus seulement d'introduire une notion abstraite de machine équivalente aux ordinateurs et aux langages informatiques pour caractériser la notion d'effectivité et généraliser ainsi l'idée de système formel ; il est maintenant question de mettre en lumière le contenu algorithmique des démonstrations et d'identifier les preuves et les programmes. Par la correspondance de Curry-Howard, la théorie de la démonstration se trouve donc directement liée à l'étude des machines abstraites, et indirectement à celle des machines informatiques réelles.

Les preuves constructives

Les démonstrations mathématiques ne présentent pas toutes un contenu algorithmique. Considérons en effet l'exemple suivant :

> Théorème : il existe des solutions de l'équation $x^y = z$ telles que x et y soient des nombres irrationnels et z soit rationnel.
>
> Démonstration : on sait que $\sqrt{2}$ est irrationnel. $\sqrt{2}^{\sqrt{2}}$ est soit rationnel soit irrationnel. Si $\sqrt{2}^{\sqrt{2}}$ est rationnel, alors posons $x = y = \sqrt{2}$; dans ce cas, x et y sont irrationnels, et z est rationnel. Si en revanche $\sqrt{2}^{\sqrt{2}}$ est irrationnel, alors posons $x = \sqrt{2}^{\sqrt{2}}$ et $y = \sqrt{2}$; alors $z = (\sqrt{2}^{\sqrt{2}})^{\sqrt{2}} = \sqrt{2}^2 = 2$, et donc, ici encore, x et y sont irrationnels, et z est rationnel. CQFD.

Cette démonstration établit qu'il existe trois nombres x, y et z vérifiant l'énoncé du théorème, mais n'offre aucun moyen de les construire ou d'en déterminer la valeur : elle distingue deux cas possibles ($\sqrt{2}^{\sqrt{2}}$ est rationnel ou $\sqrt{2}^{\sqrt{2}}$ n'est pas rationnel) sans préciser lequel est vrai. Elle n'est pas constructive dans la mesure où elle ne donne aucun algorithme pour le calcul des nombres x, y et z. Aucun programme ne correspond donc à cette démonstration qui

utilise le tiers exclu (A ou non-A) comme un principe logique valide. Dans ce qui suit, pour saisir le contenu algorithmique des preuves, on s'appuiera sur une interprétation des opérations logiques qui respecte l'exigence de constructivité. La sémantique de Heyting, conçue pour la logique intuitionniste, n'admet pas le principe logique du tiers exclu ; elle est ici particulièrement bien adaptée[1]. Elle définit la validité d'un énoncé non en termes de ses conditions de vérité, mais selon l'exigence de la construction d'une preuve, en termes de conditions d'assertabilité. Selon Heyting, l'interprétation des opérations logiques est la suivante[2] :

— une preuve de $(P \wedge Q)$ (c'est-à-dire « P et Q ») est un couple de preuves, noté $\langle p, q \rangle$, où p est une preuve de P et q est une preuve de Q ;
— une preuve de $(P \Rightarrow Q)$ (c'est-à-dire « P implique Q ») est une méthode qui transforme toute preuve de P en une preuve de Q, c'est-à-dire qui montre comment obtenir une preuve de Q à partir d'une preuve quelconque de P ;
— une preuve de $(A \vee B)$ (c'est-à-dire « A ou B ») est une preuve de A ou une preuve de B, qui précise si l'on a prouvé A ou si l'on a prouvé B[3] ;
— une preuve de $\neg A$ est une preuve de $(A \rightarrow \bot)$, où $\bot$ est la proposition absurde, qui n'a pas de preuve (par exemple $0 = 1$) ;
— une preuve de $\exists x P(x)$ est la construction d'un objet a et la preuve qu'il possède la propriété P ;
— une preuve de $\forall x P(x)$ est un procédé qui permet de construire une preuve de $P(a)$ pour tout a.

1. Le lecteur ne doit cependant pas en conclure que la correspondance de Curry-Howard ne se conçoit que dans le cadre de la logique intuitionniste. Il existe en effet des extensions de cette correspondance à la logique classique (cf. par exemple Michel Parigot, Classical proofs as programs, *Proceedings K.G.C. '93,* Berlin, Springer, « Lectures Notes in Computer Science », 713, 1993, p. 263-276).
2. A. Heyting, *Intuitionism. An Introduction,* Amsterdam, North-Holland, 1956.
3. La sémantique de Heyting pour la disjonction montre bien pourquoi le tiers exclu ne peut pas être considéré comme un principe logique valide en logique intuitionniste. Dans la démonstration du théorème donné en exemple ci-dessus, on utilise le principe (A ou non-A) sans avoir démontré ni A ni non-A.

LA CORRESPONDANCE DE CURRY-HOWARD DANS UN CAS ÉLÉMENTAIRE

Introduisons maintenant la correspondance de Curry-Howard dans sa forme la plus simple, c'est-à-dire en ne considérant, dans un premier temps, que le fragment implicatif du calcul propositionnel (le seul connecteur est l'implication). Deux formalismes doivent être définis :

(i) un calcul fonctionnel (le lambda-calcul typé, qui permet d'écrire formellement des programmes) ;
(ii) un calcul logique (la déduction naturelle, qui permet d'écrire formellement des preuves).

Les deux calculs se révèlent être strictement identiques. Une seule et même syntaxe reçoit deux interprétations, l'une informatique et l'autre logique. On indiquera ensuite comment la correspondance de Curry-Howard entre programmes et preuves, types de données et formules, calcul par β-réduction et normalisation des preuves, peut être généralisée (c'est-à-dire étendue au-delà du calcul propositionnel réduit à l'implication)[1].

1. Pour un examen plus approfondi de la correspondance de Curry-Howard, on pourra consulter les références suivantes : Thierry Coquand, Sur l'analogie entre les propositions et les types, *in* Cousineau, Curien, Robinet (ed.), *Combinators and Functional Languages*, Berlin, Springer, « Lecture Notes in Computer Science », n° 242, 1986 ; Girard, *op. cit.*, p. 450 et sq. ; Girard, Lafont, Taylor, *op. cit.* ; Jean-Louis Krivine, *Lambda-calcul, Types et Modèles*, Paris, Masson, 1990 ; Lalement, *op. cit.*, chap. III ; Piergiorgio Odifreddi (ed.), *Logic and Computer Science*, Londres, Academic Press, « The Apic Series », n° 31, 1990 ; Simon Thompson, *Type Theory and Functional Programming*, Wokingham, Addison-Wesley, 1991. Une partie de la présentation qui suit s'inspire du début de l'article de Thierry Coquand.

Le calcul fonctionnel

Du point de vue informatique, le lambda-calcul est un langage dans lequel on construit des programmes, c'est-à-dire des termes qui peuvent être considérés comme des machines à réduction[1]. Or ces machines posent un problème que les informaticiens rencontrent au sujet de tous les langages de programmation usuels : comment s'assurer que le programme qu'on a écrit calcule bien la fonction que l'on souhaite évaluer, et comment prouver que son exécution ne produira pas une suite infinie d'opérations sans jamais parvenir au résultat recherché (autrement dit : qu'il ne bouclera pas indéfiniment) ? Les résultats logiques obtenus par Turing et Church en 1936 nous apprennent que ce dernier problème n'admet pas de solution générale : pour les machines de Turing, le problème de l'arrêt est insoluble[2], comme la question de la normalisation pour les termes du lambda-calcul[3]. Il n'existe donc pas de procédure effective permettant de prouver, pour un lambda-terme quelconque, que tous les chemins de réduction possibles sont finis (propriété de normalisation forte), ni même de s'assurer que l'un des chemins de réduction conduit à une forme normale, c'est-à-dire au résultat du calcul (propriété de normalisation faible). Pour faire face à cette difficulté, le lambda-calcul typé prend la forme d'un calcul qui interdit, par des règles syntaxiques, la construction de termes qui ne sont pas normalisables, ou fortement normalisables.

Empruntée à la célèbre théorie des types de Russell, l'idée de *typage* est appliquée aux termes du lambda-calcul par Church dès 1940 dans un contexte qui est encore purement logique[4]. En informatique aujourd'hui, les types d'un langage sont les catégories auxquelles appartiennent les expressions utilisées dans un pro-

1. Cf. ci-dessus, chap. IV.
2. Cf. ci-dessus, chap. II, p. 42-43.
3. Cf. ci-dessus, chap. IV, p. 71 et 73.
4. Alonzo Church, A formulation of the simple theory of types, *Journal of Symbolic Logic*, vol. 5, 1940.

gramme. Par exemple, 0, 1, 2, etc. (les nombres entiers naturels) sont des données de type « Entier », alors que Vrai et Faux sont de type « Booléen ». A ces types simples s'ajoutent des types structurés comme les tableaux (d'entiers, de booléens, etc.), les listes, les arbres binaires ou les fonctions. On définit maintenant l'ensemble des types du lambda-calcul. Étant donné un ensemble $\mathscr{B}$ de types de base (des constantes ou des variables de type, qui représentent, par exemple, les types Entier, Booléen...), l'ensemble $\mathscr{T}$ des types du lambda-calcul est défini inductivement par les deux clauses suivantes :

1 / Les types de $\mathscr{B}$ sont dans $\mathscr{T}$;
2 / Si A et B sont deux types de $\mathscr{T}$, alors $(A \to B)$ est un type de $\mathscr{T}$.

Intuitivement, $(A \to B)$ est le type d'une fonction qui à un objet de type A associe un objet de type B. Ainsi, les objets de type $(N \to N)$ sont des fonctions numériques unaires si N est une notation pour le type « Entier ».

Par la construction du calcul fonctionnel, un type est affecté à chacun des lambda-termes. Soient par exemple un terme t et un type A. L'expression « $t : A$ » signifie que le terme t est de type A. Les termes sont définis relativement à un environnement, c'est-à-dire une liste finie de variables typées qui est donc de la forme $x_1 : A_1, \ldots, x_n : A_n$. Soient Γ un environnement, t un terme et A un type. Une expression de la forme $\Gamma \vdash t : A$ se lit : « le lambda-terme t a pour type A dans l'environnement Γ ». Le calcul fonctionnel se définit alors par les règles suivantes :

1 / Si la variable x de type A figure dans l'environnement Γ, alors $\Gamma \vdash x : A$;
2 / Si $\Gamma, x : A \vdash t : B$, alors $\Gamma \vdash \lambda x.t : (A \to B)$;
3 / Si $\Gamma \vdash t : (A \to B)$ et si $\Gamma \vdash u : A$, alors $\Gamma \vdash (t)u : B$.

La comparaison de ce formalisme avec celui du lambda-calcul pur (voir ci-dessus, chap. IV, p. 67-68), montre que dans le lambda-calcul typé, la construction des termes par abstraction (Règle 2) et par application (Règle 3) est soumise à certaines res-

trictions imposées par le typage. En particulier, les règles syntaxiques rendent impossible la construction de termes comme $(x)x$[1] (qui n'a aucune signification intuitive satisfaisante) ou $(\lambda x.(x)x)\,(\lambda x.(x)x)$ (qui n'est pas normalisable)[2]. Ces termes du lambda-calcul pur ne sont pas typables dans le calcul qui vient d'être défini.

Le calcul logique

Du point de vue logique, on considère pour l'instant le calcul propositionnel $\mathcal{I}_{mp}$, qui est restreint au connecteur de l'implication. Étant donné un ensemble $\mathcal{A}$ de lettres de propositions (ou atomes), l'ensemble $\mathcal{F}$ des formules est défini par les règles suivantes :

1 / Les atomes de $\mathcal{A}$ sont dans $\mathcal{F}$;
2 / Si A et B sont deux formules de $\mathcal{F}$, alors $(A \Rightarrow B)$ est aussi une formule de $\mathcal{F}$; $(A \Rightarrow B)$ se lit « A implique B ».

Pour introduire une relation de dérivation entre les formules (relation notée $\vdash_{\mathcal{I}_{mp}}$), on utilise le formalisme de la déduction naturelle, introduite par Gentzen en 1935 pour la théorie de la démonstration[3]. Γ désigne une liste d'hypothèses (avec d'éventuelles répétitions), et Γ, A représente la même liste à laquelle on a ajouté la formule A (dans une liste, l'ordre des formules est indifférent). Les règles de la déduction naturelle sont les suivantes :

1 / $\Gamma, A \vdash_{\mathcal{I}_{mp}} A$;
2 / Si $\Gamma, A \vdash_{\mathcal{I}_{mp}} B$, alors $\Gamma \vdash_{\mathcal{I}_{mp}} (A \Rightarrow B)$;
3 / Si $\Gamma \vdash_{\mathcal{I}_{mp}} (A \Rightarrow B)$ et si $\Gamma \vdash_{\mathcal{I}_{mp}} A$, alors $\Gamma \vdash_{\mathcal{I}_{mp}} B$.

1. Au moins dans la théorie simple des types que nous examinons.
2. Voir ci-dessus, chap. IV, p. 69, n. 1.
3. Gerhard Gentzen, Untersuchungen über das logische Schliessen, *Mathematische Zeitschrift*, 39, 1935 ; trad. franç. par R. Feys et J. Ladrière, *Recherches sur la déduction logique*, Paris, PUF, 1955 ; trad. anglaise *in* M. E. Szabo, *The collected papers of Gerhard Gentzen*, Amsterdam, North-Holland, 1969.

La règle 1 signifie que l'on peut dériver A à partir d'un ensemble d'hypothèses dont A fait partie. La règle 2 stipule les conditions requises pour l'introduction de l'implication dans une formule, et la règle 3 fait de même pour l'élimination de ce connecteur. Dans la règle 3, le cas où Γ est la liste vide correspond au *Modus Ponens*. On dit que A est un théorème du système $\mathcal{I}_{mp}$ ainsi défini si l'on peut obtenir $\vdash_{\mathcal{I}_{mp}} A$ à l'aide de ces trois règles. Par exemple, la formule $(p \Rightarrow (q \Rightarrow p))$ est un théorème (on écrit donc $\vdash_{\mathcal{I}_{mp}} (p \Rightarrow (q \Rightarrow p))$), ce qui se prouve de la manière suivante :

(i) $p, q \vdash_{\mathcal{I}_{mp}} p$ (Règle 1) ;
(ii) $p \vdash_{\mathcal{I}_{mp}} (q \Rightarrow p)$ ((i) et Règle 2) ;
(iii) $\vdash_{\mathcal{I}_{mp}} (p \Rightarrow (q \Rightarrow p))$ ((ii) et Règle 2).

La notation usuelle, qui ne fait pas apparaître Γ[1], est la suivante pour les règles 2 et 3 :

Règle d'introduction de l'implication :

$$\frac{\begin{array}{c}[A]^1 \\ \vdots \\ B\end{array}}{(A \Rightarrow B)}\ (\Rightarrow I)_1$$

Les crochets indiquent que la formule A est déchargée de la liste d'hypothèses au moment où la règle est appliquée.

Règle d'élimination de l'implication :

$$\frac{A \quad A \Rightarrow B}{B}$$

Pour faciliter l'étude de la relation de dérivation ($\vdash_{\mathcal{I}_{mp}}$), on se propose de marquer les formules démontrées par un terme qui rappelle les étapes de la démonstration. Or le lambda-calcul se pré-

1. Les deux présentations de la déduction naturelle que nous donnons ne sont pas tout à fait équivalentes. Nous n'examinerons pas ici leurs différences.

sente comme un outil parfaitement adapté à cet usage. La notation « $t : A$ » sert donc à marquer la formule A par le terme t et se lit maintenant : « t est une preuve de la proposition A ». Examinons de ce point de vue les trois règles du calcul logique.

Règle 1. Les hypothèses sont marquées à l'aide des variables, et l'ensemble Γ prend la forme d'une liste d'hypothèses ainsi marquées : $x_1 : A_1, \ldots, x_n : A_n$. Le terme x_i représente ici une preuve quelconque et indéterminée de la formule A_i. La règle 1 s'écrit donc :

$$\Gamma, x : A \vdash_{\mathcal{Imp}} x : A$$

et se comprend de la manière suivante : si la formule A, marquée par x, figure parmi les hypothèses, on peut dériver A, et la preuve de A est x.

Règle 2. Supposons que la preuve t permette de dériver la formule B à partir d'une preuve quelconque, x, de la formule A (avec éventuellement d'autres hypothèses dont la liste est Γ). Cela se note : $\Gamma, x : A \vdash_{\mathcal{Imp}} t : B$. On dispose donc d'une méthode qui à toute preuve x de A associe une preuve t de B (où t est, en général, un terme qui dépend de x). Selon la sémantique de Heyting, cette méthode est une preuve de la formule $(A \Rightarrow B)$, et il est naturel de la noter $\lambda x.t$, c'est-à-dire par l'opérateur du lambda-calcul qui à x associe t. La règle 2 s'écrit donc :

$$\text{Si } \Gamma, x : A \vdash_{\mathcal{Imp}} t : B, \text{ alors } \Gamma \vdash_{\mathcal{Imp}} \lambda x.t : (A \Rightarrow B).$$

Règle 3. Supposons que l'on ait à la fois une preuve de $(A \Rightarrow B)$, que l'on note t, et une preuve de A, que l'on note u. Selon la sémantique de Heyting, t est une méthode qui transforme toute preuve de A en une preuve de B. Il suffit donc d'appliquer t à u pour obtenir une preuve de B. Ici encore, la notation du lambda-calcul est parfaitement adaptée : $(t)u$ (c'est-à-dire t appliqué à u) est le terme qui marque la preuve de B. La règle 3 devient donc :

$$\text{Si } \Gamma \vdash_{\mathcal{Imp}} t : (A \Rightarrow B) \text{ et si } \Gamma \vdash_{\mathcal{Imp}} u : A,$$
$$\text{alors } \Gamma \vdash_{\mathcal{Imp}} (t)u : B.$$

Trois interprétations possibles du formalisme

Si l'on décide d'identifier les deux symboles « $\rightarrow$ » et « $\Rightarrow$ », on voit que les deux calculs, fonctionnel et logique, ne sont qu'un seul et même système formel qui peut recevoir deux interprétations. L'ensemble $\mathscr{B}$ des types de base correspond à l'ensemble $\mathscr{A}$ des atomes, celui des types, $\mathscr{T}$, à l'ensemble $\mathscr{F}$ des formules, et un lambda-terme de type A est l'analogue exact d'une preuve de la formule A. En outre, les symboles $\vdash$ et $\vdash_{\mathscr{I}mp}$ sont identifiés (seul le premier, dorénavant, sera utilisé; l'implication étant maintenant notée « $\rightarrow$ »). Dans la notation usuelle pour la déduction naturelle, les règles pour l'implication s'écrivent alors de la manière suivante :

Règle d'introduction de l'implication :

$$\frac{\begin{array}{c}[x : \mathrm{A}] \\ \vdots \\ t : \mathrm{B}\end{array}}{\lambda x.t : (\mathrm{A} \rightarrow \mathrm{B})}$$

Règle d'élimination de l'implication :

$$\frac{t : (\mathrm{A} \rightarrow \mathrm{B}) \quad u : \mathrm{A}}{(t)u : \mathrm{B}}$$

Pour chaque règle, deux lectures sont possibles. Par exemple, pour la règle d'élimination de l'implication :

(i) si le terme t est de type $(\mathrm{A} \rightarrow \mathrm{B})$ et si le terme u est de type A, alors l'application de t à u est de type B;

(ii) si t est une preuve de la proposition $(\mathrm{A} \rightarrow \mathrm{B})$, et si u est une preuve de la proposition A, alors, on peut construire une preuve de la proposition B, et cette preuve est le terme $(t)u$.

La correspondance de Curry-Howard accomplit ainsi la mise en rapport d'un système logique pour la formalisation des preuves et

d'un langage informatique pour la construction des machines abstraites que sont les programmes. Elle permet à la fois de dégager le contenu algorithmique des preuves (les preuves sont des lambda-termes, et donc des programmes) et d'exercer un contrôle logique sur les programmes écrits dans le système formel qui vient d'être construit. Ce système jouit en effet de la propriété de normalisation forte :

Théorème : Si $\Gamma \vdash t : A$, alors le lambda-terme t est fortement normalisable.

On s'assure ainsi par des moyens logiques qu'aucun programme ne boucle : tous les chemins de réduction des termes typables dans le système qui a été défini conduisent à la forme normale du terme en question. Bien que cela n'apparaisse pas clairement dans le système très primitif qui vient d'être présenté (du point de vue logique, on ne dispose que du calcul propositionnel restreint au connecteur de l'implication), le contrôle logique des programmes prend un sens plus fort lorsque la correspondance de Curry-Howard est étendue à des systèmes plus riches. Alors, l'expression « $t : A$ » ne se lit plus seulement « t est un terme de type A » et « t est une preuve de la formule A » mais également « A est la spécification du programme t ». Pour se faire une idée de ce qui justifie cette troisième interprétation, considérons l'exemple donné ci-dessus d'une preuve constructive de la formule $\forall x \exists y R[x,y]$. Si $R[x,y]$ exprime le fait que y est le résultat d'un tri effectué sur la liste x, la formule $\forall x \exists y R[x,y]$ est une spécification du programme qu'il est possible d'extraire de sa preuve. Le lambda-terme qui traduit la preuve est donc un programme dont le type (c'est-à-dire la formule démontrée) spécifie, comme s'il s'agissait d'un commentaire, ce qu'il permet de faire, ou quelle est la fonction qu'il permet de calculer. Dans un système suffisamment riche pour pouvoir exprimer une telle formule, on construit un programme (c'est-à-dire un lambda-terme, une machine abstraite) en écrivant une démonstration. Pourvu que le système soit fortement normalisable, cette démonstration assure à la fois que le programme obtenu ne boucle pas et qu'il effectue bien ce qui est spécifié par la formule démontrée[1].

1. On notera que cette manière de « prouver des programmes » (selon l'expression des logiciens-informaticiens) est relative à la consistance du système formel considéré.

L'extension de la correspondance à des systèmes plus riches

Pour comprendre comment le système initial peut être enrichi, précisons les modalités par lesquelles la conjonction est ajoutée au calcul formel qui vient d'être défini. Du point de vue logique, la déduction naturelle pose une règle d'introduction et deux règles d'élimination pour ce connecteur :

$$\frac{A \quad B}{A \wedge B} \qquad \frac{A \wedge B}{A} \qquad \frac{A \wedge B}{B}.$$

Selon la sémantique de Heyting, une preuve de $A \wedge B$ est un couple $\langle t, u \rangle$, où t est une preuve de A et u une preuve de B. Du point de vue informatique, la formule $(A \wedge B)$ est interprétée comme le type *produit*, $(A \times B)$, des types A et B. Les termes de type $(A \wedge B)$ seront des couples $\langle t, u \rangle$ de termes, avec $t : A$ et $u : B$. En outre, deux opérateurs notés π^1 (première projection) et π^2 (seconde projection) permettent d'extraire respectivement la première et la seconde composante d'un terme v de type $(A \wedge B)$. Il faut donc ajouter trois constructeurs de termes au calcul fonctionnel ($\langle \ , \ \rangle$, π^1 et π^2) et trois règles au système formel :

$$\frac{t : A \quad u : B}{\langle t, u \rangle : (A \wedge B)} \qquad \frac{t : (A \wedge B)}{\pi^1 t : A} \qquad \frac{t : (A \wedge B)}{\pi^2 t : B}.$$

Ces règles s'écrivent aussi :

Règle 4. Si $\Gamma \vdash t : A$ et $\Gamma \vdash u : B$, alors $\Gamma \vdash \langle t, u \rangle : (A \wedge B)$.
Règle 5. Si $\Gamma \vdash t : (A \wedge B)$, alors $\Gamma \vdash \pi^1 t : A$.
Règle 6. Si $\Gamma \vdash t : (A \wedge B)$, alors $\Gamma \vdash \pi^2 t : B$.

Chacune de ces règles se lit soit du point de vue logique (ce sont des règles d'inférence dans lesquelles chaque formule est marquée par un terme traduisant la preuve qui a permis de le dériver) soit du point de vue informatique (ce sont des règles de typage des termes du lambda-calcul).

On complète également la règle de calcul par β-réduction qui correspond à l'implication :

(i) $(\lambda x.t)u \rightarrow_\beta t[u/x]$;

par des règles de calcul pour les nouveaux constructeurs de termes :

(ii) $\pi^1 \langle t, u \rangle \rightarrow_\beta t$;
(iii) $\pi^2 \langle t, u \rangle \rightarrow_\beta u$.

Différentes voies se présentent pour la construction de systèmes formels plus riches qui prolongent la correspondance de Curry-Howard, et notamment une extension à d'autres connecteurs, à la quantification du premier ordre, ou aux logiques d'ordres supérieurs. Pour ne citer que quelques exemples, le système T de Gödel introduit deux constantes de types (Entier et Booléen), des constantes dans le lambda-calcul (pour la représentation des entiers et des booléens) et un opérateur de récursion. Le système F de Girard utilise un opérateur d'abstraction sur les types, et le système AF_2 dû à Krivine la quantification du second ordre. La théorie des types de Martin-Löf est une généralisation de l'arithmétique de Heyting qui se présente à la fois comme un système formel pour le fondement des mathématiques et comme un langage de programmation. La théorie des constructions de Coquand et Huet combine à la fois les moyens du système F de Girard et ceux de la théorie des types de Martin-Löf[1].

1. Sur le système T de Gödel, on pourra consulter Girard, *op. cit.*, p 442 et sq., et Girard, Lafont, Taylor, *op. cit.*, chap. 7 (le système T est défini par Gödel dans l'article suivant : Über eine bisher nicht benützte Erweiterung des finiten Standpunktes, *Dialectica*, 12, 1958, p. 280-287, traduction anglaise révisée et complétée par Gödel en 1972, *in* Gödel, *Collected Works*, vol. II, New York, Oxford, Oxford University Press, 1990) ; sur le système F, Girard, Lafont, Taylor, *ibid.*, chap. 11 ; sur le système AF_2, Krivine, *op. cit.* ; sur la théorie des types de Martin-löf, Per Martin-Löf, *Intuitionistic type theories*, Naples, Bibliopolis, 1984 ; Per Martin-Löf, Constructive mathematics and computer programming, *in* Cohen, Los, Pfeifer, Podewski (ed.), *Proceedings of the sixth International Congress for Logic, Methodology and Philosophy of Science*, Amsterdam, North-Holland, 1982, p. 153-175 ; et Thompson, *op. cit.* ; sur la théorie des constructions, Thierry Coquand et Gérard Huet, Concepts mathématiques et informatiques formalisés dans le calcul des constructions, in *Logic Colloquium '85*, Proceedings of the Colloquium held in Orsay, France, July 1985, ed. by The Paris Logic Group, Amsterdam, North-Holland, 1987, ou Thierry Coquand et Gérard Huet, The Calculus of Constructions, *Information and Computation*, 76, 1988.

LA NORMALISATION DES PREUVES ET L'EXÉCUTION DES PROGRAMMES

La simplification des preuves par contraction et les opérations de calcul par β-réduction

Plutôt que de poursuivre l'exposé des moyens par lesquels le système le plus primitif peut être enrichi (ce qui dépasse l'objet de cet ouvrage), il convient d'élucider un aspect de la correspondance de Curry-Howard qui n'a pas encore été considéré : celui des opérations de calcul et de l'exécution des programmes. Du point de vue informatique, le calcul est effectué par des opérations de β-réduction, et le résultat recherché est obtenu lorsque l'on arrive à un lambda-terme en forme normale. A cet égard, pour le fragment de système que nous avons considéré, les trois règles sont les suivantes :

(i) $(\lambda x.t)u \rightarrow_\beta t[u/x]$;
(ii) $\pi^1 \langle t, u \rangle \rightarrow_\beta t$;
(iii) $\pi^2 \langle t, u \rangle \rightarrow_\beta u$.

L'enrichissement du système implique l'ajout de nouvelles règles de calcul ; ainsi, par exemple, le système T de Gödel comporte une règle de récursion sur les entiers naturels. La question est de savoir à quel type d'opérations logiques correspond la réduction des lambda-termes.

Les deux redex $(\lambda x.t)u$ et $\pi^1 \langle t, u \rangle$ correspondent aux deux fragments de démonstration suivants écrits en déduction naturelle (dans la dérivation (1) ci-dessous, $(A \rightarrow B)$ peut être représentée par le terme $\lambda x.t$, et A par u ; dans la dérivation (2), la formule A peut être représentée par t et la formule B par u) :

(1)
$$\dfrac{\dfrac{\begin{array}{c}A\\ \vdots\ d\\ B\end{array}}{A \rightarrow B} \quad \begin{array}{c}\vdots\ d'\\ A\end{array}}{B}$$

et

(2)
$$\dfrac{\begin{array}{c}\vdots\ d\\ A\end{array} \quad \begin{array}{c}\vdots\ d'\\ B\end{array}}{A \wedge B}$$

Dans l'arbre (1), d représente une dérivation de B à partir de l'hypothèse A (qui peut avoir plusieurs occurrences dans d), et d' une dérivation de la formule A. Dans l'arbre (2), d est une dérivation de A et d' une dérivation de B.

(1) et (2) se réduisent, respectivement, aux arbres suivants :

$$(3)\quad \begin{array}{c} d' \\ \vdots \\ A \\ d \\ \vdots \\ B \end{array} \qquad \text{et} \qquad (4)\quad \begin{array}{c} d \\ \vdots \\ A \end{array}$$

(3) est obtenu à partir de (1) en remplaçant par la dérivation d' toutes les hypothèses A qui ont été déchargées au moment où la règle d'introduction de l'implication a été appliquée dans (1). Autrement dit, si t est la preuve de B (ce que l'on note t : B) obtenue en supposant donnée une preuve quelconque, x, de A (x : A, A est une hypothèse), et si le terme u représente la preuve de A obtenue grâce à la dérivation d' (u : A), alors, on obtient l'arbre (3) à partir de l'arbre (1) en substituant u à x dans t, c'est-à-dire par $t[u/x]$. On voit que la transformation de (1) en (3) en déduction naturelle (on parle d'une « contraction » de la preuve) correspond exactement à l'opération de β-réduction : $(\lambda x.t)u \rightarrow_\beta t[u/x]$ du lambda-calcul. De la même manière, en ce qui concerne la conjonction, la contraction de (2) en (4) correspond exactement à la réduction $\pi^1 \langle t, u \rangle \rightarrow_\beta t$.

La normalisation des preuves en déduction naturelle

Or la transformation des preuves par contractions a été étudiée de manière tout à fait indépendante du lambda-calcul, dans le cadre de la théorie de la démonstration. Étendue à l'ensemble des opérations logiques (connecteurs et quantificateurs), elle rend possible la normalisation des preuves en déduction naturelle. Du point de vue de la théorie de la démonstration, l'idée de normalisation n'est pas comprise comme une opération de calcul. Le problème est plutôt de construire une preuve en forme canonique à partir de l'une des nombreuses preuves possibles de telle ou telle formule, dont certaines pas-

sent par des détours apparemment inutiles. Un exemple de ces détours est donné par l'arbre de démonstration (2) ci-dessus : la dérivation d' de la formule B ne sert à rien pour démontrer A si l'on dispose déjà d'une dérivation d de A. De la même manière, dans l'arbre (1) on introduit le connecteur de l'implication pour le supprimer aussitôt après. L'opération de normalisation a pour but de réduire les preuves à une forme canonique en supprimant tous les détours semblables à ceux qui viennent d'être mentionnés.

Si ces détours peuvent sembler superflus, notamment dans l'exemple du schéma (2), il convient de souligner qu'ils sont très importants dans les raisonnements mathématiques usuels. Dans le cas de l'implication par exemple, supposons que $(A \rightarrow B)$ soit un théorème connu. Chaque fois que l'on dispose d'une démonstration de la formule A, on peut l'utiliser directement pour prouver B par application du *Modus Ponens* (ou de la règle d'élimination de l'implication). En revanche, si l'on veut normaliser la preuve de B, et donc éviter le détour par $(A \rightarrow B)$, il faut construire pour chaque démonstration de A une nouvelle démonstration de B, conformément à la contraction de (1) en (3). Et comme l'hypothèse A peut avoir plusieurs occurrences dans la dérivation de $(A \rightarrow B)$, la preuve normalisée peut être beaucoup plus longue que la preuve initiale. Cela montre que les détours permettent souvent de rendre les preuves plus courtes, plus générales et plus élégantes. L'importance de la normalisation ne vient donc pas de son utilité pratique mais du rôle qu'elle joue dans la solution de problèmes posés en théorie de la démonstration, et des propriétés dont jouissent les preuves en forme canonique. La normalisation est notamment utilisée pour prouver la cohérence de certaines théories mathématiques[1].

1. En 1935, dans l'article cité, Gentzen se sert du théorème de normalisation pour le calcul des séquents (voir ci-dessous) pour prouver la cohérence de l'arithmétique sans induction complète, puis en 1938, pour prouver la cohérence de l'arithmétique (cf. Largeault, *op. cit.*, textes XX et XXI). Au début des années soixante-dix, Girard démontre la normalisation forte du système F, établissant ainsi une preuve de cohérence pour l'arithmétique du second ordre (cf. J. E. Fenstad (ed.), *Proceedings of the Second Scandinavian Logic Symposium*, Amsterdam, North-Holland, 1971, p. 63-92).

Peut-elle être obtenue dans le cas général ? On se souvient que le programme de Hilbert visait un idéal de conservativité pour les preuves de l'arithmétique : réduire les démonstrations qui utilisent des méthodes idéales ou abstraites à des démonstrations réelles ou concrètes. Après la mise en échec du programme de Hilbert par les travaux de Gödel, la normalisation apparaît comme une autre manière de concevoir la pureté des méthodes dans les démonstrations. Historiquement, c'est Gentzen qui prouve le premier, en 1935, que cet idéal peut être atteint sous certaines conditions dans un formalisme logique appelé « calcul des séquents »[1]. En 1965, Prawitz étudie, en s'inspirant des travaux de Gentzen, la normalisation des preuves en déduction naturelle, et démontre le théorème suivant[2].

> Théorème de normalisation : si la formule A est démontrable dans le système de la déduction naturelle, alors il existe une preuve canonique de A (une preuve sans détours).

En outre, il existe une méthode effective permettant de normaliser une preuve quelconque, de même qu'il existe une méthode effective pour réduire les termes du lambda-calcul à une forme normale. Le théorème précédent peut même être renforcé par l'énoncé du

> Théorème de normalisation forte : quel que soit l'ordre dans lequel les règles de contraction sont appliquées, on obtient une preuve canonique après un nombre fini de ces transformations.

Enfin, il est possible de démontrer pour la déduction naturelle l'équivalent du théorème de Church-Rosser pour le lambda-

1. Ce point est précisé ci-dessous. L'idée de normalisation conçue comme une élimination des détours dans les preuves est exposée par Gentzen lui-même dans l'article de 1935 : « On peut formuler les propriétés les plus essentielles d'une telle démonstration normale à peu près de la façon suivante : elle ne comporte pas de détour *[Er macht keine Umwege]*. On n'y introduit aucun concept qui ne soit pas contenu dans son résultat final et qui, par conséquent, ne doive pas nécessairement être utilisé pour obtenir ce résultat » (Gentzen, *op. cit.*, p. 177 (p. 4-5 de la trad. franç.)). Cette citation indique en quel sens on peut parler de pureté des méthodes au sujet des preuves normalisées.
2. Dag Prawitz, *Natural Deduction. A proof theoretical study*, Stockholm, Almquist & Wiksell, 1965 ; et Dag Prawitz, Ideas and results in proof-theory, *in* J. E. Fenstad (ed.), *Proceedings of the second scandinavian logic symposium*, Amsterdam, North-Holland, 1971.

calcul[1], ce qui permet d'établir que tous les chemins de normalisation possibles pour une preuve donnée conduisent en définitive à la même preuve en forme normale.

L'ensemble de ces résultats montre qu'il existe une stricte correspondance entre la normalisation des preuves en déduction naturelle (l'élimination des détours) et la β-réduction d'un lambda-terme typé (l'exécution d'un programme par élimination des redex). Historiquement, la syntaxe de la déduction naturelle, l'idée de normalisation et les propriétés des preuves normalisées ont été étudiées d'un point de vue strictement logique, et n'avaient *a priori* aucun rapport avec les notions de calcul, de machine et de programme. Pourtant, les termes du lambda-calcul typé peuvent être mis en rapport avec les démonstrations en déduction naturelle, en sorte que les formules prouvées correspondent précisément aux types des lambda-termes, et la normalisation des preuves à leur transformation par β-réduction, c'est-à-dire à une suite d'opérations de calcul réglée par un programme. Nous trouvons donc ici un nouvel exemple d'outils, de méthodes et de résultats élaborés dans le cadre logique de la théorie de la démonstration avant la construction des premiers ordinateurs, et qui reçoivent ultérieurement une interprétation informatique en termes de calculs, de machines et de programmes. L'interprétation informatique des constructions et des concepts logiques ne se limite donc pas à la théorie de la calculabilité. Par la correspondance de Curry-Howard, elle concerne également en un sens très fort la notion de démonstration, centrale en logique.

En outre, on voudrait montrer ici encore que l'importance logique de la notion de machine ne vient pas simplement de la singularité d'un formalisme comme celui de la déduction naturelle. Les rapports entre informatique et théorie de la démonstration touchent également des constructions formelles et des théorèmes aussi importants que le théorème fondamental de Herbrand, le calcul

1. On trouvera une démonstration de ce fait par exemple dans Girard, *op. cit.*, chap. 2, annexe B.

des séquents ou le *Hauptsatz* de Gentzen, qui datent tous des années trente, et qui reçoivent tous aujourd'hui une interprétation informatique en termes de machines et de programmes. Pour mettre ce point en évidence dans les paragraphes qui suivent, on considérera le cas du calcul des séquents et le théorème de Gentzen sur l'élimination des coupures.

Le calcul des séquents et l'élimination des coupures

Bien que le théorème de normalisation pour la déduction naturelle soit dû à Prawitz, c'est Gentzen qui conçut ce formalisme en 1935 pour les besoins de la théorie de la démonstration. Dans ses *Recherches sur la déduction logique*, Gentzen introduit également une autre syntaxe pour l'étude des preuves formelles, le calcul des séquents. Un *séquent* est une expression de la forme $\Phi \Rightarrow \Psi$ où Φ et Ψ sont des suites finies et ordonnées de formules. L'interprétation intuitive est la suivante : la conjonction des formules de Φ a pour conséquence la disjonction des formules de Ψ[1]. Le calcul est formé de règles qui définissent les conditions auxquelles il est possible de déduire un séquent à partir d'un ou de plusieurs autres séquents. Donnons à titre d'exemples deux règles du calcul pour l'opérateur logique de la conjonction et une pour l'implication (C et D sont des formules quelconques) :

$$\frac{\Phi, C \Rightarrow \Psi}{\Phi, C \wedge D \Rightarrow \Psi} \qquad \frac{\Phi \Rightarrow C, \Psi \quad \Phi' \Rightarrow D, \Psi'}{\Phi, \Phi' \Rightarrow C \wedge D, \Psi, \Psi'} \qquad \frac{\Phi, C \Rightarrow D, \Psi}{\Phi \Rightarrow C \rightarrow D, \Psi}$$

La première de ces règles peut se lire : s'il est possible de déduire l'une des formules de Ψ à partir des formules de Φ et de C, alors, il est possible de déduire l'une des formules de Ψ à partir des formules

1. Ici, le symbole « $\Rightarrow$ » met en relation des suites de formules. Il ne s'agit donc pas du tout d'un connecteur propositionnel (bien que le même signe ait été temporairement utilisé ci-dessus pour représenter l'implication).

de Φ et de $C \wedge D$. On se servira ci-dessous des deux règles suivantes pour le quantificateur universel :

$$\frac{\Phi \Rightarrow A, \Psi}{\Phi \Rightarrow \forall x A, \Psi} \qquad \frac{\Phi, A[t] \Rightarrow \Psi}{\Phi, \forall x A[x] \Rightarrow \Psi}$$

Dans la première de ces règles, la variable x ne doit être libre ni dans Φ ni dans Ψ[1].

Comme en déduction naturelle, les preuves sont représentées par des arbres formés à partir des règles du calcul. Les axiomes logiques sont des séquents qui ont la forme suivante : $A \Rightarrow A$, où A est une formule. Des règles sont posées pour chacun des opérateurs logiques (connecteurs et quantificateurs), auxquelles s'ajoutent les règles structurelles[2] et enfin la règle de coupure qui s'écrit :

$$\frac{\Phi \Rightarrow C, \Psi \quad \Phi', C \Rightarrow \Psi'}{\Phi, \Phi' \Rightarrow \Psi, \Psi'}$$

Une formule A est démontrée par le calcul des séquents lorsqu'il existe un arbre dont le sommet est le séquent $\Rightarrow A$ et dont les feuilles sont toutes des axiomes logiques. Ainsi, l'arbre suivant est une preuve de la formule $(p \rightarrow (q \rightarrow p))$:

$$\cfrac{\cfrac{\cfrac{p \Rightarrow p}{p, q \Rightarrow p}}{p \Rightarrow (q \rightarrow p)}}{\Rightarrow (p \rightarrow (q \rightarrow p))}$$

Axiome
Règle structurelle d'affaiblissement
Règle pour l'implication
Règle pour l'implication.

1. L'interprétation de cette règle est la suivante : si à partir des formules de Φ on peut démontrer soit une formule de Ψ soit la formule A pour un *x quelconque* (en général, x est libre dans A), alors, à partir des formules de Φ on peut démontrer soit une formule de Ψ soit la formule $\forall x A$.
2. Donnons deux exemples de règles structurelles :

$$\frac{\Phi \Rightarrow \Psi, A, B, \Psi'}{\Phi \Rightarrow \Psi, B, A, \Psi'} \qquad \frac{\Phi \Rightarrow \Psi}{\Phi, A \Rightarrow \Psi}$$

La première (règle d'échange) autorise la permutation de deux formules ; la seconde est une règle d'affaiblissement : s'il est possible de déduire l'une des formules de Ψ à partir des formules de Φ, alors il est possible de déduire l'une des formules de Ψ à partir des formules de Φ et de A.

Dans le calcul des séquents, la règle de coupure joue un rôle similaire aux détours signalés ci-dessus au sujet de la déduction naturelle : elle permet d'appliquer facilement des résultats généraux à des cas particuliers. Supposons par exemple que l'on ait démontré la formule A[x] pour un entier x quelconque. On a donc une dérivation (c'est-à-dire une démonstration) d du séquent $\Rightarrow$A[x]. En appliquant la première des deux règles énoncées ci-dessus pour le quantificateur universel, on obtient la dérivation suivante (dans le cas présent, Φ et Ψ sont des suites vides) :

$$\dfrac{\begin{array}{c} d \\ \Rightarrow \mathrm{A}\,[x] \end{array}}{\Rightarrow \forall x \mathrm{A}[\mathrm{x}]}$$

La règle de coupure permet alors d'appliquer facilement ce théorème général pour démontrer le cas particulier A[a] :

$$\dfrac{\dfrac{\begin{array}{c} d \\ \Rightarrow \mathrm{A}[x] \end{array}}{\Rightarrow \forall x \mathrm{A}[\mathrm{x}]} \qquad \dfrac{\mathrm{A}[a] \Rightarrow \mathrm{A}[a]}{\forall x \mathrm{A}[\mathrm{x}] \Rightarrow \mathrm{A}[a]}}{\mathrm{A}[\mathrm{a}]}$$

(axiome)
(règle pour $\forall$)
(coupure)

Pour démontrer A[a] sans utiliser la règle de coupure, il faut transformer la dérivation d de A[x] en substituant à x le cas particulier a, ce que l'on peut noter informellement : $d[a/x]$. On voit que si l'on ne dispose pas de la règle de coupure, dans chaque cas particulier, pour démontrer A[i], il faut transformer la dérivation d en une dérivation $d[i/x]$; *a contrario*, la règle de coupure permet d'utiliser directement le résultat général $\forall x$A[x]. L'élimination des coupures dans une preuve en calcul des séquents consiste donc, comme la normalisation en déduction naturelle, à supprimer tous les détours de la preuve pour obtenir une preuve canonique.

Le principal théorème pour le calcul des séquents est démontré par Gentzen dans l'article de 1935 :

> Théorème d'élimination des coupures *(Hauptsatz)* : si le séquent $\Phi \Rightarrow \Psi$ est démontrable, alors il existe une démonstration de $\Phi \Rightarrow \Psi$ qui n'utilise pas la règle de coupure.

En outre, la preuve de ce théorème fournit une procédure effective permettant de transformer toute démonstration en une démonstration sans coupure. Cependant, contrairement au formalisme de la déduction naturelle, le calcul des séquents tel qu'il est formulé par Gentzen ne satisfait aucun énoncé équivalent au théorème de Church-Rosser pour le lambda-calcul. En conséquence, l'élimination des coupures d'une preuve écrite dans le calcul des séquents peut produire plusieurs preuves sans coupure différentes du même séquent. On dit aussi, plus brièvement, que l'élimination des coupures n'est pas convergente.

Du point de vue informatique, cette absence de convergence est comprise comme un défaut du calcul des séquents de Gentzen puisqu'elle correspond à une procédure de calcul, ou une machine, qui aurait la possibilité d'appliquer des règles dans un ordre quelconque et pourrait ainsi produire des résultats différents selon l'ordre choisi. Une telle situation signifie simplement que l'élimination des coupures dans le calcul des séquents conçu par Gentzen ne correspond pas à ce que l'on entend habituellement par « procédure effective », « calcul » ou « machine ». On voit ici sur un exemple précis en quel sens une préoccupation informatique peut être féconde pour la recherche en logique. Car les logiciens n'ont pas manqué de s'interroger pour savoir comment le calcul des séquents pouvait être modifié en sorte qu'il satisfasse la propriété de convergence et celle de normalisation forte. Le calcul des séquents de Gentzen peut être modifié de multiples manières, et chaque modification particulière définit un nouveau système logique, caractérisé par de nouvelles règles. Par exemple, on peut limiter à un le nombre des formules dans la partie droite de chaque séquent, et l'on obtient ainsi un système adéquat pour la logique intuitionniste[1]. En modifiant les règles structurelles du calcul, il est possible de définir un système pour la logique

1. Les différentes versions modifiées du calcul des séquents ne répondent évidemment pas nécessairement à des préoccupations informatiques. Le calcul intuitionniste est considéré dès 1935 par Gentzen lui-même (Gentzen, *op. cit.*).

linéaire[1]. Le calcul peut aussi être complété par l'introduction de règles pour les quantificateurs du second ordre, ou encore par l'ajout de règles équationnelles (c'est-à-dire qui concernent la relation d'égalité). Or certains des calculs obtenus par de telles modifications satisfont effectivement les propriétés de normalisation forte et de confluence.

CONCLUSION : LOGIQUE ET INFORMATIQUE

Ces deux propriétés fondamentales sont évidemment loin d'être les seules que l'on cherche à satisfaire en concevant de nouvelles versions modifiées du calcul des séquents de Gentzen. Ce qu'il importe ici de souligner, c'est qu'à travers la correspondance de Curry-Howard, le point de vue de l'informatique et des machines introduit dans la recherche en logique des principes heuristiques qui ne se présentent plus seulement comme des exigences de vérité, de cohérence, ou de modélisation du raisonnement humain. Ainsi, à la question de la cohérence logique vient s'ajouter celle d'une cohérence algorithmique : les différentes suites d'opérations possibles pour éliminer les coupures d'une preuve convergent-elles vers un seul et même résultat ? Plus généralement, les propriétés que l'on cherche à satisfaire en concevant un système logique concernent autant les machines, les algorithmes ou les calculs que la vérité, le raisonnement, ou la modélisation du langage et du raisonnement humains. Outre la normalisation forte et la confluence, les questions peuvent porter par exemple sur la capacité d'expression du système ou la possibilité de contrôler par des moyens logiques la complexité algorithmique des programmes : quels sont les types de données, quelles sont les

1. Jean-Yves Girard, Linear Logic, *Theoretical Computer Science*, 50, 1987.

fonctions qui sont représentables[1] ? Pour les fonctions représentables dans un système donné, quels sont les algorithmes qu'il est possible d'extraire à partir d'une preuve ? La procédure d'élimination des coupures qui est donnée par la preuve du *Hauptsatz* de Gentzen est d'une complexité algorithmique trop élevée pour pouvoir être effectivement utilisée comme une méthode de calcul[2]. D'où le problème suivant : comment concevoir les règles logiques d'un formalisme en sorte qu'il jouisse de propriétés comme la convergence et la normalisation forte, et qu'en outre la complexité algorithmique des programmes extraits à partir de preuves puisse être aussi faible que possible ?

Ces quelques questions suffisent à montrer en quel sens les recherches logiques peuvent être orientées par des interrogations relatives aux machines et aux algorithmes, qui complètent, sans nécessairement les contredire, les exigences logiques liées à la formalisation des raisonnements, et notamment du raisonnement mathématique. Et cette orientation de la logique ne concerne pas seulement la théorie de la calculabilité. A travers la correspondance de Curry-Howard, c'est la théorie de la démonstration elle-même qui peut être considérée comme une étude des machines et à certains égards de la complexité algorithmique. Cela ne signifie pas que les outils et les méthodes de la logique soient simplement mis au service des problèmes informatiques ou appliqués à l'étude des machines. Une telle interprétation des relations entre logique et informatique serait par trop restrictive et ne prendrait pas en considération toute la portée de l'isomorphisme de Curry-Howard. Par cette correspondance, les problèmes d'informatique théorique deviennent des problèmes de logique à part entière et inversement. C'est l'objet même de la logique et son orientation

1. Par exemple, les fonctions représentables dans le système T de Gödel sont exactement celles dont on peut démontrer qu'elles sont totales dans l'arithmétique de Peano (PA), alors que les fonctions représentables dans le système F de Girard sont exactement celles dont on peut démontrer qu'elles sont totales dans l'arithmétique de Peano du second ordre (PA_2) ; sur ce point, voir Girard, Lafont, Taylor, *op. cit.*, p. 53 et 124.
2. Girard, Lafont, Taylor, *op. cit.*, p. 111-112.

qui prennent des formes nouvelles et quelque peu inattendues si l'on se réfère à son histoire jusqu'au développement récent de l'informatique.

Bien que les pages qui précèdent montrent suffisamment en quel sens la logique prend pour objet les machines, les programmes et leur complexité algorithmique, elles sont loin de dresser un tableau exhaustif des relations qui existent entre la logique et l'informatique. Un examen plus complet s'attarderait sur la méthode de résolution de Robinson, utilisée pour la démonstration automatique dans des langages de programmation comme Prolog, et fondée sur les travaux de Gentzen concernant le calcul des séquents[1]. Il montrerait l'importance et l'interprétation informatiques des travaux de Herbrand, ainsi que ses rapports avec le *Hauptsatz* de Gentzen[2]. Il expliciterait les motivations logiques et informatiques de la logique linéaire[3]. Il ne se limiterait pas à la programmation fonctionnelle conçue sur le modèle du lambda-calcul mais considérerait aussi, par exemple, le fondement logique de l'interrogation des bases de données relationnelles. Il approfondirait l'étude logique de la complexité algorithmique[4]. Au-delà des théories de la calculabilité et de la démonstration, il montrerait qu'il existe une étude *sémantique* des programmes et des langages de programmation[5]. Ou encore, il compléterait la correspondance de Curry-Howard en cherchant à donner une interprétation informatique non seulement aux systèmes de règles logiques étudiés en théorie de la démonstration, mais encore aux axiomes utilisés pour la formalisation des théories mathématiques. Cette énumération elle-même

1. Cf. Girard, Lafont, Taylor, *op. cit.*, p. 112-113, et Jean Gallier, *op. cit.*
2. Sur les rapports entre les travaux de Gentzen et de Herbrand, cf. Jean van Heijenoort, L'œuvre logique de Jacques Herbrand et son contexte historique, *in* J. Stern (ed.), *Proceedings of the Herbrand Symposium. Logic Colloquium '81,* Amsterdam, North-Holland, 1982 ; cf. aussi Girard, *Proof theory and logical complexity,* chap. 2. Sur l'importance informatique des résultats obtenus par Herbrand, cf. par exemple Gallier, *op. cit.*, ou Lalement, *op. cit.*
3. Girard, Linear Logic, *op. cit.* ; Girard, Lafont, Taylor, *op. cit.*, Appendice B.
4. Hopcroft et Ullman, *op. cit.* ; Lalement, *op. cit.* ; Richard Lassaigne et Michel de Rougemont, *Logique et complexité,* Paris, Hermès, 1996 ; Mosconi, *op. cit.*, chap. XII.
5. Glynn Winskel, *The formal semantics of programming languages. An introduction,* Cambridge, Mass., The MIT Press, 1993.

n'est pas exhaustive[1] ; elle donne seulement quelques indications sur la manière dont les analyses qui précèdent pourraient être prolongées. Ensemble, elles permettent cependant de comprendre de manière suffisamment claire en quel sens la logique est liée à l'informatique et pourquoi elle ne traite pas seulement de la vérité, du langage, du raisonnement ou des paradoxes, mais également des machines, des programmes et de la complexité des algorithmes.

1. Pour la compléter, on pourra consulter Abramsky, Gabbay, Maibaum, *op. cit.* (*i.e. Handbook of Logic in Computer Science*, vol. 1 à 6).

IX

LA PENSÉE MÉCANIQUE

POSITION DU PROBLÈME

Depuis les années trente jusqu'à aujourd'hui, la machine est progressivement devenue l'un des objets majeurs de l'analyse logique. Les chapitres qui précèdent montrent suffisamment en quel sens cette affirmation doit être comprise. Mais avant d'examiner quelques conséquences philosophiques de ce fait, rappelons deux idées principales qui peuvent maintenant être considérées comme acquises, et qui permettent d'écarter plusieurs préjugés communément reçus.

(i) L'étude des ordinateurs, des calculateurs et des systèmes informatiques n'est pas une simple application, secondaire et annexe, de l'analyse logique des machines. A la différence de ce qu'elle pouvait être en 1936 dans l'article de Turing, la machine en logique n'est plus seulement, aujourd'hui, un outil mathématique abstrait dont la fonction principale serait la solution de certains problèmes posés par les logiciens. Parce qu'il existe une correspondance précise entre les machines de Turing, les termes du lambda-calcul, les calculateurs universels et les programmes informatiques, les machines réelles sont directement concernées par les résultats obtenus en théorie de la calculabilité. En outre, un grand nombre des travaux élaborés en logique mathématique – y compris à une époque qui précède la conception des premiers ordinateurs – peut recevoir une interprétation informatique ; et à l'inverse, des ques-

tions purement informatiques à l'origine comme la complexité algorithmique, la conception des langages de programmation, la terminaison des calculs ou la faisabilité d'un algorithme viennent féconder les recherches logiques et leur donner une seconde vie que les préoccupations relatives au fondement et à la formalisation du raisonnement mathématique ne laissaient pas prévoir. Entre logique et informatique, il existe à la fois d'étroites relations et un enrichissement mutuel dont la correspondance de Curry-Howard est certainement l'expression la plus fine et la plus profonde.

(ii) En logique, la machine est beaucoup plus qu'un objet particulier de la théorie de la calculabilité, ou qu'un outil *ad hoc* pour la solution négative de l'*Entscheidungsproblem*. Loin d'être une construction ou une idée marginale, elle apparaît comme l'une des notions centrales de la logique contemporaine. Rappelons les trois principales raisons qui justifient cette affirmation.

1/Machines abstraites et formalismes logiques. Les machines abstraites sont définies comme des systèmes de signes caractérisés par une syntaxe et sont à cet égard tout à fait comparables aux systèmes logiques conçus pour la formalisation des raisonnements. Le point de vue logique de la métamathématique permet d'étudier aussi bien ces machines qu'un système formel pour une théorie mathématique. Et les méthodes sémantiques peuvent être appliquées autant aux machines abstraites que sont les programmes et les langages informatiques qu'aux langages logiques et aux théories formalisées.

2/La thèse de Church-Turing. Un système formel peut être défini par une procédure effective permettant de déterminer si une suite quelconque de formules est une démonstration. En conséquence, par la thèse de Church-Turing, les machines abstraites permettent de donner la définition la plus générale d'un système formel : toute procédure mécanique pour la production de formules (que l'on appelle alors des théorèmes) peut être considérée comme un système formel et inversement, tout système formel peut être ramené à une machine capable de produire des théorèmes.

3 / La correspondance de Curry-Howard. Si les procédures mécaniques, ou algorithmes, servent à définir les systèmes formels en général, elles peuvent elles-mêmes être formalisées dans ce que l'on appelle un « calcul fonctionnel », comme le sont les preuves dans les formalismes appelés « calculs logiques ». Or cette double formalisation révèle l'existence d'une correspondance entre les preuves et les programmes, les formules et les types de données, les opérations de calcul et la normalisation des preuves. Elle met aussi en évidence le contenu algorithmique des preuves. Ce sont alors les démonstrations formelles elles-mêmes qui sont conçues comme des machines.

Ces trois idées très générales permettent de comprendre pourquoi un grand nombre de constructions et de résultats fondamentaux élaborés ou obtenus en logique mathématique peuvent être interprétés en termes de machines, d'algorithmes, et de programmes ; pourquoi, également, les relations entre logique et informatique ne se limitent pas à la théorie de la calculabilité mais concernent aussi bien la théorie de la démonstration et la théorie des modèles.

Pourtant, avant la publication de l'article de Turing, et même encore plusieurs années après, il était loin d'être évident que les machines et la logique dussent nouer des liens aussi étroits. On connaissait la tradition des machines logiques à la Jevons capables de tirer mécaniquement les principales conséquences de quelques prémisses explicitement posées au préalable, grâce à un dispositif physique qui reflète la situation logique et la déduction. Mais ces machines n'eurent guère de postérité et ne sont pas à l'origine des relations qui existent aujourd'hui entre logique et informatique. Comme on l'a vu, c'est en retraçant l'histoire de la formalisation des raisonnements, celle des calculs arithmétiques, algébriques et logiques, et celle, enfin, des machines à calculer, mécaniques puis électroniques, que l'on peut suivre les fils qui se nouent, à partir de 1936, entre la logique d'une part et les machines de l'autre.

On peut légitimement s'étonner de l'importance acquise en logique par une notion qui ne fut introduite dans ce domaine

qu'à une époque relativement récente. Car dans la mesure où la logique se définit traditionnellement comme une discipline de la pensée, du raisonnement, de la vérité ou du langage, elle ne semble pas prédisposée, *a priori*, à prendre la forme d'une étude des machines et des algorithmes. Cette orientation générale d'une grande partie de la logique contemporaine pose donc le problème de son unité et de son objet principal. La question se pose tant au sujet de la logique dans sa forme actuelle que du point de vue de son évolution historique, de l'Antiquité jusqu'à aujourd'hui, et elle intéresse à la fois l'historien des sciences et le philosophe. En effet, en faisant des machines l'un des objets majeurs de la logique contemporaine, les développements de cette discipline posent le problème des rapports entre le raisonnement, la vérité, la connaissance et le langage d'une part, le calcul, les algorithmes et les machines de l'autre.

Pourtant, force est de constater que les philosophes ne posent habituellement pas le problème en ces termes. Il est vrai que la philosophie de l'esprit élabore à partir des machines logiques et des systèmes informatiques une réflexion concernant la connaissance des phénomènes psychiques, la cognition, la nature des états mentaux et leur rapport avec les états physiques, le fonctionnement de l'esprit, le fondement de la psychologie, ou encore la pensée des machines. Mais le chapitre VII a suffisamment montré que l'ensemble de cette problématique ne prend pas véritablement en considération la machine en logique. Elle fait seulement usage de certaines idées et de certains résultats logiques bien précis au sujet des machines dans l'espoir d'éclairer plusieurs problèmes relatifs à l'esprit et la psychologie. Ainsi, de ce point de vue, une construction aussi fondamentale que la correspondance de Curry-Howard est entièrement ignorée. D'autre part, les philosophes qui attachent une grande importance à l'analyse logique s'intéressent généralement à des problèmes dont la solution ne requiert que très exceptionnellement un examen de l'idée de machine. Considérons les réflexions qui s'appuient sur la logique et l'analyse du langage pour clarifier les concepts, lever une difficulté de raisonnement, expli-

quer les paradoxes, ou dans le but de résoudre ou dissoudre certains problèmes philosophiques ; celles qui interrogent d'un point de vue logique la vérité, le sens, la traduction, la possibilité d'une théorie de la signification, la référence, ou l'opposition entre réalisme et antiréalisme ; celles qui recherchent dans la logique la définition d'une ontologie ou d'une grammaire de la réalité ; ou encore, celles qui portent sur la formalisation des connaissances, des croyances, ou d'autres aspects du discours et de la pensée humaine ; elles mettent surtout en valeur l'importance de la philosophie du langage, et n'ont, de fait, que rarement conduit le philosophe à s'intéresser de près à la question de la machine en logique dans toute sa généralité. On ne s'étonnera donc pas que des ouvrages d'introduction ou de synthèse sur la philosophie de la logique puissent passer presque entièrement sous silence la notion de machine et la question des rapports entre logique et informatique.

Ainsi, ni la philosophie de l'esprit ni la philosophie de la logique telles qu'elles sont habituellement conçues ne traitent la question de la machine en considérant, de manière globale, l'ensemble des rapports qui existent entre logique et informatique. La machine en logique semble donc soulever un problème philosophique spécifique qui demande encore à être simplement posé. Le lecteur voudra bien considérer la suite de ce chapitre comme un essai pour répondre à cette demande, par la recherche d'une problématique philosophique capable de prendre en considération dans leur globalité les relations entre la logique d'une part, les machines et l'informatique d'autre part. Avant même d'en expliciter le sens et sans prétendre pouvoir l'énoncer d'emblée en toute précision, on peut proposer une première formulation de cette problématique de la manière suivante : conformément à une conception traditionnelle, la logique peut être considérée, très généralement, comme une discipline relative à certaines conditions de possibilité de la connaissance et de la pensée vraie ; mais en tant qu'elle s'oriente vers une analyse des machines et des programmes, elle n'est pas tant discipline de la pensée humaine que d'une pensée *mécanique*. De cette notion, la philosophie de l'esprit

ne dit rien, alors qu'elle examine de près, comme on l'a vu, les projets, les ambitions et les thèses de l'intelligence artificielle ; cela vient de ce que ses analyses portent essentiellement sur les problèmes relatifs à l'esprit et la pensée humaine, et non sur des questions suscitées par les machines et les programmes eux-mêmes. On voudrait mettre en évidence qu'*a contrario* le point de vue de la machine donne à la logique contemporaine la forme d'une discipline de la pensée mécanique[1].

Pour cela, considérons maintenant à nouveau la problématique propre à la philosophie de l'esprit dont il a été question au chapitre VII. On s'apercevra qu'il existe une idée qu'elle évite et contourne soigneusement, comme un objet étranger à son domaine : l'idée de pensée mécanique, précisément, sur laquelle porte cette partie de la logique qui est orientée vers les machines et les algorithmes.

La pensée des machines

En philosophie de l'esprit, les réflexions qui prennent appui sur la thèse de Church-Turing, l'intelligence artificielle et les machines logico-informatiques peuvent être distribuées autour des deux interrogations suivantes :

(i) dans quelle mesure les machines peuvent-elles nous éclairer sur la pensée ou sur l'esprit et son fonctionnement ?

1. Les analyses de la philosophie de l'esprit concernent la pensée humaine qui est conçue, dans ce cas, comme un vécu psychologique, un état ou un processus mental, ou encore une faculté. Mais lorsque l'on dit que la logique est une discipline de la pensée, ce terme est compris en un autre sens ; car la logique détermine les lois de la pensée vraie et les règles du raisonnement correct sans rapporter ces lois et règles au vécu psychologique des êtres pensants. Nous verrons ci-dessous quelle position singulière occupe la pensée mécanique tant à l'égard de la logique que de la philosophie de l'esprit ou de la psychologie.

(ii) quelles sont les capacités des machines relativement à celles d'un être humain en tant qu'il est doué de pensée, et les machines pourraient-elles devenir elles-mêmes capables de penser ?

La première interrogation se décline en une multitude de questions qu'on a déjà mentionnées : La pensée humaine est-elle réductible à une forme de calcul ? Les capacités de l'esprit peuvent-elles être programmées ? Une psychologie cognitive est-elle possible ?, etc. Chacune ajoute un trait supplémentaire au dessin d'une problématique qui articule le projet d'une science cognitive des phénomènes psychiques et la recherche d'une conception philosophique de l'esprit fondée sur l'idée de cognition. S'il ne se place pas lui-même dans la mouvance des sciences cognitives pour contribuer à leur développement, le philosophe ne manquera pas d'observer leur progrès, car il ne saurait demeurer indifférent à la question de savoir jusqu'où et selon quelles modalités pourra se constituer ou se développer une science des phénomènes de l'esprit orientée par le paradigme de la machine et l'idée de calcul, et reposant sur la thèse de Church-Turing.

Alors que la première interrogation porte clairement sur l'esprit et la pensée humaine, la seconde concerne, au moins en apparence, les machines et leurs capacités. Mais la question de savoir si une machine peut penser ne signifie rien si elle ne fait pas référence à l'expérience humaine de la pensée, et il faut donc comprendre : une machine pourrait-elle faire ce que fait un être humain lorsqu'il pense ? Ici encore, la question vise moins les machines que la nature de la pensée ou les conditions de possibilité d'une théorie de l'esprit. Précisons : à quelles fins les machines sont-elles prises en considération par la philosophie de l'esprit, et de quelles machines s'agit-il ? Il n'est évidemment question ici que de celles qu'étudient la logique et l'informatique ; mais elles peuvent être considérées de points de vue différents, et l'usage philosophique qui en est fait est variable. En fonction de la perspective adoptée sur la question de la pensée des machines, on distinguera trois catégories : les machines réelles, possibles et fictives.

Les machines réelles

L'intelligence artificielle ne se réduit pas à un ensemble de thèses philosophiques sur la possibilité de concevoir des machines pensantes ou sur la nature de l'esprit. Elle se présente d'abord comme une partie de l'informatique, réalisant physiquement des machines capables d'effectuer des tâches qui requéraient auparavant l'intelligence humaine. Relèvent de son domaine de recherche toutes les capacités et les comportements par lesquels un être humain manifeste une forme d'intelligence ; elle se propose de programmer ces capacités et comportements afin de les mettre à la portée d'une machine. C'est ici qu'apparaît l'intérêt philosophique de la discipline, dans la mesure où elle prétend réduire l'intelligence humaine à un ensemble de calculs programmés, et l'implémenter ensuite sur des machines réelles. Au cours des années soixante et jusqu'à la fin des années soixante-dix, lorsque Dreyfus s'interroge sur les fondements philosophiques de l'intelligence artificielle, il mène une enquête systématique sur les programmes, les robots et les diverses machines effectivement réalisées ou en projet ; sa « critique de la raison artificielle »[1] ne repose donc pas seulement sur l'examen de quelques présupposés relatifs à l'esprit et son fonctionnement mais également sur une étude minutieuse des machines réelles. Pourtant, le but de Dreyfus n'est pas tant de caractériser l'ensemble des capacités accessibles à une machine (bien qu'il prenne soin d'apporter à cette question un début de réponse)[2], que de soumettre à une sévère analyse critique l'idée que la programmation adéquate d'une machine suffirait à la rendre intelligente, et que la pensée humaine procède à la manière d'un calculateur numérique. Sa critique vise la théorie de l'esprit qui est présupposée par le projet de l'intelligence artificielle et que Dreyfus juge insoutenable. Du point de vue des machines réelles, la question de

1. Hubert Dreyfus, *op. cit.*, p. 18 (trad. franç.).
2. *Ibid.*, p. 375.

la pensée des machines prend donc la forme suivante : l'esprit est-il de nature telle que nous puissions espérer construire des machines capables de manifester un comportement intelligent et capables de penser comme un être humain ? Selon Dreyfus, les difficultés empiriques rencontrées par l'intelligence artificielle sont significatives d'une théorie de l'esprit fondamentalement erronée qui est pourtant présupposée par les projets les plus audacieux de l'intelligence artificielle[1].

Lorsqu'elle se place du point de vue des machines réelles, la réflexion philosophique sur la pensée des machines porte surtout sur l'intelligence artificielle et vise essentiellement le problème de la nature de l'esprit humain.

Les machines possibles

Si les performances des machines intelligentes effectivement réalisées ont incontestablement joué un rôle important pour le développement des sciences cognitives, les difficultés rencontrées par les projets d'intelligence artificielle tels qu'ils étaient initialement conçus ont aussi largement contribué à orienter les sciences de la cognition vers des explications qui ne s'appuient pas directement sur le modèle de la programmation des ordinateurs. Ce sont bien les machines informatiques et l'intelligence artificielle qui ont introduit en psychologie l'idée d'un point de vue cognitif intermédiaire entre le niveau biologique des neurosciences et le niveau purement intentionnel de certaines formes d'explication psychologique, et les développements actuels des sciences cognitives semblent indiquer que cette approche des phénomènes de l'esprit peut être éclairante et féconde. Mais ils montrent également que ces sciences peuvent s'affranchir d'un modèle explicatif inspiré de la programmation des machines, et que l'intelligence artificielle n'est que l'une des com-

1. Pour un examen plus précis des arguments de Dreyfus, cf. notre étude sur L'intelligence artificielle selon Hubert Dreyfus, in *Machine et pensée,* chap. 6.

pensant et une machine imaginaire, aussi perfectionnée qu'on voudra : à supposer que la simulation soit parfaite, pourrait-on affirmer que la machine pense ? Ou plutôt, si l'on souhaite insister sur le caractère énigmatique de la question elle-même : cette machine pourrait-elle penser *réellement* ?

Ici encore, il est moins question de déterminer quelles sont les capacités des machines réelles (celles dont il s'agit sont imaginaires) que de s'interroger sur l'essence de la pensée ou sur la nature de l'esprit.

Considérons cette troisième version du problème de la pensée des machines : que conclure de l'hypothèse fictive d'une simulation parfaite ? L'énigme vient de ce que l'on a l'impression de s'interroger sur un *quelque chose* dont on se demande si la machine le possède, tout en sachant qu'il ne peut appartenir qu'à une intériorité tout à fait inaccessible, et que la question, *ex hypothesi*, ne pourra pas être résolue de l'extérieur grâce à un critère de la pensée reposant sur le comportement. Elle évoque évidemment le problème métaphysique de l'âme ou de la pensée des autres : Autrui pense-t-il ou lui attribué-je une pensée qu'il n'a pas réellement ? Comment puis-je m'assurer qu'il pense vraiment ? Quelle raison pourrait me le prouver ? Car « que vois-je de cette fenêtre, sinon des chapeaux et des manteaux, qui peuvent couvrir des spectres ou des hommes feints qui ne se remuent que par ressorts ? »[1]. Or cette similitude laisse penser que l'on ne doit pas attendre une solution de l'énigme qui ne soit ni métaphysique (comme par exemple, au sujet de la pensée d'autrui, l'idée d'un dieu vérace me donnant une certitude morale que les autres pensent) ni dogmatique (par exemple les thèses de l'intelligence artificielle forte selon lesquelles l'esprit est un programme dont le corps n'est qu'une implémentation parmi d'autres possibles)[2]. Le caractère énigmatique du problème suggère aussi qu'il pourrait être non pas résolu

1. Descartes, *Méditations métaphysiques,* Méditation seconde.
2. A cet égard, l'expérience de pensée de la chambre chinoise ne fait pas exception (pour les références, cf. ci-dessus, chap. VII, n. 1, p. 137 ; cf. aussi notre analyse de l'argumentation de Searle, La chambre chinoise, in *Machine et pensée,* chap. 4).

mais dissolu, par un examen de sa signification. Dans « Le Fantôme dans la machine »[1], à partir de plusieurs textes de Wittgenstein, Bouveresse montre que la question de la pensée des machines pourrait bien être avant tout un problème *grammatical.* Afin de réfuter définitivement le scepticisme relatif à la pensée d'autrui, plusieurs philosophes de l'époque classique ont recherché des *raisons* susceptibles de prouver que « les autres » ne sont pas des spectres ou des hommes feints qui se remuent par des ressorts, et que leur pensée n'est pas l'objet d'une croyance illusoire. Cependant, si l'on rapproche les deux difficultés (la pensée d'autrui et celle des machines), on ne peut qu'être frappé par la facilité avec laquelle nous accordons spontanément la pensée à autrui, en l'absence de toute preuve capable de justifier cette attitude, par opposition aux réticences que nous éprouverions à reconnaître qu'une machine pense, même dans le cas d'une simulation parfaite du comportement humain :

> (...) nous éprouvons une sorte de répugnance irrationnelle à dire que les machines pensent et (...) nous ne pouvons à proprement parler la fonder sur aucune impossibilité du type de celles qui peuvent normalement être remises en question ou même supprimées par des productions théoriques ou technologiques ultérieures. Il nous semble que, quoi que les machines puissent faire et quoi que la science puisse dire *légitimement* (sur la base de théories qu'il ne nous viendrait pas à l'esprit de contester) un jour, nous ne changerons pas d'avis sur ce point crucial[2].

Comme s'il ne fallait pas s'attendre à résoudre l'énigme par une argumentation fondée sur des raisons et visant à prouver qu'autrui pense *réellement.* Comme s'il n'y avait pas, relativement à cette question comme à celle de la pensée des machines, de place pour un *savoir.*

> Lorsque nous nous demandons si les autres personnes sont réellement des personnes, c'est-à-dire si elles ont des pensées, des sentiments, etc., il semble que nous envisagions clairement et distinctement qu'elles puissent être autre chose. Mais quoi ? Ce que veut dire Wittgenstein, c'est qu'en réalité nous n'envisageons rien de clair et de précis, qu'il n'y a pas de « par opposition à... »[3].

1. Jacques Bouveresse, Le Fantôme dans la machine, in *La parole malheureuse. De l'alchimie linguistique à la grammaire philosophique,* Paris, Éditions de Minuit, 1971.
2. *Ibid.,* p. 414.
3. *Ibid.,* p. 445.

Selon l'analyse de Bouveresse, l'origine de la prévention par laquelle nous reconnaissons sans hésitation et sans preuve la pensée d'autrui, et qui *a contrario* nous empêche d'acquiescer à l'idée qu'on pourrait un jour nous présenter une machine capable de penser, doit être recherchée dans l'institution de notre langage. C'est son usage et son apprentissage qui n'accordent pas de signification claire à la représentation d'une pensée des machines.

> Nous devons donc supposer que les concepts d'âme, de pensée, d'émotion, de douleur, etc., ont été acquis dans l'apprentissage d'un langage commun et qu'ils ont été rattachés d'une manière essentielle à l'homme et à ce qui a la forme humaine[1].

C'est la question elle-même qui n'appartient pas à notre langage, à son usage, et qui, par conséquent, *ne se pose pas*[2]. S'il en est bien ainsi, la difficulté ne saurait être résolue par l'acquisition d'un savoir sur les machines ou sur l'esprit et la pensée, mais elle pourrait être dissolue par un examen de la grammaire des mots « machine » et « pensée ».

Si le langage lui-même, à travers son usage et son apprentissage, associe de manière essentielle la pensée à ce qui est humain, on comprend aisément la répugnance et la difficulté que nous pouvons éprouver à concevoir quelque chose comme une pensée mécanique, et à nous représenter la pensée d'une machine sans viser indirectement des problèmes relatifs à l'esprit et la pensée humaine. De fait, les différentes interprétations données à la question de la pensée des machines (qu'elles soient réelles, possibles ou fictives) sont toutes orientées vers l'esprit, la pensée humaine ou la psychologie.

Ce qui semble inconcevable – et qui est probablement rendu inconcevable par l'institution de notre langage –, c'est l'idée que nous pourrions considérer un jour une machine réelle comme étant capable de penser, et que notre comportement à son égard puisse devenir semblable à celui que nous adoptons à l'égard des êtres que nous avons appris à reconnaître comme des êtres pensants. Peut-être

1. *Ibid.*, p. 444.
2. *Ibid.*, p. 424.

cela explique-t-il la réticence que nous éprouvons à considérer les machines comme étant douées d'une forme de pensée, fût-elle mécanique, singulière, différente de la pensée humaine. Peut-être cela explique-t-il aussi que la réflexion philosophique sur l'informatique soit le plus souvent tournée vers l'intelligence artificielle, comprise comme le projet d'une programmation des capacités de l'esprit. Une irrésistible inclination nous porte à comprendre l'expression « pensée mécanique » comme une métaphore, plus propre, semble-t-il, à engendrer la confusion qu'à éclaircir une réflexion philosophique sur les machines. Car à la fois par l'expérience que nous en avons et par ce que nous avons appris à en dire, la pensée est essentiellement liée à ce qui est humain. Ce n'est donc pas seulement en énonçant une définition que l'on donnera un sens satisfaisant à l'expression « pensée mécanique », mais également en indiquant la possibilité de porter sur les machines informatiques quelque chose comme un autre regard. Pour les raisons qu'on a dites, il serait certainement absurde de les considérer comme des choses pensantes ; pourtant, leurs capacités présentes et futures empiètent et empiéteront suffisamment sur ce qui était naguère l'apanage de l'esprit humain pour qu'elles ne puissent plus être considérées comme entièrement étrangères à ce qui relève de la pensée. Le problème est de concevoir l'existence de ce moyen terme qui n'appartient pas à la fiction, entre la réalité purement physique des machines et la pensée proprement humaine.

Pensée mécanique et logique

La pensée mécanique réelle : le point de vue empirique

A cause des préjugés de notre langage, et aussi bien par la définition même de son objet, la philosophie de l'esprit évite et contourne soigneusement l'idée qu'en un sens (qui reste encore à préciser), les machines réelles puissent penser *hic et nunc*. L'existence

d'une forme de pensée mécanique susceptible d'être considérée en elle-même et pour elle-même mérite pourtant la définition d'une problématique singulière, différente de celle qui concerne la nature de l'esprit et des états mentaux.

Il est vrai qu'en parlant de *pensée* mécanique on se réfère implicitement, une nouvelle fois, à la pensée humaine. Mais il est possible de considérer les capacités des machines relativement au langage, au raisonnement, à la connaissance ou à l'action intelligente sans s'interroger sur la nature de l'esprit, sur son fonctionnement, sur les états mentaux, la conscience, ou l'essence de la pensée. De ce point de vue, la question principale est la suivante : de quelles performances les machines logico-informatiques sont-elles capables ? Considérons premièrement une approche empirique du problème. Ces machines sont des dispositifs abstraits ou physiquement réalisés qui permettent de traiter l'information, c'est-à-dire de l'enregistrer, de la conserver en mémoire, de la transformer selon des règles parfaitement déterminées, de la transmettre ou de la communiquer. Toutes ces opérations sont effectuées par la machine sur une représentation physique des informations, qui peuvent elles-mêmes représenter, par formalisation, codage et modélisation, d'autres entités comme des nombres, des textes, des sons, des paroles, des images, des connaissances, des raisonnements, des langages, des règles, des jeux, des objets ou des phénomènes physiques, l'évolution d'un processus naturel ou artificiel, la description d'une situation, la structure d'une organisation, diverses aptitudes, des comportements, ou encore une stratégie. Par l'intermédiaire du codage, les machines informatiques peuvent effectuer sur toutes ces entités un grand nombre d'opérations : calculs sur des nombres, traitement de texte, articulation de paroles, mise en mémoire des informations, classifications, recherches sélectives et transmission de connaissances, démonstration automatique, diagnostic médical, modélisation de phénomènes physiques, choix rationnel par calcul d'optimisation, commande d'autres machines, etc. Par le traitement d'informations codées, les machines informatiques sont ainsi capables de réaliser des performances qui étaient naguère l'apanage de

l'intelligence ou de la pensée, humaine ou animale, dans des domaines qui, aujourd'hui, ne leur sont plus inaccessibles : les connaissances, le langage, les actions, le raisonnement... On appellera *pensée mécanique* l'ensemble des capacités que manifestent ou pourraient manifester les machines et qui reposent sur des possibilités de calcul et de traitement de l'information. Ici, l'usage du mot « pensée » se justifie uniquement par le fait que les machines peuvent réaliser certaines performances dont seuls les êtres humains pouvaient donner l'exemple jusqu'à une période récente, et qui requièrent chez eux ce qu'ils reconnaissent eux-mêmes comme l'exercice de la pensée. Il n'implique aucune hypothèse relative à ce que fait un homme lorsqu'il pense, ni au fonctionnement de l'esprit. Par ailleurs, il n'est pas limité aux capacités qui, lorsqu'elles sont celles d'un être humain, exigent de l'intelligence. En particulier, cette définition de la pensée mécanique ne retient pas la distinction usuelle en informatique entre les méthodes algorithmiques (utilisées lorsque l'on connaît une procédure effective permettant de résoudre le problème que l'on se pose) et les méthodes heuristiques propres à l'intelligence artificielle (utilisées comme des moyens de recherche dans l'espace d'un problème lorsque aucune solution algorithmique n'est connue)[1]. Toutes les techniques de traitement de l'information par une machine relèvent de cette forme de pensée bien particulière, non humaine[2], qui est celle des ordinateurs.

D'un point de vue empirique, la notion de pensée mécanique se justifie par l'existence d'une similitude entre certaines facultés de l'esprit humain et certaines performances des machines informatiques. Pourtant, elle permet de concevoir une problématique différente de celle qui oriente habituellement la question de la pensée des machines. On ne demande pas si le calcul et le traitement de l'information sont susceptibles de fonder une théorie de l'esprit ou

1. On s'efforce alors de simuler l'exercice de l'intelligence humaine à l'aide de règles heuristiques suivies par la machine.
2. Cette expression ne préjuge en rien du rapport de la pensée mécanique à la pensée humaine. Sur ce point, on s'abstient seulement, au moins provisoirement, de tout jugement. On se propose de poser un *autre* problème.

de nous éclairer sur la pensée humaine ; on s'interroge seulement sur les capacités des machines considérées en elles-mêmes. On ne cherche pas à prouver que les phénomènes psychiques peuvent être expliqués par une analyse fonctionnelle ou un programme informatique ; on s'efforce de mesurer quelles sont les opérations qu'il est possible de mécaniser et quelles sont celles qui résistent à toute tentative de formalisation et de programmation. On ne postule aucun recouvrement des capacités de l'esprit par celles des machines, aucune identité de nature entre la pensée mécanique et la pensée d'un être humain ; on définit seulement l'idée d'une pensée non humaine afin d'en comprendre les limites et les propriétés. La notion de pensée mécanique n'est donc pas mise au service d'une philosophie de l'esprit ; elle doit permettre de constituer une philosophie des machines. A l'intelligence artificielle orientée par le projet d'une programmation des capacités de l'esprit humain s'oppose une pensée mécanique définie, du point de vue empirique, par un ensemble de performances relatives au raisonnement, au langage, à l'action, aux connaissances, et fondées sur des possibilités de calcul et de traitement de l'information.

Il est vrai qu'une grande partie des travaux menés en intelligence artificielle relève également d'une étude de la pensée mécanique, tout comme certaines analyses de Dreyfus, dans la mesure où elles s'efforcent de caractériser, afin de les mesurer, les capacités et les incapacités des machines. L'opposition qui vient d'être esquissée doit donc être comprise comme une différence d'orientation et de visée. De fait, le plus souvent, l'intérêt philosophique de l'intelligence artificielle est compris par référence à la possibilité d'élaborer une théorie de l'esprit et de concevoir les principes et les méthodes d'une forme de psychologie. *A contrario,* l'idée de pensée mécanique circonscrit l'objet d'une étude et d'une philosophie des machines informatiques dont le problème principal n'est pas celui de la nature de l'esprit, des états mentaux, ou de la pensée humaine. Encore faudrait-il préciser davantage, dès lors que la connaissance de l'esprit n'est plus directement visée, quels seraient à la fois l'unité de la pensée mécanique ainsi comprise et la nature des pro-

blèmes philosophiques qu'elle soulève. Or le point de vue empirique qui vient d'être considéré ne permet pas, à lui seul, d'apporter ces précisions. Il doit être complété par la définition d'un contexte théorique de référence dans lequel puisse trouver place une analyse de la pensée mécanique.

La pensée mécanique possible : le point de vue logique

Le concept de pensée mécanique ne peut être éclairant et fécond que s'il s'inscrit dans un cadre formel, et s'il est associé à des moyens théoriques rendant possible son étude. Or la logique, en tant qu'elle prend pour objet d'analyse le calcul, les algorithmes et les machines, prouve des énoncés relatifs à ce qui est mécaniquement démontrable et se présente, plus généralement, comme une étude systématique et abstraite des capacités des machines. Elle constitue à ce titre une discipline formelle de la pensée mécanique, comme le montre l'interprétation informatique des principaux résultats et énoncés logiques dont il a été question précédemment.

Théorème de complétude : toute démonstration mathématique peut être formalisée en sorte que la validité de la preuve puisse être vérifiée par une machine[1].

Thèse de Church-Turing : les fonctions calculables sont exactement celles qu'il est possible de calculer par une machine ; en conséquence, « un système formel peut simplement être défini comme une procédure mécanique quelconque pour la production de formules, appelées formules démontrables »[2].

Premier théorème d'incomplétude : étant donné un système formel consistant et permettant d'exprimer une partie élémentaire de l'arithmétique, aucune machine ne permet, par une production systématique de tous ses théorèmes, de répondre à toutes les ques-

1. Cf. ci-dessus, chap. I, n. 1, p. 27.
2. Gödel, cf. ci-dessus, Introduction, p. 3.

tions qu'il est possible de formuler dans le système ; certains énoncés sont mécaniquement indécidables.

Second théorème d'incomplétude : étant donné un système formel satisfaisant les mêmes conditions que le théorème précédent, la consistance du système est mécaniquement indécidable.

Théorème d'indécidabilité : aucune machine ne permet de déterminer de manière systématique pour une formule quelconque du calcul des prédicats si elle est satisfaisable, ou si elle est valide.

Correspondance de Curry-Howard : il existe des systèmes logiques de lambda-calcul typé dans lesquels, en démontrant une formule A, on construit un algorithme (et donc une machine) dont la fonction est précisément ce qui est spécifié par la formule A[1].

Ces énoncés montrent en quel sens la logique peut être considérée comme un cadre théorique général pour l'analyse des capacités des machines ; elle détermine *a priori* quels sont les problèmes qu'il est possible ou impossible de résoudre mécaniquement, et si tel ou tel type de question est à la portée ou hors de portée d'une pensée mécanique.

Les résultats obtenus en théorie de la calculabilité dans les années trente permirent d'établir clairement que certains problèmes tombent en dehors des capacités des machines. Mais comme on a déjà pu le remarquer précédemment, si les logiciens élaborèrent à cette époque différentes analyses de la notion d'effectivité, les principales questions qui les occupaient étaient relatives à la calculabilité et ne concernaient pas la faisabilité des algorithmes. Pour la caractérisation de l'effectivité comme pour la formalisation des raisonnements, on cherchait à concevoir des formalismes qui requissent des moyens aussi primitifs et élémentaires que possibles, dans le but de simplifier la démonstration des théorèmes métamathématiques[2]. Or l'extrême simplicité des outils syntaxiques utilisés pour caractériser les procédures effectives (les machines de Turing, les

1. Cf. l'exemple donné ci-dessus (chap. VIII, p. 160-161 et 170) d'un algorithme pour le tri d'une liste de nombres.
2. Afin de montrer, aussi, que nos méthodes de raisonnement et de calcul peuvent être réduites à la combinaison de procédés très rudimentaires.

lambda-termes) ou pour formaliser les preuves (les règles de la déduction naturelle et du calcul des séquents, les règles d'inférence d'un système à la Hilbert) avait pour contrepartie l'extrême longueur des suites d'opérations qu'ils engendraient ou des objets formels qu'ils permettaient de construire : les étapes d'un calcul par une machine de Turing, les réductions successives d'un lambda-terme, une démonstration formalisée, la normalisation d'une preuve ou l'élimination des coupures sont des exemples d'objets ou de suites d'opérations qui atteignent souvent, même dans les cas concrets les plus courants, une longueur si grande qu'elle rend impossible leur manipulation par un être humain. Cette impossibilité ne pose aucun problème tant que les objets ou les opérations formelles restent des outils ou des moyens analytiques purement théoriques et sans rapport avec les machines réelles. C'est surtout à partir du moment où l'on conçoit l'idée de confier à un ordinateur des calculs qui requièrent un nombre très élevé d'opérations élémentaires que se présente le problème d'une mesure de la complexité des algorithmes. Alors que les débuts de la théorie de la calculabilité datent d'une période qui précède la construction des premiers ordinateurs, la théorie de la complexité n'apparaît qu'après les premiers développements de l'informatique[1]. Le problème se pose notamment au sujet des fonctions récursives connues. Par exemple, comment comparer, de ce point de vue, l'addition de deux entiers, le problème du voyageur de commerce[2] et le calcul de

1. Sur les débuts de la théorie de la complexité, on pourra se référer à Mosconi (*op. cit.*, chap. XII) qui écrit : « (...) la théorie de la complexité ne commence d'exister que vers 1960 (au mieux), surtout si on veut la considérer comme une branche de l'informatique théorique. La théorie classique de la calculabilité avait bien suscité une certaine classification des fonctions, mais c'est très tardivement que celle-ci a été conçue en termes de machines. La particularité de la théorie de la complexité est (...) d'être à la fois un chapitre de la théorie des fonctions récursives et un prolégomène mathématique à la théorie de la programmation. Historiquement et même théoriquement elle ne se conçoit guère sans l'existence et le maniement de machines réelles. Elle n'apparaît d'ailleurs qu'au moment où commencent à être mieux comprises les relations entre celles-ci et les machines de Turing (...) », p. 553-554.
2. Un voyageur de commerce doit passer dans n villes déterminées. On connaît la distance qui sépare deux villes quelconques. Dans quel ordre doit-il les visiter pour que son trajet soit le plus court possible ?

la fonction d'Ackermann[1] ? En cherchant à résoudre ce type de questions par des moyens logiques, la théorie de la calculabilité se prolonge en une théorie de la complexité qui évalue la faisabilité des algorithmes. De cette manière, on définit formellement une distinction entre les procédures effectives qui sont à la portée d'une machine réelle, et celles dont la complexité est rédhibitoire. On voit donc que si la théorie de la calculabilité élaborée dans les années trente constitue le cadre le plus général d'une analyse de la pensée mécanique, la théorie de la complexité et l'idée d'une faisabilité des algorithmes permettent d'affiner davantage l'étude théorique de cette forme de pensée.

La complexité des algorithmes et leur faisabilité sont des questions informatiques relatives aux machines réelles qui relèvent pourtant d'une analyse logique. Elles se posent à l'intérieur du domaine des problèmes susceptibles d'être résolus par une machine. La logique considère également, à l'extérieur de ce domaine, une hiérarchie de problèmes ordonnés selon leur degré de difficulté, qui peut dépasser de très loin celle qu'une machine est susceptible de surmonter. La forme générale de ces problèmes est la suivante : quels sont les n-uplets d'entiers qui satisfont la formule arithmétique $A[x_1, \ldots, x_n]$? Par exemple, pour le problème de la décision, quels sont les entiers qui satisfont la formule $\mathbf{Th}[x]$ [2] ? La difficulté du problème dépend directement de la complexité logique de la formule $A[x_1, \ldots, x_n]$, mesurée par l'alternance des quantificateurs[3]. Or seuls les problèmes situés aux

1. La fonction d'Ackermann, $A(x, y)$, est définie par les égalités suivantes :
$$A(1, 0) = 2;$$
$$A(0, y) = 1;$$
$$A(x, 0) = x + 2 \text{ si } x \geq 2;$$
$$A(x + 1, y + 1) = A(A(x, y + 1), y).$$
Cette fonction récursive croît plus rapidement que n'importe quelle fonction primitive récursive. Elle est étudiée par Ackermann en 1928 dans Zum Hilbertschen Aufbau der reellen Zahlen, *Mathematische Annalen*, 99, trad. angl., *in* Van Heijenoort, *From Frege to Gödel*, et elle a joué un rôle important pour l'étude de la classification des fonctions.
2. En reprenant les notations du chapitre VI ; cf. p. 116, n. 2.
3. Étant donné les formules A_0, $A_1[x]$, $A_2[x, y]$, $A_3[x, y, z]$, etc., sans quantificateur, les formules suivantes ont une complexité logique croissante : A_0, $\exists x A_1$, $\forall y \exists x A_2$, $\exists z \forall y \exists x A_3$, etc. Pour une définition plus précise et plus complète de la hiérarchie arithmétique (hiérarchie de Kleene), cf. par exemple Lalement, *op. cit.*, chap. VII, § 4 et 5.

niveaux les plus bas de la hiérarchie peuvent être résolus ou partiellement résolus par une machine : si les formules $A[x]$ et $B[x, y]$ d'une arithmétique formelle sont sans quantificateur, des problèmes de la forme « n satisfait $A[x]$ » et « n satisfait $\exists x B[x, y]$ » sont, respectivement, décidables et semi-décidables[1]. Au-delà, les problèmes posés à partir de formules plus complexes (deux quantificateurs alternés ou plus) sont tout à fait hors de la portée des machines. Ces précisions permettent de comprendre en quel sens la logique apparaît comme un cadre théorique général qui offre les moyens analytiques et formels d'une étude de la pensée mécanique, de ses capacités et de ses limites.

C'est cependant l'isomorphisme de Curry-Howard qui montre de la manière la plus convaincante pourquoi la logique se présente comme une étude formelle et *a priori* de la pensée mécanique. A travers la correspondance qu'il établit entre les preuves et les programmes, les formules et les types de données, l'opération de normalisation des preuves et l'exécution des calculs, la théorie de la démonstration elle-même prend la forme d'une théorie des programmes, et donc des machines abstraites. Les inférences sont associées aux opérations d'une machine. Par exemple, à un niveau très élémentaire, le *Modus Ponens* correspond à l'application d'une procédure $\lambda x.t$ (opérateur qui transforme chaque x en un objet t qui – en général – dépend de x) à un argument u.

$$\frac{A \to B \quad A}{B} \qquad \frac{\lambda x.t \quad u}{(\lambda x.t)\, u}$$

L'isomorphisme de Curry-Howard généralise cette mise en correspondance des déductions élémentaires et des opérations d'une machine à l'ensemble des règles d'inférences d'un formalisme comme la déduction naturelle. Il permet de voir les preuves formalisées comme des programmes ou des machines, et montre parfaite-

1. On dit (abusivement) que le problème « n satisfait $A[x]$ » est décidable si l'ensemble des entiers n qui satisfont $A[x]$ est décidable. Dans ce cas, il existe une machine de Turing permettant de déterminer, pour tout entier n, si n satisfait ou ne satisfait pas $A[x]$. Le problème « n satisfait $\exists x B[x, y]$ » est semi-décidable parce qu'une machine de Turing ne peut le résoudre que partiellement : elle donne une réponse seulement dans le cas où n satisfait la formule $\exists x B[x, y]$.

ment en quel sens la logique peut être conçue comme un cadre formel pour l'étude de la pensée mécanique[1].

Deux points de vue complémentaires permettent d'appréhender la pensée mécanique. Au niveau empirique, elle se présente comme l'ensemble des capacités que donnent aux machines réelles les techniques de traitement de l'information. Sur le plan logique, elle fait l'objet d'une analyse théorique qui en mesure les possibilités à partir d'une étude des algorithmes et des machines abstraites. Ainsi caractérisée, la notion de pensée mécanique a des implications qui concernent à la fois l'idée d'une philosophie des machines et le problème de l'objet de la logique.

Machines et logique

Pensée mécanique et philosophie des machines

Lorsque l'on définit l'intérêt philosophique des machines informatiques par la possibilité qu'elles offrent ou qu'elles pourraient offrir d'élaborer une théorie de l'esprit et de constituer une forme de psychologie scientifique, on s'intéresse moins aux problèmes singuliers soulevés par les développements actuels de l'informatique qu'aux solutions apportées par l'intelligence artificielle à des problèmes anciens : Comment l'esprit fonctionne-t-il ? Quelle est la nature des états mentaux ?, etc. Seuls sont jugés importants les aspects de l'informatique susceptibles d'éclairer ces questions propres à la philosophie de l'esprit. A cet égard, c'est surtout la perspective d'une naturalisation de la psychologie qui éveille la réflexion et suscite l'espoir. Pourtant, lorsque le philosophe examine la question de savoir si une machine peut penser, cette possibilité est toujours proje-

1. Du point de vue de l'isomorphisme de Curry-Howard, il existe aussi un lien entre la complexité logique des formules démontrables dans tel système considéré, et la complexité algorithmique de la normalisation des preuves.

tée dans un avenir lointain ou conçue à travers une expérience de pensée qui relève de la fiction. Comme si les machines n'étaient pas dès aujourd'hui suffisamment capables de manifester une forme de pensée singulière, non humaine, mécanique, qui mériterait d'être considérée en elle-même et comme telle par le philosophe. La notion de pensée mécanique ne prétend pas apporter une solution nouvelle à des problèmes anciens comme celui de la nature de l'esprit ; mais elle pourrait définir les conditions d'une réflexion philosophique sur les questions nouvelles que posent les machines informatiques.

Dans son étude sur *Intelligence artificielle. Mythes et limites*, Dreyfus se donnait pour tâche une « critique de la raison artificielle » élaborée à partir d'une enquête empirique sur les travaux et les projets de l'intelligence artificielle. Le point de vue de la pensée mécanique appelle un double déplacement de cette problématique, et par conséquent, une orientation différente de la réflexion. Premièrement, l'opposition entre les méthodes de l'intelligence artificielle d'une part et celles de l'informatique classique d'autre part doit être dépassée parce qu'il n'est plus question de savoir si telle théorie de l'esprit est recevable ou si elle ne l'est pas, mais de considérer les capacités des machines pour elles-mêmes et de manière globale. Deuxièmement, l'examen empirique de la pensée mécanique doit être situé dans un cadre théorique plus général, celui de l'analyse logique des machines possibles. Ce double déplacement caractérise une problématique sensiblement différente de celle que définissait Dreyfus : une critique de la pensée mécanique.

Définir les conditions d'une réflexion philosophique sur les machines qui ne soit pas orientée par la recherche d'une théorie de l'esprit ne signifie pas renoncer à toute comparaison avec la pensée humaine. Une analyse de la pensée mécanique pourrait au contraire mettre en lumière les raisons pour lesquelles certaines formes de comportement intelligent résistent aux efforts déployés pour tenter de les programmer. Car la pensée mécanique ne se présente pas comme la forme à laquelle devraient pouvoir se réduire les facultés de l'esprit et la variété des comportements intelligents, mais comme une catégorie bien particulière d'opérations et de capacités, permettant de recon-

naître et de comprendre ce qui fait la différence et la singularité des diverses formes de la pensée humaine.

L'informatisation du diagnostic médical montrera sur un exemple concret comment la pensée mécanique peut mettre en lumière certaines caractéristiques propres aux raisonnements humains. Selon Anne Fagot[1], la logique du raisonnement médical est telle que dans nombre de cas, et pour nombre de questions, la machine pourrait, aujourd'hui ou dans l'avenir, remplacer avantageusement le médecin, notamment pour l'établissement de certains diagnostics ; mais d'autre part, les problèmes de formalisation des connaissances et de mécanisation du raisonnement médical ne peuvent absolument pas être traités *in abstracto*, parce qu'ils peuvent dépendre d'un grand nombre de paramètres d'ordre culturel, géographique, moral, historique, etc. Par conséquent, la décision de confier une tâche à la machine dépend non seulement de ce qu'il est *possible*, mais également de ce qu'il est *souhaitable* ou *légitime* de mécaniser[2]. Anne Fagot donne l'exemple d'un système informatique de diagnostic utilisé à Leeds. Alors que dans cette ville ses performances étaient tout à fait remarquables, il ne donnait, dans d'autres villes, que de médiocres résultats. C'est que le raisonnement mécanique du système dépendait de données médicales propres à une localité particulière :

> Le programme de Leeds, pour calculer la probabilité (par exemple) d'une appendicite, donnés les symptômes présentés par le patient (ex. fièvre, défense abdominale, rougeur du visage, etc.) se fondait sur la fréquence relative du diagnostic d'appendicite à Leeds, et la fréquence de la fièvre, de la défense abdominale, de la rougeur du visage, etc., dans l'appendicite. Les anglais avaient remarqué que, transporté à Edinburgh ou Copenhague, le programme de Leeds se comportait d'abord assez médiocrement, et ne retrouvait peu à peu son niveau élevé de performance qu'en intégrant des données locales, en « apprenant » la fréquence des divers diagnostics dans la nouvelle population[3].

1. Le concept de maladie sous-jacent aux tentatives d'informatisation du diagnostic médical, *Hist. Phil. Life Sci.*, 10, suppl., 1988, p. 89-110.
2. Le problème peut par exemple être posé en termes de *risques* : dans quels cas sommes-nous fondés à accorder une plus grande confiance à la machine qu'à l'être humain ? le raisonnement médical mécanisé n'est-il pas souvent plus sûr que le raisonnement d'un médecin ?
3. Anne Fagot, *op. cit.*, p. 100.

Or, comme le montre la suite du texte, non seulement la fréquence d'un diagnostic varie d'une ville à l'autre, mais encore les signes cliniques n'ont pas la même valeur prédictive :

> Une appendicite ne se traduit pas par les mêmes symptômes à Paris qu'à Londres, la séméiologie des maladies varie selon les populations. (...) Seule est transposable de Leeds à Colombes la méthode générale de raisonnement (bayesienne), qui tient en une équation. La base de données, qui fait l'efficacité diagnostique de cette méthode, est au mieux une base de données locales, constamment remise à jour[1].

Cet exemple met en lumière l'intérêt philosophique d'une étude de la pensée mécanique, car dans le cas précis examiné par Anne Fagot, l'articulation entre une structure formelle de raisonnement et une situation singulière et concrète fait apparaître certains aspects du concept de maladie ainsi que l'importance du contexte géographique et culturel dans le diagnostic médical ; l'informatisation de cette forme de raisonnement soulève aussi plusieurs problèmes de risque, de choix rationnel, d'éthique. L'examen philosophique de la pensée mécanique ne se limite donc pas à l'analyse des possibilités théoriques et techniques pour la mise en œuvre des procédures effectives. Une connaissance précise des capacités et incapacités des machines permet également de souligner certaines singularités de la pensée et du raisonnement humains, et d'éclairer les questions de droit, d'éthique, de choix rationnel, de liberté que posent l'usage des machines informatiques et le développement d'une forme de pensée mécanique.

La pensée mécanique et l'objet de la logique

Aussi longtemps qu'on la considère comme un objet second et marginal de l'analyse logique, ou encore comme l'un de ses domaines d'application, la machine est sans rapport avec le problème général de l'unité et de l'objet de cette discipline. Mais les

1. *Ibid.*, p. 100-101.

chapitres qui précèdent suffisent à établir que la machine a progressivement acquis une place centrale dans la logique contemporaine. Ils donnent ainsi tout son sens à la difficulté soulevée ci-devant : Comment comprendre qu'une seule et même discipline traite à la fois de la vérité, du raisonnement, du langage et de la connaissance d'une part, du calcul, des algorithmes et des machines d'autre part ? Comment comprendre l'existence de liens étroits entre l'informatique et la logique ? La question est posée d'un point de vue épistémologique particulier qui n'est pas normatif mais descriptif et historique. On ne demande pas ce qu'est la logique *de jure* ; on s'efforce de comprendre pourquoi elle prend aujourd'hui la forme d'une science des algorithmes et des machines, étroitement liée à l'informatique, alors qu'elle est généralement considérée comme une analyse des conditions formelles de la pensée vraie, du raisonnement correct et du discours rationnel, ou encore comme une étude des langues formulaires et du fondement des mathématiques.

Si l'on accepte de définir très généralement la logique comme une recherche des conditions formelles de la connaissance, de la pensée vraie et de l'inférence valide en tant qu'elles dépendent d'un langage permettant de les exprimer, on peut voir dans l'entreprise de formalisation du raisonnement inaugurée par Frege et redéfinie dans un esprit différent par Hilbert une façon particulière et propre à la logique contemporaine de conduire cette recherche. L'un des résultats les plus remarquables des travaux menés en logique dans cette direction à partir du programme de Hilbert est la construction de systèmes qui réussissent à formaliser les raisonnements mathématiques et dont l'étude métamathématique permet de démontrer certaines propriétés. Un résultat non moins remarquable de l'entreprise de formalisation, révélé par les travaux de Gödel en 1931, est l'impossibilité de maintenir une stricte correspondance entre les vérités mathématiques et les énoncés formellement démontrables. Ainsi, après qu'on eut réussi à prouver qu'une explicitation intégrale des justifications d'un raisonnement mathématique est réalisable par élimination de tout recours à l'intuition

et par formalisation de la pensée, les résultats négatifs obtenus par Gödel, ainsi que d'autres qui suivirent[1], mirent en évidence quelques-unes des limites intrinsèques à cet idéal de formalisation, et notamment l'impossibilité de saisir et de représenter adéquatement par ce moyen l'ensemble de toutes les vérités mathématiques ou le prédicat de vérité. Or le chapitre VI a montré pourquoi la mise en évidence de ces limites peut être considérée comme l'un des prolégomènes à l'introduction en logique d'une notion de machine et l'une des origines de la perspective informatique sur l'étude des formalismes. Les systèmes formels furent étudiés indépendamment de la question du fondement des mathématiques, et l'on s'aperçut qu'ils exprimaient aussi bien des algorithmes que des preuves. Ils se révélèrent être particulièrement éclairants pour l'analyse de la calculabilité, du traitement de l'information, de la complexité des algorithmes et plus généralement des machines abstraites et des systèmes informatiques. En outre, la correspondance de Curry-Howard permit de voir dans les rapports entre logique et informatique les conditions d'un enrichissement mutuel. Surtout, elle mit en rapport deux sortes de constructions formelles syntaxiquement définies : en logique, celles qui sont utilisées pour l'expression des formules et des preuves ; en informatique, les machines abstraites et les langages de programmation dans lesquels s'écrivent les algorithmes. Les systèmes de typage dont il a été question au chapitre VIII reçoivent une double interprétation : du côté logique, les preuves sont une écriture formalisée de raisonnements mathématiques et du côté informatique, les programmes sont le modèle théorique de suites d'opérations effectuées par des machines réelles. Ce type de formalisme met parfaitement en évidence les rapports qui existent entre les deux aspects de la pensée mécanique ; entre le point de vue empirique des machines réelles, de leurs capacités à traiter l'information, et le point de vue logique relatif aux possibili-

1. A ceux que nous avons examinés il faut ajouter ici le théorème de Tarski sur l'impossibilité d'exprimer un prédicat de vérité par une formule dans un système formel consistant pour l'arithmétique.

tés de calcul, à la complexité des algorithmes. Par la voie de la formalisation, la machine en logique met en rapport les raisonnements mathématiques et les opérations informatiques.

C'est donc à partir d'une recherche sur la vérité, le raisonnement correct, leurs conditions de possibilité, leur expression logique dans le langage, et à travers l'entreprise de formalisation de la pensée mathématique que put être mise en évidence, isolée, considérée pour elle-même, une forme singulière de raisonnement et de pensée, abstraite, formelle, fruit d'un travail systématique d'élimination de l'intuition et qui s'exprime dans les constructions formelles de la logique et de l'informatique ; c'est une recherche concernant l'expression logique de la pensée qui conduisit les logiciens à concevoir, construire et étudier une forme de pensée mécanique.

Eu égard à son orientation vers les machines et l'informatique, la logique est encore analyse du langage. Mais il ne s'agit plus seulement de notre langage. Il est aussi question en logique des langages de programmation qui permettent le traitement mécanique de l'information et l'écriture des algorithmes. La logique est encore une discipline de la pensée et de ses règles. Mais elle ne se constitue plus seulement à partir d'une analyse des formes de la pensée humaine. Elle étudie aussi de manière systématique les constructions, les capacités et les limites de la pensé mécanique. La logique est encore une recherche relative aux conditions de possibilité les plus générales de la connaissance, et à la question de l'ordre du savoir. Mais cette analyse ne concerne pas uniquement les connaissances appréhendées ou le savoir ordonné par l'esprit. La logique est aussi devenue étude des méthodes et des possibilités de classification, de recherche et de communication des informations et des connaissances enregistrées dans des machines[1]. Lorsqu'elle prend pour objet les algorithmes et les machines, la logique se présente

1. Étude dont l'intérêt apparaît surtout lorsque la quantité des informations et des connaissances dépasse ce que l'esprit humain à lui seul serait capable de retenir et de maîtriser (le raisonnement mécanisé – diagnostic médical informatisé, démonstration automatique, etc. –, tout comme la constitution de banques de données, s'appuie le plus souvent sur des systèmes à base de connaissances, fondés sur des principes étudiés en logique).

donc encore comme une discipline du langage, de la connaissance et de la pensée; mais il s'agit alors d'une forme de pensée mécanique.

L'existence indéniable d'une orientation de la logique contemporaine vers les algorithmes et les machines n'implique évidemment en aucune manière qu'elle doive être conçue exclusivement comme une discipline de la pensée mécanique. Premièrement parce que toutes les parties, tous les développements de la logique ne se prêtent pas de manière égale à une interprétation informatique. Deuxièmement, parce que la logique concerne encore et avant tout l'analyse de notre langage, les lois de la pensée vraie et la constitution de la connaissance en un système ordonné. Ces questions relèvent traditionnellement de la logique et font partie des problèmes les plus généraux qu'elle s'efforce de résoudre. Mais si elles ne font, par elles-mêmes, nullement référence aux algorithmes et aux machines, elles peuvent pourtant être considérées du point de vue de la pensée mécanique et appellent alors une réflexion philosophique élargie et sensiblement différente. Pour ne donner qu'un exemple relatif à la constitution d'un système de la connaissance et la définition d'un ordre du savoir logiquement fondés, il n'est pas indifférent que certains raisonnements puissent être mécanisés, ou que d'innombrables informations et connaissances soient enregistrées dans des bases de données, et rendues accessibles, communiquées, sélectivement choisies par des machines, selon des opérations de calcul et de traitement de l'information dont l'étude relève précisément de la logique elle-même. En ce sens, le problème philosophique d'un système du savoir logiquement ordonné ne concerne plus seulement les sciences et disciplines rationnelles habituellement intégrées à l'encyclopédie, mais également les différentes formes de la pensée mécanique.

RÉFÉRENCES BIBLIOGRAPHIQUES

Abramsky S., Gabbay Dov M., Maibaum T. S. E., *Handbook of Logic in Computer Science*, vol. 1 à 6, Oxford, Clarendon Press, 1992-

Ackermann Wilhelm, Zum Hilbertschen Aufbau der reellen Zahlen, *Mathematische Annalen*, 99, 1928, trad. angl. *in* Van Heijenoort 1967.

Anderson Alan Ross, *Minds and machine*, Englewood Cliffs, NJ, Prentice-Hall, 1964, trad. franç., *Pensée et machine*, Seyssel, Champ Vallon, 1983.

Andler Daniel, Les sciences de la cognition, *in* Jean Hamburger (dir.), *La philosophie des sciences aujourd'hui*, Paris, Gauthier-Villars, 1986.

Andler Daniel (dir.), *Introduction aux sciences cognitives*, Paris, Gallimard, 1992.

Andler Daniel, Turing : pensée du calcul, calcul de la pensée, *Rapport n° 9611*, CREA, février 1996, à paraître dans Frédéric Nef *et al.*, *Les années 1930 : réaffirmation du formalisme*, Paris, Vrin.

Bechtel William, *Philosophy of Mind. An overview for Cognitive Science*, Hillsdale, NJ, Lawrence Erlbaum, 1988.

Blanché Robert, Dubucs Jacques, *La logique et son histoire*, Paris, Armand Colin, 1996.

Boden Margaret (ed.), *The Philosophy of Artificial Intelligence*, Oxford, Oxford UP, 1990.

Boole George, *An Investigation of The Laws of Thought, on which are founded the mathematical theories of logic and probabilities*, 1854, trad. franç., *Les lois de la pensée*, Paris, Vrin, 1992.

Bouveresse Jacques, *La parole malheureuse. De l'alchimie linguistique à la grammaire philosophique*, Paris, Éd. de Minuit, 1971.

Bruijn N. G. de, The mathematical language AUTOMATH, its usage and some of its extensions, *Symposium on Automatic Demonstration, IRIA, Versailles, 1968*, Berlin, Springer, « Lecture Notes in Mathematics », vol. 125, 1970.

Bruijn N. G. de, A survey of the project Automath, *in* Hindley et Seldin 1980.

Church Alonzo, A set of postulates for the foundation of logic, *Annals of Math.*, 2 s., 34, 1932, p. 346-366.

Church Alonzo, A set of postulates for the foundation of logic (second paper), *Annals of Math.*, 2 s., 34, 1933, p. 839-864.

Church Alonzo, A formulation of the simple theory of types, *Journal of Symbolic Logic*, vol. 5, 1940.

Church Alonzo, An unsolvable problem of elementary number theory, *The American Journal of Mathematics*, vol. 58, 1936, p. 345-363, reproduit *in* Davis 1965.

Church Alonzo, A Note on the Entscheidungsproblem, *Journal of Symbolic Logic*, vol. 1, n° 1 et vol. 1, n° 3, 1936, reproduit dans Davis 1965.

Coquand Thierry, Sur l'analogie entre les propositions et les types, *in* Cousineau, Curien, Robinet (eds.), *Combinators and Functional Languages*, Berlin, Springer, « Lecture Notes in Computer Science », n° 242, 1986, p. 71-84.

Coquand Thierry, Huet Gérard, Concepts mathématiques et informatiques formalisés dans le calcul des constructions, in *Logic Colloquium '85*, Proceedings of the Colloquium held in Orsay, France, July 1985, ed. by The Paris Logic Group, Amsterdam, North-Holland, 1987.

Coquand Thierry, Huet Gérard, The Calculus of Constructions, *Information and Computation*, 76, 1988.

Cori René, Lascar Daniel, *Logique mathématique*, t. 1 et t. 2, Paris, Masson, 1993.

Courcelle Bruno, *Logique et informatique. Une introduction*, Rocquencourt, Inria, 1990.

Cummins Robert, *The nature of psychological explanation*, Cambridge, Mass., Londres, The MIT Press, 1983.

Curry H. B., Feys R., *Combinatory Logic I*, Amsterdam, North-Holland, 1958.

Davis Martin, *The undecidable. Basic papers on undecidable propositions, unsolvable problems and computable functions*, Hewlett, NY, Raven Press, 1965.

Davis Martin, Why Gödel didn't have Church's thesis, *Information and control*, 1982, 54, p. 3-24.

Dennett Daniel, *Brainstorms. Philosophical Essays on Mind and Psychology*, Montgomery, Bradford, 1978, nouv. éd., Brighton, Harvester Press, 1981.

Descartes René, *Œuvres philosophiques*, t. I, II et III, éd. de F. Alquié, Paris, Garnier, 1963-1973.

Dreyfus Hubert, *What Computers Can't Do. The limits of Artificial Intelligence*, 1972, 2e éd., 1979, trad. franç., *Intelligence artificielle. Mythes et limites*, Paris, Flammarion, 1984.

Dreyfus Hubert, *What Computers* Still *Can't Do. A Critique of Artificial Intelligence*, Cambridge, Mass., The MIT Press, 1992.

Fagot Anne, Le concept de maladie sous-jacent aux tentatives d'informatisation du diagnostic médical, *Hist. Phil. Life Sci.*, 10, suppl., 1988.

Fenstad J. E. (ed.), *Proceedings of the Second Scandinavian Logic Symposium*, Amsterdam, North-Holland, 1971.

Frege Gottlob, *Nachgelassene Schriften und wissenschaftlicher Briefwechsel*, vol. 1, Hambourg, Felix Meiner, 1969.

Frege Gottlob, *Grundlagen der Arithmetik*, Breslau, Max und Hermann Marcus, 1884, trad. franç., *Les fondements de l'arithmétique*, Paris, Éd. du Seuil, 1969.

Frege Gottlob, *Écrits logiques et philosophiques*, Paris, Éd. du Seuil, 1971.

Gabbay Dov, Guenthner F. (eds.), *Handbook of Philosophical Logic*, Dordrecht, D. Reidel, vol. 1 à 4, 1983-1988.

Gallier Jean, *Logic for Computer Science. Foundations of Automatic Theorem Prooving*, New York, Harper & Row, 1986.

Ganascia Jean-Gabriel, *L'âme-machine. Les enjeux de l'intelligence artificielle*, Paris, Éd. du Seuil, 1990.

Ganascia Jean-Gabriel, *L'intelligence artificielle*, Paris, Flammarion, 1993.

Gandy Robin, The confluence of ideas in 1936, *in* Herken 1988.

Gardner Martin, *Logical machines and diagrams*, New York, McGraw-Hill, 1959, trad. franç., *L'étonnante histoire des machines logiques*, Paris, Dunod, 1964.

Gentzen Gerhard, Untersuchungen über das logische Schliessen, *Mathematische Zeitschrift*, 39, 1935, trad. franç. par R. Feys et J. Ladrière, *Recherches sur la déduction logique*, Paris, PUF, 1955, trad. angl. *in* M. E. Szabo, *The collected papers of Gerhard Gentzen*, Amsterdam, North-Holland, 1969.

Georgieff Nicolas, *La schizophrénie*, Paris, Flammarion, 1995.

Girard Jean-Yves, *Proof theory and logical complexity*, vol. 1, Naples, Bibliopolis, 1987.

Girard Jean-Yves, Linear Logic, *Theoretical Computer Science*, 50, 1987.

Girard Jean-Yves, Lafont Yves, Taylor Paul, *Proofs and Types*, Cambridge, Cambridge UP, 1989.

Girard Jean-Yves, Turing Alan M., *La machine de Turing*, Paris, Éd. du Seuil, 1995.

Gödel Kurt, Über eine bisher nicht benützte Erweiterung des finiten Standpunktes, *Dialectica*, 12, 1958, p. 280-287, trad. angl. révisée et complétée par Gödel en 1972, *in* Gödel, *Collected Works*, vol. II.

Gödel Kurt, *Collected works*, vol. I et II, New York, Oxford UP, 1986 et 1990.

Goldstine H. H., *The computer from Pascal to Von Neumann*, Princeton, NJ, Princeton UP, 1972.

Haugeland John, *Artificial Intelligence. The Very Idea*, Cambridge, Mass., Londres, The MIT Press, 1985, trad. franç., *L'esprit dans la machine. Fondements de l'intelligence artificielle*, Paris, Odile Jacob, 1989.

Heijenoort Jean van, *From Frege to Gödel. A Source Book in Mathematical Logic, 1879-1931*, Cambridge, Mass., Harvard UP, 1967.

Heijenoort Jean van, L'œuvre logique de Jacques Herbrand et son contexte historique, *in* J. Stern (ed.), *Proceedings of the Herbrand Symposium. Logic Colloquium '81*, Amsterdam, North-Holland, 1982.

Herken Rolf, *The Universal Turing Machine. A Half-Century Survey*, Oxford UP, 1988, 2e éd., Vienne, Springer, 1994.

Heyting A., *Intuitionism. An Introduction*, Amsterdam, North-Holland, 1956.

Hilbert David, Sur les problèmes futurs des mathématiques : les 23 problèmes, *Comptes rendus du IIe Congrès international de mathématiques tenu à Paris du 6 au 12 août 1900*, Gauthier-Villars, 1902, réimpr., Sceaux, Gabay, 1990.

Hilbert David, Ackermann Wilhelm, *Grundzüge der theoretischen Logik*, Berlin, Springer, 1928.

Hindley Roger, Seldin Jonathan (eds.), *To H. B. Curry : Essays on Combinatory Logic, Lambda Calculus and Formalism*, New York, Academic Press, 1980.

Hindley Roger, Seldin Jonathan, *Introduction to Combinators and λ-Calculus*, Cambridge, Cambridge UP, 1986.

Hodges Andrew, *Alan Turing : the enigma of intelligence*, Londres, Unwin Paperbacks, 1985, trad. franç., *Alan Turing : l'énigme de l'intelligence*, Paris, Payot, 1988.

Hopcroft John, Ullman Jeffrey, *Introduction to automata theory, languages and computation*, Reading, Mass., Addison-Wesley, 1979.

Howard W. A., *The formulæ-as-types notion of construction*, *in* Hindley, Seldin 1980.

Imbert Claude, *Phénoménologies et langues formulaires*, Paris, PUF, 1992.

Kleene Stephen C., *Introduction to metamathematics*, Amsterdam, North-Holland, 1952.

Kleene Stephen C., Origins of recursive function theory, *Ann. Hist. C.*, 1981, 3-1.

Krivine Jean-Louis, *Lambda-calcul, Types et Modèles*, Paris, Masson, 1990.

Krivine Jean-Louis, Une preuve formelle et intuitionniste du théorème de complétude de la logique classique, *The Bulletin of Symbolic Logic*, vol. 2, 4, décembre 1996.
Lalement René, *Logique, réduction, résolution*, Paris, Masson, 1990.
Largeault Jean, *Intuitionisme et théorie de la démonstration*, Paris, Vrin, 1992.
Lassaigne Richard, Rougemont Michel de, *Logique et fondements de l'informatique*, Paris, Hermès, 1993.
Lassaigne Richard, Rougemont Michel de, *Logique et complexité*, Paris, Hermès, 1996.
Leibniz Gottfried Wilhelm, *Die philosophischen Schriften von G. W. Leibniz*, éd. par C. I. Gerhardt, 7 vol., 1875-1890, rééd. Hildesheim, Georg Olms, 1978.
Leibniz Gottfried Wilhelm, *Opuscules et fragments inédits de Leibniz*, éd. par Couturat, Hildesheim, Georg Olms, 1903.
Leibniz Gottfried Wilhem, *Œuvres*, éd. par Lucy Prenant, Paris, Aubier Montaigne, 1972.
Marguin Jean, *Histoire des instruments et machines à calculer. Trois siècles de mécanique pensante. 1642-1942*, Paris, Hermann, 1994.
Martin-Löf Per, Constructive mathematics and computer programming, *in* Cohen, Los, Pfeifer, Podewski (eds.), *Proceedings of the sixth International Congress for Logic, Methodology and Philosophy of Science*, Amsterdam, North-Holland, 1982.
Martin-Löf Per, *Intuitionistic type theories*, Naples, Bibliopolis, 1984.
Minsky Marvin L., *Computation. Finite and Infinite Machines*, Englewood Cliffs, NJ, Prentice-Hall, 1967.
Moreau René, *Ainsi naquit l'informatique. Histoire des hommes et des techniques*, Paris, Dunod, 1987.
Mosconi Jean, *La constitution de la théorie des automates*, thèse d'État, Univ. Paris 1, 1989.
Odifreddi Piergiorgio (ed.), *Logic and Computer Science*, Londres, Academic Press, « The Apic Series », vol. 31, 1990.
Parigot Michel, Classical proofs as programs, *Proceedings KGC '93*, Berlin, Springer, « Lectures Notes in Computer Science », vol. 713, 1993.
Prawitz Dag, *Natural Deduction. A proof theoretical study*, Stockholm, Almquist & Wiksell, 1965.
Prawitz Dag, Ideas and results in proof-theory, *in* Fenstad 1971.
Putnam Hilary, Minds and machines, *in* Sidney Hook (ed.), *Dimensions of Mind : A Symposium*, New York, New York UP, 1960, reproduit *in* Anderson 1964.
Putnam Hilary, *Representation and reality*, Cambridge, Mass., The MIT Press, 1988, trad. franç., *Représentation et réalité*, Paris, Gallimard, 1990.
Randell B., *The origins of digital computers. Selected papers*, Berlin, Springer, 1972.
Rivenc François, *Introduction à la logique*, Paris, Payot, 1989.
Rivenc François, *Recherches sur l'universalisme logique. Russell et Carnap*, Paris, Payot et Rivages, 1993.
Rivenc François, Rouilhan Philippe de, *Logique et fondements des mathématiques. Anthologie (1850-1914)*, Paris, Payot, 1992.
Rosser Barkley J., Highlights of the history of the lambda-calculus, *Ann. Hist. C.*, 6-4, 1984.
Rouilhan Philippe de, De l'universalité de la logique, *L'âge de la science*, 4, t. 1, Paris, Odile Jacob, 1991.

Schönfinkel Moses, Über die Bausteine der mathematischen logik, *Mathematische Annalen 92*, 1924, trad. angl. *in* Van Heijenoort, 1967.

Scriven Michæl, The mechanical concept of mind, *Mind*, vol. LXII, 246, 1953, reproduit *in* Anderson 1964.

Searle John, Minds, Brains and Programs, *The Behavioral and Brain Sciences*, vol. 3, p. 417-458, 1980, trad. franç. partielle *in* Hofstadter, Dennett (dir.), *Vues de l'esprit*, Paris, InterÉditions, 1987.

Searle John, L'esprit est-il un programme d'ordinateur ?, *Pour la science*, n° 149, mars 1990.

Séris Jean-Pierre, Descartes et la mécanique, *Bulletin de la Société française de philosophie*, Paris, Armand Colin, t. LXXXI, avril-juin 1987.

Séris Jean-Pierre, *Langages et machines à l'âge classique*, Paris, Hachette, 1995.

Sperber Dan, Connaître l'acte de connaître, *Le Monde*, 22 octobre 1993.

Stern Jacques, *Fondements mathématiques de l'informatique*, Paris, McGraw-Hill, 1990.

Stern Nancy, *From ENIAC to UNIVAC. An appraisal of the Eckert-Mauchly Computer*, Bedford, Mass., Digital Press, 1981.

Thompson Simon, *Type Theory and Functional Programming*, Wokingham, Addison-Wesley, 1991.

Turing Alan M., On Computable Numbers, with an application to the Entscheidungsproblem, *Proceedings of the London Mathematical Society*, series 2, vol. XLII, reproduit *in* Davis 1965 ; trad. franç. *in* Girard, Turing 1995.

Turing Alan M., Intelligent Machinery, *Report to the National Physical Laboratory*, 1947, reproduit *in* Meltzer, Michie (eds.), *Machine Intelligence*, vol. 5.

Turing Alan M., Computing Machinery and Intelligence, *Mind*, vol. LIX, 236, 1950, reproduit *in* Anderson 1964, traduit dans Girard, Turing 1995.

Turner R., *Logics for Artificial Intelligence*, Chichester, NY, Ellis Horwood, 1984, trad. franç., *logiques pour l'intelligence artificielle*, Paris, Masson, 1986.

Wagner Pierre, *Machine et pensée : l'importance philosophique de l'informatique et de l'intelligence artificielle*, thèse de l'Univ. Paris I, 1994.

Wagner Pierre, Les sciences cognitives, l'intelligence artificielle et la connaissance de l'esprit, *Le Banquet*, 10, 1997.

Wagner Pierre, L'approche cognitive du mental, le point de vue de l'intelligence artificielle, à paraître dans la *Revue philosophique* en 1998.

Webb Judson C., *Mechanism, Mentalism, and Metamathematics. An essay on finitism*, Dordrecht, D. Reidel, 1980.

Webb Judson C., Gödel's Theorems and Church's Thesis. A Prologue to Mechanism, *in* R. S. Cohen, M. W. Wartofsky (eds.), *Language, Logic, and Method*, Boston Studies in the Philosophy of Science, vol. XXXIII, Dordrecht, D. Reidel, 1983, p. 309-353.

Winskel Glynn, *The formal semantics of programming languages. An introduction*, Cambridge, Mass., The MIT Press, 1993.

INDEX NOMINUM

INDEX RERUM