UN RETOUR AUX ORIGINES DU CALCULABLE

La thèse de CHURCH/POST

Alain CARDON, Christian CHARRAS, Daniel KROB

Groupe Informatique de l'IREM de Rouen

1.— Introduction

L'existence de l'Informatique nécessite a priori naturellement de se poser le problème de savoir ce qui est ou non manipulable automatiquement par une machine. C'est là tout le problème du calculable qui consiste à essayer de définir formellement le champ de compétence d'une machine abstraite qui modéliserait un ordinateur. Il est donc d'autant plus surprenant de savoir que cette problématique a émergé dès les années 30, bien avant l'existence des ordinateurs et que ce sont ces travaux qui permettront au contraire à la jeune science informatique de se développer sur des fondements théoriques solides.

De nombreux modèles de calculabilité ont été proposés au fil des ans. Les plus connus sont les suivants :

1) La théorie des fonctions récursives : il s'agit d'un formalisme développé par GÖDEL et HERBRAND vers 1934 (cf [1] p. 351) où la notion de calculabilité est modélisée par l'ensemble des fonctions de $\mathbb{N}^n$ dans $\mathbb{N}^m$ que l'on peut construire à partir des fonctions primitives suivantes :

- la fonction zéro de $\mathbb{N}^n$ dans $\mathbb{N}$ qui à tout n-uplet d'entiers associe 0,
- la fonction successeur de $\mathbb{N}$ dans $\mathbb{N}$ qui à tout $n \in \mathbb{N}$ associe $n+1$,
- les fonctions projections de $\mathbb{N}^n$ dans $\mathbb{N}$ qui à tout n-uplet d'entiers associe sa i-ième composante (i étant fixé dans $[1, n]$),

et à l'aide des trois opérations suivantes :

- la *composition* : si f et g sont des fonctions récursives, $f \circ g$ est aussi récursive quand elle est définie,
- la *récurrence* : si f et g sont des fonctions récursives, alors la fonction h définie par les relations $h(x_1, \ldots, x_n, y+1) = g(x_1, \ldots, x_n, y, h(x_1, \ldots, x_n, y))$ et $h(x_1, \ldots, x_n, 0) = f(x_1, \ldots, x_n)$ est aussi récursive,
- la *minimisation* : si f est une fonction récursive, alors la fonction qui associe à tout n-uple $(x_1, \ldots, x_n)$, le plus petit y tel que $f(x_1, \ldots, x_n, y) = 0$, s'il existe, est aussi récursive.

2) Le lambda-calcul : c'est une théorie dont les fondements ont été définis en 1933-1935 par CHURCH et KLEENE (cf [2,5]). L'intention de ses créateurs était l'étude des règles de définition liées à la notion de fonction. De façon plus précise,

une fonction est pensée ici comme un lien abstrait entre une fonction et une valeur, ce qui est fondamentalement différent de la vision classique d'une fonction comme un graphe. Dans le lambda-calcul, l'accent est aussi mis sur le caractère structurellement libre des objets manipulés qui sont à la fois fonctions et arguments (en particulier une fonction peut s'appliquer à elle-même).

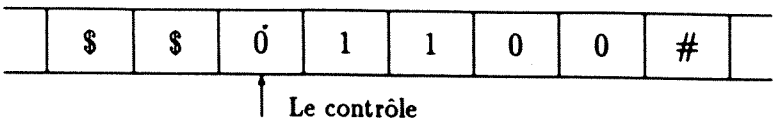
Le lambda-calcul permet de modéliser une classe de fonctions où la composition est l'opération primitive. Pour cela, on se donne d'abord un alphabet de variables. On construit alors inductivement des termes - qui représentent les fonctions que l'on cherche à modéliser - à l'aide des règles suivantes :
- Toute variable est un terme.
- Si M est un terme et si x est une variable, $\lambda x.M$ est un terme.
- Si M, N sont des termes, $M.N$ est un terme.

La notation $\lambda x.M$ modélise le fait que M est une fonction de x. La sémantique de la dernière règle de construction ci-dessus est donnée par la règle de transformation de termes appelée β-réduction qui est définie par :

$$(\lambda x.M).N \longrightarrow M[x/N]$$

où $M[x/N]$ désigne la substitution de N à la variable x dans M. Cette règle montre donc que le terme $M.N$ modélise la composition des deux fonctions représentées par les termes M et N. On peut alors définir à l'aide du lambda-calcul une classe de fonctions sur les entiers dites λ-définissables. Il est à noter que CHURCH a prouvé que ces fonctions sont exactement les fonctions récursives au sens de GÖDEL-HERBRAND.

3) Les machines de TURING : il s'agit d'un modèle mécanique introduit par TURING en 1936 en [7]. Dans ce modèle, les calculs sont effectués par un programme qui interagit sur une bande doublement infinie. De façon plus précise, on dispose d'une bande séparée en cases dans lequelles on peut placer ou non des symboles (en particulier, on peut coder ici tout entier par une succession de symboles du même type séparés par deux délimiteurs).



Une tête de lecture/écriture pointe sur l'une des cases de la bande et selon le contenu de ce qu'elle voit, modifie ou non la case qu'elle pointe ou se déplace d'une case vers la droite ou la gauche. Ce mécanisme est contrôlé par un programme qui a pour but de répéter de façon identique ce qu'on observe dans des situations identiques.
L'objectif de TURING était de modéliser de manière discrète tout procédé calculable, même s'il relève a priori du domaine du continu. Il est intéressant de noter que dans l'article originel de TURING (cf [7]), tout un paragraphe est consacré à une justification philosophico-mathématique du fait que tout procédé de calcul peut être effectué dans ce modèle : l'argumentation de TURING est basée sur le fait que

lorsqu'un individu effectue un calcul réel, on peut discrétiser à la fois l'espace dans lequel se fait ce calcul et le temps durant lequel se déroule ce calcul de manière à ce que le cerveau du "calculateur" passe de manière discrète d'un état à un autre.

4) Les machines de POST : c'est un modèle très proche de celui de TURING, qui a été introduit indépendemment de ce dernier par POST (cf [6]) dans l'article étudié en détails ci-après. Le lecteur pourra donc se référer au paragraphe 2.2 qui suit pour la description précise de ce type de machines.

C'est CHURCH le premier qui a prouvé l'équivalence (c'est-à-dire la même puissance d'expressibilité) entre le λ-calcul et la théorie des fonctions récursives (cf [1]). Ce résultat a motivé le problème fondamental de savoir si deux modèles quelconques de calculabilité sont équivalents ou non. C'est POST qui a émis en fait le premier, peu après la publication du résultat de CHURCH en 1936, la thèse philosophique (1) (connue maintenant sous le nom de thèse de CHURCH) affirmant que tous les modèles de calculabilité sont équivalents. De fait, cette thèse a toujours été vérifiée par la suite.

Il nous paraît aussi important de replacer rapidement ce problème du calculable dans son contexte historique de façon à pouvoir comprendre comment cette question a émergé. Pour cela, il faut remonter au début du siècle. En effet, Hilbert voulait à cette époque prouver la non-contradiction des Mathématiques par la méthode suivante :

1. Tout d'abord, réaliser un codage de tout énoncé mathématique sous une forme inambigüe et purement formelle.
2. Ensuite, construire un algorithme permettant de décider si un énoncé donné sous la forme obtenue par la première étape, est vrai ou faux.

La première partie du programme de HILBERT a été réalisée et a conduit au développement de la logique formelle. La seconde a échoué en raison du théorème d'incomplétude de GÖDEL qui prouve qu'il existe toujours des propositions indécidables dans tout système formel contenant au moins l'arithmétique. L'échec du programme de HILBERT a conduit tout naturellement à essayer de définir un domaine plus restreint où des mécanismes de décision pouvaient être fabriqués : c'est ainsi qu'est apparu le domaine du calculable dont l'étude est à la base de la science informatique (2).

2.— Emil L. POST : Processus combinatoires finis - Un premier modèle

2.1. Présentation

L'article d'Emil POST que nous allons maintenant présenter plus en détails, est paru sous le titre "Finite combinatory Processes - Formulation 1" dans le Journal of Symbolic Logic en Septembre 1936. Il nous a paru intéressant d'en proposer une

(1) L'adjectif "philosophique" est là pour bien indiquer que cette thèse n'est pas une affirmation à caractère scientifique - puisqu'elle n'est pas vérifiable par essence - mais un parti-pris idéologique.
(2) Il est intéressant de noter que l'on peut construire explicitement des fonctions de $\mathbb{N}$ dans $\mathbb{N}$ non calculables. Le lecteur pourra se référer à [4] où est présentée une fonction (la fonction de Rado) qui croit plus vite que toute fonction calculable.

traduction commentée car il s'agit d'un texte qui joue en quelque sorte un rôle pivot dans l'histoire du calculable et donc des fondements théoriques de l'informatique.

L'un des intérêts de cet article est qu'il est un des premiers à présenter un modèle "mécaniste" de calculabilité. En effet, le texte de POST est consacré à la description de ce que nous appelons maintenant la *machine de* POST. Il est à noter que le formalisme des machines de TURING (cf [7]) est apparu la même année que celui des machines de POST, mais de façon tout à fait indépendante. Bien que le modèle de TURING ait désormais supplanté celui de POST, ce dernier garde cependant tout son intêret épistémologique car il dénote une rupture historique fondamentale avec les précédents modèles de calculabilité qui étaient tous de nature fonctionnelle ou logique.

Le second intérêt majeur de cet article est qu'il énonce, vraisemblablement pour la première fois, la thèse philosophique, appelée désormais *thèse de* CHURCH, qui affirme l'équivalence entre tous les modèles possibles de calculable. En fait, la plupart des auteurs contemporains citent l'article [1] de CHURCH comme référence originelle pour cette thèse bien qu'il ne contienne nulle part cette affirmation de nature plus philosophique (ou idéologique) que mathématique. Dans [1], CHURCH se contente en effet de montrer l'équivalence de deux modèles de calculabilité (le lambda calcul qu'il vient de créer et la théorie des fonctions récursives au sens de GÖDEL-HERBRAND). Tout au contraire, POST termine son article [6] en affirmant de manière très forte que, dès qu'un nouveau modèle de calculabilité sera développé, il faudra montrer son équivalence avec un modèle déjà existant. De plus, POST parle bien d'un tel phénomène comme d'une *loi naturelle.* Il nous semble donc incontestable que la thèse de CHURCH doive en fait être atribuée à POST.

Terminons cette courte présentation en signalant que nous avons choisi dans la traduction qui suit, d'utiliser une numérotation classique pour toutes les notes que nous apportons au texte et d'utiliser un des symboles $*$, †, ‡ ou § pour les notes originelles de POST ou de son éditeur.

2.2. Traduction et commentaires

* (3)
* Le modèle (4) que nous présentons ici devrait avoir une grande importance pour le développement de la logique symbolique. Il s'inscrit dans le prolongement du théorème d'incomplétude de GÖDEL et des résultats de CHURCH sur les problèmes

---

* Le lecteur pourra comparer cet article avec celui de A.M. TURING "Sur les nombres calculables" (3) qui devrait paraître sous peu dans les Compte-Rendus de la Société Mathématique de Londres. Notre article cependant, bien que publié ultérieurement, a été écrit de manière totalement indépendante de celui de TURING. *Note de l'Editeur.*

(3) Il s'agit de l'article de TURING dans lequel ce dernier introduisait les "machines de TURING".
(4) Nous avons choisi de traduire systématiquement par "modèle" le terme anglais "formulation" utilisé par POST,— faisant en cela une relecture a posteriori de cet article : en effet, la "machine de POST" que celui-ci présente ici n'est désormais pour nous qu'un modèle de calculabilité parmi d'autres.

indécidables (5) (6).

Nous considèrerons qu'un *problème global* (7) consiste en une classe de *problèmes particuliers* (8). Une solution du problème global devra fournir une réponse à chaque problème particulier.

Pour préciser la notion de solution dans notre modèle, nous introduirons deux concepts : celui d'un *espace symbolique* (9) dans lequel le travail conduisant du problème à sa réponse est réalisé (†) et celui d'un *ensemble* inaltérable *d'instructions* qui à la fois contrôle les opérations dans l'espace symbolique et détermine l'ordre dans lequel les instructions doivent être appliquées.

Dans ce modèle, l'espace symbolique est en une suite doublement infinie d'espaces ou de boîtes, i.e. est en bijection ordonnée avec la suite des entiers $.., -3, -2, -1, 0, 1, 2, 3, ..$ L'opérateur (10) (c'est-à-dire celui qui est chargé de résoudre le problème) doit nécessairement se déplacer et travailler dans cet espace symbolique et n'est capable que de se trouver dans une et une seule boîte à la fois et de la manipuler. A part la présence de l'opérateur, une boîte est réputée n'avoir que deux positions possibles : à savoir être vide (ou non marquée) ou bien avoir une marque unique, par exemple un trait vertical (11).

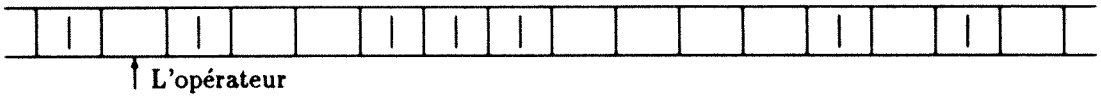
On distingue une boîte que l'on appelle le point de départ. Nous supposerons désormais de plus qu'un problème particulier est donné sous une forme symbolique

---

† Il y a à la fois un espace et un temps symboliques.

---

(5) Nous avons choisi de traduire l'expression anglaise "absolutely unsolvable", mot à mot "complètement non résoluble", par le terme plus moderne "indécidable".
(6) POST fait ici référence dans son article : 1) à l'article fondamental de GÖDEL de 1931 [3] dans lequel le théorème d'incomplétude a été énoncé; 2) à l'article de CHURCH [1].
(7) POST emploie le terme "general problem", c'est-à-dire "problème général", que nous avons choisi de rendre systématiquement ici par "problème global".
(8) Dans ce paragraphe, POST introduit le concept de "problème" qu'il "définit" plutôt vaguement. A la lumière du développement de l'informatique, nous pouvons risquer quelques hypothèses : par "problème global", POST entend vraisemblablement une question d'ordre général susceptible d'être résolu par un traitement informatique, comme par exemple "calculer le carré d'un nombre entier" ou bien "trier un jeu de données". Dans ces deux exemples, le problème consiste en la liste des cas particuliers correspondants. Ainsi, les deux exemples précédents reviennent respectivement à considérer la liste des problèmes particuliers : ("calculer le carré de n")$_{n\in\mathbb{N}}$ et ("trier la liste L")$_{L\in Listes}$. Pour être plus précis, des problèmes particuliers seront donc ici "calculer le carré de 7" ou "trier la liste (1 9 9 0)".
(9) Nous avons choisi de traduire le terme "symbol space" par "espace symbolique" plutôt que par "espace des symboles".
(10) Nous traduisons ici par "opérateur" le terme anglais "worker", c'est-à-dire "travailleur".
(11) On peut donc visualiser la machine de POST dont l'auteur est en train de préciser le fonctionnement comme une bande doublement infinie de la forme suivante :



Cette bande est l'espace symbolique dans lequel un programme (i.e. l'ensemble d'instructions) pilote la tête de lecture (i.e. l'opérateur) pour effectuer un certain nombre d'opérations.

par un certain nombre de boîtes marquées d'un trait. De même, la réponse sera donnée de manière symbolique à l'aide d'une configuration similaire de boîtes marquées. Pour être précis, la réponse est la configuration des boîtes marquées qui restent dans l'espace symbolique à la fin du processus de résolution (12).

L'opérateur est supposé capable de réaliser les opérations primitives suivantes *:

(a) *Marquer la boîte (supposée vide) dans laquelle il se trouve*
(b) *Effacer la marque de la boîte (supposée marquée) dans laquelle il se trouve*
(c) *Se déplacer vers la boîte à sa droite*
(d) *Se déplacer vers la boîte à sa gauche*
(e) *Déterminer si la boîte où il se trouve, est ou non marquée*

L'ensemble des instructions qui - cela doit être remarqué - est le même pour tous les problèmes particuliers et est donc adapté au problème général, doit être de la forme suivante. Il doit commencer par l'instruction :

Partir au point de départ et suivre l'instruction 1.

Puis il consiste en un ensemble fini d'instructions numérotées $1, 2, 3, \ldots, n$. La i-ième instruction doit être de l'un des types suivants :

(A) *Effectuer l'opération* O_i [$O_i = (a), (b), (c)$ *ou* (d)] *et passer à l'instruction* j
(B) *Effectuer l'opération (e) et suivant la réponse, passer à l'instruction* j_i' *ou* j_i''
(C) *Arrêter.*

Il suffit clairement qu'une et une seule intruction soit du type (C). On remarquera aussi que l'état de l'espace symbolique n'affecte directement le processus qu'à travers les instructions de type (B) (13).

Un ensemble d'instructions sera dit *applicable* à un problème global donné si, dans son application à chaque cas particulier de ce problème, il ne permet jamais l'exécution d'une opération (a) (resp. (b)) quand la boîte où se trouve l'opérateur, est marquée (resp. non marquée) †. Un ensemble d'instructions applicable à un problème global induit un processus déterministe quand il est appliqué à chaque cas particulier de ce problème. Ce processus se terminera quand et seulement quand il rencontrera l'instruction de type (C). On dira alors que l'ensemble d'instructions détermine un *1-processus fini* associé au problème global s'il est applicable au problème et *si le processus qu'il détermine se termine pour chaque problème*

* Et, bien sûr, de respecter les instructions décrites ci-dessous.
† Bien que notre définition d'un ensemble d'instructions aurait pu être aisément formulée de façon à ce que l'applicabilité soit automatiquement assurée, il ne vaut mieux pas procéder ainsi pour de multiples raisons.

(12) L'auteur propose donc ici un codage (binaire) des données.
(13) On remarquera que les instructions (A) sont des instructions de branchement inconditionnel et que les instructions (B) sont des instructions de test. Autrement dit, le "langage" séquentiel que donne POST contient un "Goto" et un "If Then Else", ce qui est assez puissant pour décrire l'ensemble des fonctions calculables.

particulier (14). On dira qu'un 1-processus fini associé à un problème global est une *1-solution* du problème si la réponse qu'il apporte à chaque problème particulier est toujours correcte (15).

Nous ne nous intéresserons pas ici à la façon dont la configuration des boîtes marquées correspondant à un problème particulier ou à sa réponse, symbolise la sémantique du problème (16) et de sa solution. En fait, nous supposons ici que le problème particulier est donné sous forme symbolique grâce à un mécanisme extérieur et qu'il en est de même pour la réception de la réponse. On peut plus précisément dire que le problème général consiste en une infinité au plus dénombrable de problèmes particuliers. Nous ne considérerons pas le cas fini. Imaginons donc une bijection établie entre la classe des entiers positifs et celle des problèmes particuliers. Nous pouvons représenter par exemple l'entier n en marquant les n premières boîtes à droite du point de départ (17). On dira que le problème général est *1-donné* si on peut obtenir un 1-processus fini qui énumère de manière bijective la classe des problèmes particuliers constituant le problème global, quand il est appliqué à la classe des entiers positifs symbolisée comme précédemment. De plus, il sera agréable de supposer que quand le problème global est 1-donné, chaque processus particulier replace l'opérateur au point de départ. Alors, si un problème global est 1-donné et 1-résolu, nous pouvons combiner, avec quelques modifications évidentes, les deux ensembles d'instructions pour obtenir un 1-processus fini qui donne la réponse à chaque problème particulier quand ce dernier est donné par un entier sous forme symbolique (18).

Avec quelques modifications, la formulation précédente est aussi applicable à la logique symbolique. Nous n'avons pas maintenant une classe de problèmes particuliers, mais un unique marquage initial fini de l'espace symbolique, représentant les axiomes de la logique considérée. D'un autre côté, il n'y aura plus d'instruction du type (C). En conséquence, si l'on suppose l'applicabilité réalisée, on obtiendra un processus déterministe *sans fin*. Nous supposerons aussi que dans l'exécution de ce processus, certains groupes de symboles seront reconnaissables - i.e. certaines suites finies de boîtes marquées et non marquées - et qu'ils ne seront pas modifiés

---

(14) Cette première définition est essentiellement de nature syntaxique : un programme est un 1-processus fini s'il vérifie à la fois la condition de cohérence qu'est l'applicabilité et une condition de terminaison.

(15) Cette deuxième définition est de nature sémantique. En effet, un 1-processus fini n'est qu'un programme correct qui s'arrête, mais qui peut faire n'importe quoi (comme répondre n^2 alors que la fonction à calculer est $n \longrightarrow 2.n$ par exemple). Un 1-processus fini devient donc une 1-solution quand il résoud bien le problème qu'on lui demande de résoudre.

(16) L'auteur a employé ici le terme "meaningful problem", mot à mot "problème significatif", que nous avons choisi de rendre par "sémantique du problème".

(17) On remarquera que POST élude ici le problème, toujours délicat, de la représentation de l'entier 0.

(18) Le processus que POST décrit, permet de ramener la notion de problème global à celle de fonction de $\mathbb{N}$ dans l'espace des réponses. En effet, il nous dit simplement que si chaque problème particulier est codé par un entier, alors on peut combiner un programme qui associe à chaque entier le problème particulier qu'il code et le programme de résolution du problème particulier pour obtenir un programme résolvant bien le problème global mais qui part de $\mathbb{N}$.

par la suite du processus quand ils apparaitront. . Ces groupes seront les énoncés dérivés de cette logique. Bien entendu, l'ensemble des instructions correspond aux règles de déduction de cette logique. On dira alors que cette logique est *1-engendrée* (19).

Un autre moyen, dans un esprit un peu plus éloigné de la logique formelle, consisterait à créer un 1-processus fini qui nous donnerait le n-ième théorème ou la n-ième assertion formelle de cette logique quand n est donné sous la même forme symbolique que précédemment.

Le concept de problème particulier que nous avons introduit, contient une difficulté qui doit être mentionnée. En effet, si un procédé extérieur donne le marquage fini initial de l'espace symbolique, il n'y a pas de moyen pour nous de déterminer quelle est par exemple la première ou la dernière boîte marquée. On évite complètement cette difficulté quand le problème général est 1-donné. On l'évite également dès que l'on s'est donné un 1-processus bien précis. En pratique, les problèmes particuliers significatifs seront symbolisés de telle manière que les bornes du marquage correspondant soient reconnaissables par des groupes caractéristiques de boîtes marquées et non marquées.

Le point fondamental de nos difficultés réside cependant dans notre hypothèse d'un espace symbolique infini. Dans notre modèle, les boîtes sont, au moins conceptuellement, des entités physiques comme par exemple des carrés contigus (20). Si on suppose donné un mécanisme extérieur, celui-ci ne peut pas plus nous fournir un nombre infini de ces boîtes, qu'il ne peut marquer une infinité de celles-ci. Mais si ce mécanisme traite le problème particulier dans une partie finie de l'espace symbolique, la difficulté disparait. Bien entendu, cela demande une extension des opérations primitives pour autoriser une nécessaire expansion de l'espace symbolique quand le processus en a besoin. Une version complète de notre modèle comporterait aussi des instructions pour engendrer l'espace symbolique (‡).

L'auteur espère que le modèle présenté ici s'avèrera être logiquement équivalent à la récursivité au sens de GÖDEL-CHURCH (§) . Son objectif cependant n'est pas seulement de présenter un système ayant une certaine potentialité logique, mais d'être

‡ Le développement du modèle 1 est en fait plutôt simple pour le moment. Bien que ce ne soit pas dans l'esprit d'un tel modèle, sa forme définitive pourra peut-être perdre une partie de sa simplicité pour permettre une plus grande souplesse. On pourra par exemple marquer une boîte de plus d'une façon. Le développement le plus naturel conduira peut-être à autoriser un nombre fini - par exemple 2 - d'objets physiques pour servir de curseurs, que l'opérateur pourra identifier et déplacer de boîte en boîte.

(§) Cette relation pourrait peut-être être établie de façon plus claire en définissant une notion de 1-fonction et en montrant alors qu'elle est équivalente à celle de fonction récursive (cf [1]). On pourrait définir une 1-fonction $f(n)$ sur les entiers positifs de telle façon qu'on puisse construire un 1-processus qui pour tout entier n modélisant un problème donné, produit $f(n)$ comme réponse; n et $f(n)$ étant définis symboliquement comme nous l'avons déjà vu.

(19) POST décrit ici le mécanisme, maintenant classique, de dérivation des énoncés valides d'une théorie axiomatisable en logique du premier ordre.

(20) Il semblerait que le distinguo moderne entre modèle et réalité ne soit pas encore très clair dans la pensée de POST.

aussi un modèle qui soit satisfaisant pour l'esprit. Dans cette dernière direction, on peut chercher des modèles de plus en plus larges. *D'un autre côté, notre objectif sera alors de montrer que tous ces modèles sont logiquement équivalents à notre premier modèle* . (21). Nous offrons pour le moment cette conclusion comme une *hypothèse de travail.* Nous pensons en particulier que l'identification de CHURCH entre la calculabilité effective et la récursivité est un tel résultat * (23). Outre cette hypothèse, et malgré son apparente contradiction avec tous les travaux mathématiques issus de la preuve de Cantor sur la non dénombrabilité du continu, la recherche se poursuit de façon indépendante sur les traces de GÖDEL et de CHURCH (24). Le succès du programme précédent changerait pour nous cette hypothèse, non pas tant en une définition ou un axiome qu'en une *loi naturelle.* Il semble à l'auteur que ce n'est que de cette façon que le théorème de Gödel sur l'incomplétude de certaines logiques symboliques et que les résultats de CHURCH sur l'indécidabilité (25) de certains problèmes pourront être transformés en conclusions concernant toutes les logiques symboliques et toutes les méthodes de résolution.

Références

[1] CHURCH A., *An unsolvable problem of elementary number theory*, American Journal of Mathematics, **58**, p. 345-363, 1936

[2] CHURCH A., Annals of Mathematics, **34**, 1933

[3] GÖDEL K., *Über formal unenetscheidbare Sätze der Principia Mathematica und verwandter Systeme I*, Monatshefte Math. Phys., **38**, p. 173-198, 1931

[4] HOPCROFT J., *Les machines de* TURING, Pour la Science, Juillet 1984

[5] KLEENE S.C., American Journal of Mathematics, **57**, 1935

[6] POST E., *Finite combinatory processes - Formulation I*, Journal of Symbolic Logic, **1**, 3, p. 103-105, 1936

[7] TURING A., *On computable numbers, with an application to the entscheidungsproblem*, Proc. of the London Math. Soc., Ser. 2, **42**, p. 230-265, 1936

* Cf CHURCH [1]. En fait le travail déjà effectué par CHURCH et d'autres porte cette identification considérablement au delà de l'étape d'une hypothèse de travail. Mais, masquer cette identification derrière une définition, cache le fait qu'une découverte fondamentale dans les limitations du pouvoir mathématisant de l'Homo Sapiens a été faite et nous fait oublier qu'il faut continuellement la vérifier.

(21) Nous avons mis en italique cette phrase car voilà affirmée pour la première fois de manière tout à fait claire la "thèse de CHURCH".
(23) POST fait référence au travail de CHURCH (cf ([1])) qui a montré l'équivalence entre calculabilité au sens du lambda-calcul et au sens de GÖDEL-HERBRAND.
(24) La contradiction dont parle POST pourrait être liée au fait que les fonctions calculables ne peuvent essentiellement manipuler que des structures discrètes et dénombrables et donc ne peuvent pas vraiment agir sur les nombres réels par exemple.
(25) Nous avons choisi de traduire le terme "recursive unsolvability", c'est-à-dire "non résolubilité récursive", qu'utilise POST, par le terme plus moderne "indécidabilité".