Philosophia Scientiæ
Travaux d'histoire et de philosophie des sciences

16-3 | 2012
Alan Turing

Les deux formes de la thèse de Church-Turing et l'épistémologie du calcul

Maël Pégny

Édition électronique
URL : http://journals.openedition.org/philosophiascientiae/769
DOI : 10.4000/philosophiascientiae.769
ISSN : 1775-4283

Éditeur
Éditions Kimé

Édition imprimée
Date de publication : 1 novembre 2012
Pagination : 39-67
ISBN : 978-2-84174-603-3
ISSN : 1281-2463

Référence électronique
Maël Pégny, « Les deux formes de la thèse de Church-Turing et l'épistémologie du calcul », *Philosophia Scientiæ* [En ligne], 16-3 | 2012, mis en ligne le 01 novembre 2015, consulté le 19 avril 2019. URL : http://journals.openedition.org/philosophiascientiae/769 ; DOI : 10.4000/philosophiascientiae.769

Les deux formes de la thèse de Church-Turing et l'épistémologie du calcul

Maël Pégny
Université de Paris 1 (France)

Résumé : La thèse de Church-Turing stipule que toute fonction calculable est calculable par une machine de Turing. En distinguant, à la suite de nombreux auteurs, une forme algorithmique de la thèse de Church-Turing portant sur les fonctions calculables *par un algorithme* d'une forme *empirique* de cette même thèse, portant sur les fonctions calculables *par une machine,* il devient possible de poser une nouvelle question : les limites empiriques du calcul sont-elles identiques aux limites des algorithmes ? Ou existe-t-il un moyen empirique d'effectuer un calcul qu'aucun algorithme ne permet d'effectuer ? Je montrerai ici la pertinence philosophique de cette question. Elle interroge la capacité de processus symboliques comme les calculs à simuler certains processus empiriques. Elle permet également d'étudier le statut épistémologique des calculs réalisés par des machines. S'il existait une fonction calculable par une machine sans être calculable par un algorithme, il existerait un problème mathématique qui serait soluble par un dispositif empirique, sans être soluble par aucune méthode mathématique *a priori*.

Abstract: Church-Turing's thesis states that every computable function is computable by a Turing machine. By distinguishing, like many authors in the recent literature, between an algorithmic form of Church-Turing's thesis on the functions computable by an algorithm, and an empirical form of the same on the functions computable by a machine, it becomes possible to ask a new question: Are the limits of empirical calculation identical to the limits of the algorithms? Or is there a way to empirically perform a calculation no algorithm can perform? I will show the philosophical relevance of this question. It questions the ability of symbolic processes as calculations to simulate certain empirical processes. It is also essential to the epistemological status of calculations performed by machines. If there were a function computable by a machine without being computable by an algorithm, there would be a mathematical problem, which would be solvable by an empirical device, but would be unsolvable by any *a priori* mathematical method.

Philosophia Scientiæ, 16 (3), 2012, 39–67.

Introduction

Selon une opinion presque universellement partagée dans la communauté informatique, la thèse de Church-Turing, selon laquelle toute fonction calculable est calculable par une machine de Turing, énonce les justes limites de la calculabilité. De nombreux auteurs, tels que R. Gandy [Gandy 1980], J. Copeland [Copeland 2003], W. Sieg [Sieg 2008], D. Deutsch [Deutsch 1985], [Deutsch 1997], Peter Smith [Smith 2007], G. Dowek [Dowek 2007] ou G. Piccinini [Piccinini 2011] ont néanmoins insisté sur la nécessité de distinguer deux formes de cette thèse de Church-Turing. La première forme, ou forme *algorithmique,* affirme que toute fonction calculable *par un algorithme* est calculable par une machine de Turing. La seconde, ou forme empirique, affirme que toute fonction calculable *par une machine* est calculable par une machine de Turing. Or, si la première forme est relativement consensuelle, la seconde est plus difficile à définir, et la question de sa démonstration au sein des théories physiques fondamentales est encore ouverte. Il n'est donc pas à l'heure actuelle établi si les limites du calcul fixées par ces deux propositions coïncident : *l'ensemble des fonctions calculables par une machine est-il identique à l'ensemble des fonctions calculables par un algorithme ?*

Le présent article vise à dégager l'intérêt philosophique de cette question, et en particulier de la thèse suivante, nommée thèse A : *toute fonction calculable par une machine est calculable par un algorithme.*

Pour ce faire, je commencerai par justifier la distinction entre ces deux formes de la thèse de Church-Turing (partie 1). Après avoir énoncé précisément ces deux thèses, et fixé la terminologie (section 1.1), je justifierai leur distinction en discutant la définition des concepts d'« algorithme » et de « machine » (sections 1.2.1 et 1.2.2). Je montrerai notamment que, bien qu'il soit très difficile de définir ce qu'est un algorithme, il est possible d'énoncer une condition significative sur le calcul d'une fonction par un algorithme : ce calcul est descriptible par un texte écrit dans un langage formel, et exécutable dans un langage formel. Cela me mènera à souligner l'importance du caractère symbolique du calcul effectué en suivant un algorithme.

La distinction entre les deux formes de la thèse de Church-Turing permet de formuler la thèse A, et de dégager son intérêt philosophique (partie 2). Après avoir discuté son énoncé (section 2.1), je montrerai que la thèse A implique une thèse non-triviale : si la thèse A est vraie, nos langages formels nous permettent de simuler l'intégralité des procédures *empiriques* implémentant une fonction (section 2.2). La thèse A est également importante pour l'examen du statut épistémologique des calculs effectués sur des machines (section 3). À l'heure actuelle, on n'utilise des machines de calcul que pour des raisons pragmatiques : les tâches effectuées par une machine demeurent en droit simulables par l'exécution d'un algorithme par un calcul papier-crayon. Si la thèse était fausse, cette situation serait bouleversée : il existerait une machine qu'on ne

pourrait en droit simuler à l'aide d'un algorithme. Je finirai en montrant l'importance de cette dernière remarque pour la discussion du caractère *a priori* de la connaissance obtenue par le calcul (section 4).

1 Les deux formes de la thèse de Church-Turing

1.1 Énoncés

La thèse de Church-Turing porte sur les limites de la calculabilité : elle affirme que toute fonction entière [1] calculable est une fonction calculable par une machine de Turing. On peut tout aussi bien parler de fonctions λ-définissables, de fonctions récursives, de fonctions calculables par une machine à registres, par une machine de Post, etc., puisqu'il a été démontré que toutes ces conceptions sont extensionnellement équivalentes. À de pures fins de brièveté, j'emploierai le plus souvent la formulation « fonction récursive ».

À partir de la calculabilité des fonctions, la notion de calculabilité peut être étendue à des objets non-fonctionnels, comme les ensembles, les prédicats, les relations ou encore les équations [2]. Pour éviter d'introduire des complications techniques inutiles à notre propos philosophique, j'ignorerai ces extensions dans la suite de cet article, et je parlerai toujours d'évaluation de fonctions.

On distingue couramment dans la littérature deux formes de cette thèse :

a) *Toute fonction calculable par un algorithme est récursive.* On parle alors parfois de « procédure mécanique », au sens où l'on pense qu'un homme qui calcule en suivant un algorithme agit de manière mécanique, sans qu'aucun dispositif d'ingénierie ne soit en jeu.

b) *Toute fonction calculable par une machine est récursive.* La thèse se réfère bien alors à un dispositif réalisant le calcul à la place d'un homme.

Dans l'immense majorité des cas, c'est la première forme de la thèse qui est mentionnée, notamment dans les manuels d'informatique et de théorie de la calculabilité. Elle fait l'objet d'un consensus pratiquement universel dans la communauté informatique. Il en existe de très nombreuses variantes : la thèse de Church-Turing étant exprimée dans un langage informel, il n'en existe pas de formulation standard. En lieu et place d'« algorithme », on pourra tout aussi bien lire « procédure effective », « procédure effectivement calculable », « procédure mécanique », « procédure combinatoire finie », « procédure finitaire », ou tout simplement « calcul ». Dans le cadre de cet article,

1. On peut étendre la notion de fonction calculable à d'autres domaines, mais la spécification du domaine de la fonction étant sans intérêt pour les problèmes abordés dans la suite de l'article, je parlerai simplement de « fonction » dans la suite du texte.

2. Pour une présentation systématique de ces extensions, voir notamment [Pour-El & Richards 1989].

nous ne chercherons pas à formuler de fines distinctions entre ces différentes dénominations. Toutes sont censées exprimer le même concept intuitif, auquel la thèse de Church-Turing confère une extension mathématique précise. Puisque c'est aujourd'hui le terme le plus fréquemment employé dans les communautés mathématique et informatique, et qu'il a l'avantage de se prêter à la formation d'un adjectif, j'emploierai donc uniquement le terme d'« algorithme ». En outre, dans le cadre de notre analyse, il est souhaitable d'éviter de désigner l'objet de la première forme de la thèse de Church-Turing par l'expression « procédure mécanique », puisqu'elle crée une ambiguïté indésirable avec la notion de calcul par une machine, dont on va voir qu'elle forme une notion distincte.

Bien que ce soit cette première forme qui est le plus souvent visée par l'appellation « thèse de Church-Turing », de nombreux auteurs contemporains se réfèrent en réalité à la seconde forme lorsqu'ils discutent de la thèse de Church-Turing (pour un florilège des citations identifiant immédiatement la thèse de Church-Turing sous sa forme empirique, voir [Copeland 2003, 11–13]).

Cette deuxième forme ne doit pas être confondue avec un de ses cas particuliers, la forme physique de la thèse de Church : toute fonction calculable par un système physique est récursive. La forme physique doit être comprise comme un cas particulier de la forme empirique[3], dans la mesure où la machine en question n'est pas forcément décrite en termes physiques : on peut ainsi songer au calcul avec des brins d'ADN. Quant à savoir si une machine biologique pourrait calculer plus de fonctions qu'une machine décrite dans les termes de la physique fondamentale, ou si la physique théorique peut au contraire poser des limites fondamentales au traitement de l'information permettant l'implémentation d'un calcul, c'est une question empirique difficile, qui ne peut être tranchée *a priori*[4]. Il serait donc préférable de parler de « thèse de Church-Turing empirique », et de réserver le qualificatif de « physique » au cas particulier où cette machine est décrite en termes physiques. Cette façon de faire se démarque quelque peu de l'usage courant, où les expressions de « calcul par un système physique », « forme physique de la thèse de Church-Turing » et de « thèse de Church-Turing physique » sont déjà raisonnablement bien implantées. Bien que je ne souhaite pas multiplier *ad libitum* les appellations, il me semble que cet usage introduit une ambiguïté regrettable, et je distinguerai à présent la thèse de Church-Turing empirique de son cas particulier, la thèse de Church-Turing physique, qui s'applique aux machines décrites en termes physiques.

3. Ni la forme physique ni la forme empirique ne doivent être confondues avec la thèse M formulée par R. Gandy (voir [Gandy 1980, 125–126]), qui porte sur les fonctions calculables par un certain type de machines, les *discrete mechanical devices*.

4. La résolution de ce problème pourrait d'ailleurs constituer un nouvel enjeu du débat vénérable autour du réductionnisme en biologie.

Pour éviter toute confusion, je parlerai donc de « thèse de Church-Turing algorithmique », de « thèse de Church-Turing empirique » et de « thèse de Church-Turing physique », ainsi que de forme algorithmique, de forme empirique et de forme physique de la thèse de Church-Turing. On se référera aux fonctions calculables selon ces thèses, respectivement, comme les « fonctions calculables par un algorithme », les « fonctions calculables par une machine », et les « fonctions calculables par un système physique ».

Les converses des deux thèses de Church-Turing nous seront également utiles. La converse de la forme algorithmique — « *toute fonction récursive est calculable par un algorithme* » — est presque universellement admise. La raison en est simple : les fonctions récursives, tout comme les notions extensionnellement équivalentes, comme celles de fonctions calculables par une machine de Turing, par une machine à registres ou de fonctions λ-définissables, ont été introduites pour capturer la notion intuitive de fonction calculable par un algorithme. Il est donc naturel qu'on les considère toutes comme calculables par un algorithme. La converse de la forme empirique — « toute fonction récursive est calculable par une machine » — doit aussi être tenue pour vraie. Les ordinateurs modernes sont des implémentations d'une machine de Turing universelle et ont donc la capacité de calculer l'intégralité des fonctions récursives. Cette capacité est attribuée *de jure,* c'est-à-dire en ignorant les problèmes pragmatiques, bien souvent totalement insurmontables, liés à la complexité en temps et en espace de certaines fonctions. Mais il en va de même pour la notion mathématique de calculabilité, qui ignore également toute limitation contingente liées aux ressources. Ce n'est donc pas un défaut particulier de la notion de « calculabilité par une machine » que d'ignorer ces difficultés pratiques. Je reviendrai sur cette abstraction des difficultés concrètes du calcul, lors de la discussion du statut épistémologique des calculs effectués par des machines (sections 3.1 et 3.2).

1.2 Machines et algorithmes : justification de la distinction

1.2.1 Les machines

À ma connaissance, tous les auteurs ayant pris soin de comparer les deux thèses de Church-Turing ont souligné qu'elles ne doivent pas être confondues (voir [Gandy 1980], [Sieg 2002], [Sieg 2008], [Copeland 2003], [Dowek 2007], [Smith 2007], [Deutsch 1985]). Les deux thèses constituent deux propositions distinctes, mettant en jeu des notions sémantiquement autonomes, celle d'algorithme et celle de machine. Pour bien le comprendre, il nous faut définir plus précisément ces deux concepts.

Il n'est pas possible de donner une définition scientifique générale du concept de « machine de calcul ». Les différentes machines conçues dans la

recherche contemporaine relèvent de disciplines scientifiques hétérogènes et peuvent être décrites en termes physiques, chimiques ou encore biologiques. Face à une telle diversité de stratégies d'implémentation, il est malaisé de formuler une terminologie standard précise, permettant d'embrasser dans son intégralité cette série ouverte de développements scientifiques. Je vais donc me contenter d'une définition intuitive générale, dont la portée sera ensuite précisée par quelques *caveats*.

On désignera ici par 'machine implémentant une fonction', ou plus brièvement 'machine', tout dispositif dans lequel on peut encoder l'information représentée par les arguments, laisser traiter cette information par une série de transformations de l'état du dispositif et récupérer l'information par une opération de lecture, tel que l'ensemble de ces opérations commute avec l'application de la fonction à son argument. Pour que l'appellation de « machine » soit légitime, l'homme doit au plus intervenir dans le traitement de l'information au moment de l'encodage de la donnée et de la lecture du résultat.

L'observateur humain peut parfaitement intervenir pour permettre l'exécution des transformations sur le dispositif, tant qu'il n'a pas accès à l'information encodée, et qu'il ne peut donc en encoder une nouvelle ou en effacer. C'est le cas, par exemple, du calcul avec brins d'ADN, où l'opérateur humain doit intervenir dans le processus de transformation de l'information en opérant des manipulations chimiques, ou de modèles mécaniques rudimentaires, où l'être humain doit fournir de l'énergie au système par l'action d'une manivelle, d'un levier ou d'un poussoir.

Les machines digitales programmables, qui font aujourd'hui partie de notre quotidien, sont bien sûr des machines au titre de notre définition. Mais cette dernière embrasse des cas très différents des machines digitales programmables. Une machine est n'importe quel dispositif permettant d'évaluer au moins une fonction. Elle n'est pas nécessairement une implémentation d'une machine de Turing universelle, et elle n'est pas nécessairement programmable à l'aide d'un langage de programmation moderne. C'est le cas notamment des machines analogiques. Pour le lecteur peu familier de l'histoire de l'informatique, il est bon de rappeler que les machines analogiques sont des machines encodant entrée et sortie dans des paramètres continus d'un système physique, et utilisant l'évolution dynamique du système pour exécuter un certain calcul. Considérons par exemple un circuit électrique simple, composé d'une résistance, d'une bobine et d'une capacité (système oscillant amorti RLC). L'évolution de la tension u aux bornes du condensateur dans le circuit est modélisée par l'équation différentielle $RC.u''(t) + LC.u'(t) + u(t) = 0$. On peut donc utiliser ce circuit pour étudier l'équation différentielle $A.x''(t) + B.x'(t) + C.x(t) = 0$.

Il existe entre ces machines et les machines digitales programmables une différence essentielle pour notre sujet. L'exécution du calcul sur une machine digitale programmable consiste en une manipulation d'un ensemble de données discrètes, qui implémente pas à pas les étapes d'exécution de l'algorithme

dans le langage machine [5]. Dans le cas d'une machine analogique, l'évolution dynamique du système ne peut être vue comme une implémentation des étapes d'un algorithme, bien qu'elle implémente une fonction. Afin de mieux voir ce dernier point, prenons un exemple. Pour des raisons de complexité, on a pu ainsi choisir d'utiliser directement, pour calculer les solutions de l'équation de Navier-Stokes, des systèmes hydrodynamiques gouvernés par cette équation, comme des souffleries (voir [Farge 2007, 11–12]). On peut donc dire qu'une ligne de courant implémente le calcul d'une solution de l'équation de Navier-Stokes. Néanmoins, il serait vain de chercher à distinguer dans une ligne de courant une étape où elle calcule une dérivée partielle par rapport au temps, ou la divergence d'une quantité donnée. Il serait encore plus vain de se demander selon quel algorithme elle effectue ces calculs [6].

En toute rigueur, l'expression même « exécution du calcul » prend un sens différent dans le cas des machines analogiques. « Exécuter un calcul » ne signifie plus « suivre pas à pas les instructions de l'algorithme », mais « laisser le système évoluer à partir de conditions initiales fixées ». Ces deux acceptions du calcul et de son exécution sont si radicalement différentes, qu'on pourrait souhaiter employer des termes différents. Cependant, l'application d'un programme à une entrée donnée et le lancement d'une machine analogique à partir de conditions initiales connues poursuivent la même fin, à savoir l'évaluation d'une fonction en certains de ses arguments, afin d'obtenir des valeurs. Si, pour fixer les idées, on considère la fonction f, on dira que la fin du calcul est de déterminer la valeur de vérité de propositions du type « $f(n) = m$ ». L'exécution d'un algorithme et le lancement d'une machine analogique sont donc deux moyens hétérogènes d'arriver à une même fin et doivent en ce sens être considérés tous deux comme des formes de calcul [7].

1.2.2 Les algorithmes

Il est bien plus difficile de préciser la signification du terme « algorithme ». L'extension du concept de « fonction calculable par un algorithme » est considérée comme fixée par la thèse de Church-Turing algorithmique. Cela ne signifie pas que les machines de Turing constituent une définition de la notion d'algorithme. Depuis le temps des pères fondateurs, les informaticiens ont inventé

5. Pour plus de détails sur cette notion, voir section 1.2.2 plus bas.

6. Pour l'expression d'une position très similaire, voir [Pitowski 1990, 87].

7. On parle parfois de « langage de programmation analogique » et d'« algorithme » ou « programme analogique », mais il ne s'agit là que d'un problème terminologique, qui ne remet pas en cause la distinction que nous venons de faire. Le « langage de programmation » désigne alors le langage dans lequel on va pouvoir décrire la préparation d'une machine analogique, en vue d'exécuter une certaine tâche ou « algorithme analogique ». Pour éviter toute confusion, nous restreindrons l'emploi des termes « algorithme », « programme » et « langage de programmation » à l'examen du calcul symboliquement exécutable, tel que nous l'avons défini.

de nombreuses classes d'algorithmes, aux propriétés très différentes de celle d'un modèle de calcul séquentiel et déterministe comme la machine de Turing : algorithmes parallèles, probabilistes, distribués, interactifs, quantiques... Face à cette prolifération ouverte de la typologie des algorithmes, il n'est pas même certain qu'on puisse parler d'une notion absolue d'algorithme, à laquelle on ajouterait ensuite des qualificatifs sur le mode d'une définition *per genus* et *differentiam proximam*, ou si l'on doit comprendre ce terme comme un terme de famille (voir [Gurevich 2012a], [Gurevich 2012b]). Une définition rigoureuse et générale de la notion d'algorithme n'en est rendue que plus difficile, et au jour d'aujourd'hui, il n'en existe pas de consensuelle, bien que des tentatives existent (voir notamment [Dershowitz & Gurevich 2008]). Si la communauté informatique est tombée d'accord pour fixer l'extension du concept d'algorithme par la thèse de Church-Turing, elle n'est arrivée à aucun consensus quant à son intension.

Cet état de l'art ne va pas sans poser quelques problèmes méthodologiques pour notre présente discussion philosophique. Il peut sembler malaisé de discuter des relations entre les fonctions calculables par des algorithmes et des fonctions calculables par des machines, si l'on ne dispose pas même d'une définition du terme « algorithme ».

Cette difficulté, quoique réelle, ne doit cependant pas être exagérée. Dans le cadre de cet article, nous n'avons pas besoin de formuler une définition générale, même informelle, de la notion d'algorithme, mais seulement de montrer pourquoi les notions de « fonction calculable par un algorithme » et de « fonction calculable par un dispositif empirique » ne sont pas trivialement équivalentes. Notre travail est facilité par le fait que, dans l'état actuel de nos connaissances, tous ces différents modèles de calcul ne calculent pas de fonctions non-récursives. Toutes les fonctions calculables par un algorithme sont donc calculables par un algorithme séquentiel déterministe. Je vais donc me contenter de souligner quelques propriétés bien connues de ces algorithmes, qui seront tout ce dont j'aurais besoin pour formuler ma thèse philosophique.

Avant de procéder, il est nécessaire de distinguer entre algorithmes et programmes. Nombre d'auteurs défendent la nécessité de distinguer les algorithmes, qui sont des objets abstraits, de leur expression dans un langage de programmation particulier, par exemple Donald Knuth :

> An expression of a computational method in a computer language is called a *program.* [Knuth 1997, 5]

L'une des visées d'une bonne définition d'un algorithme serait d'ailleurs de permettre l'identification des algorithmes, au-delà de leurs différentes expressions dans divers langages. Si une telle approche peut faire débat, il est tout à fait consensuel de dire que tout algorithme doit être formalisable dans *quelque* langage de programmation. Cette contrainte de formalisabilité est essentielle à la notion d'algorithme : l'impossibilité d'un « algorithme intuitif », insaisissable par un langage formel de programmation, fait partie de nos intuitions

préthéoriques sur le concept même d'algorithme. On peut donc formuler la contrainte suivante sur la notion d'algorithme :

> *Un algorithme est exprimable par un programme rédigé dans quelque langage de programmation.*

Un langage de programmation L est un langage formel, ou langage symbolique. Un programme rédigé en L est usuellement une suite finie d'instructions de longueur finie, rédigées dans le respect de la syntaxe formelle de L. L'aspect séquentiel de la notation n'est cependant pas essentiel, puisqu'il existe des notations bidimensionnelles, comme le langage flowchart, et autres notations diagrammatiques. J'emploierai par conséquent les termes de « symbole » et de « texte » dans un sens plus large que dans l'usage courant, où ils sont fréquemment réservés pour les systèmes de notation unidimensionnelle : les termes de « symbole », « langage symbolique » et « texte symbolique » désigneront ici tous les types de notations formelles. Un programme est donc, sous ces définitions, un texte formel fini, ou texte symbolique fini [8].

Pour que ce texte symbolique puisse pratiquement compter comme un programme, il faut encore décrire comment l'exécuter, en lui adjoignant une notion d'opérationnalité. On le fera typiquement en définissant une sémantique opérationnelle du langage, ou en concevant un compilateur. Non seulement les algorithmes sont descriptibles par un texte symbolique fini, mais leur exécution pas à pas est descriptible par un texte symbolique. On rejoint ainsi l'intuition commune selon laquelle le calcul est un processus syntaxique [9], consistant en une suite de transformations de suites finies de symboles [10]. Lorsque le calcul

8. La *signature* d'un langage formel est l'ensemble des symboles autorisés dans ce langage. Cet ensemble peut être de cardinalité infinie. Ainsi, lorsqu'on définit un langage, on peut en droit lui attribuer un ensemble infini de noms de variables. Cependant, dans la pratique, tout programme, étant de longueur finie, n'en comprendra qu'un nombre fini.

9. On se restreint ici implicitement au cas où notre algorithme séquentiel déterministe est appliqué à une donnée qui est elle-même représentable par une suite de symboles. Comme le fait remarquer Y. Gurevich [Gurevich 2012a, § 3], un algorithme séquentiel déterministe comme l'algorithme d'Euclide pour trouver le plus grand diviseur commun peut s'appliquer à la fois à des entiers, représentables par des symboles numériques, et des segments de droite, qu'on ne peut représenter ainsi de manière finitaire. De manière générale, les algorithmes séquentiels déterministes « à la règle et au compas » prennent pour entrée des figures et non des symboles. Je rajouterai seulement que sous cette compréhension généreuse du terme « algorithme », la méthode de pliage permettant d'obtenir un origami donné compte également comme un algorithme séquentiel déterministe. De telles constructions doivent-elles cependant comptées comme calcul ? Ne souhaitant pas m'engager dans ce débat épineux, je limiterai mes considérations aux algorithmes séquentiels déterministes dont les données peuvent être représentées par des symboles.

10. Si le langage de programmation est muni d'une sémantique opérationnelle bien définie, l'exécution peut être effectuée, à l'aide d'un interpréteur, dans la syntaxe du langage lui-même. Dans le cas contraire, le langage doit être compilé dans le langage machine pour être exécuté.

termine, ce texte symbolique est fini ; sinon, il est en droit infini. Qu'elle soit ou non infinie, l'exécution peut être décrite étape par étape par une manipulation de symboles [11].

Pour fixer les idées, imaginons que nous cherchions à trouver le plus grand commun diviseur de deux entiers, en utilisant l'algorithme d'Euclide. Nous devrons d'abord lire l'algorithme afin d'évaluer s'il est correct, soit s'il effectue bien la tâche qu'il est censé effectuer. Puis nous devrons rédiger l'exécution du calcul sur certaines données, par exemple les entiers '119' et '544'. La lecture et la compréhension de l'algorithme, et de son exécution symbolique sur une entrée suffisent aux mathématiciens et informaticiens pour savoir qu'ils ont atteint un résultat correct, exprimé par la proposition : « Le plus grand diviseur commun de '119' et '544' est '17'. »

On dira donc d'un algorithme séquentiel déterministe qu'il est *symboliquement descriptible* et *symboliquement exécutable*. Je résumerai ces deux propriétés en disant que le calcul effectué en suivant un tel algorithme est un calcul symbolique. L'exécution symbolique du calcul évoquera spontanément le « calcul papier-crayon », pour reprendre l'expression employée par les informaticiens. On pourrait donc objecter que lorsqu'un calcul est exécuté sur un ordinateur moderne, l'exécution du calcul ne consiste nullement en la manipulation de symboles, mais en celle de signaux électroniques. Cependant, ces signaux électroniques encodent des suites de bits à traiter par le processeur, que les informaticiens désignent par le nom évocateur de « langage machine ». On pourrait donc effectuer en droit le calcul par une manipulation symbolique en notation binaire, même si, bien entendu, aucun être humain n'y parviendrait dans les cas de complexité non triviaux. L'exécution par une machine d'un programme donné dans son langage machine compte donc également comme calcul symbolique.

Puisque toute fonction calculable par un algorithme est calculable par un algorithme séquentiel déterministe, on peut donc conclure que les fonctions calculables par un algorithme sont les fonctions calculables par un calcul symbolique. Cela suffit à les distinguer sémantiquement des fonctions calculables par

11. En toute rigueur, on peut souhaiter distinguer entre la lecture du programme, conçu comme un objet textuel, et la compréhension de l'algorithme, conçu comme un objet abstrait. On pourrait également réserver la notion d'exécution au programme, comme cela correspondrait mieux à l'emploi courant dans la communauté informatique. Néanmoins, lorsqu'on parlera du calcul papier-crayon, il serait maladroit de dire qu'on exécute un programme pour l'addition, alors que l'immense majorité des personnes réalisant des additions à la main n'a jamais lu un tel programme. De même, il serait maladroit de parler de lecture de programme lorsqu'on présente un algorithme en pseudo-code ou de manière informelle. Je parlerai donc, dans la suite de cet article, de lecture, compréhension et exécution d'un algorithme, pensant que le gain en légèreté de l'expression sera supérieur à la perte en rigueur, qui est négligeable pour notre présent sujet.

une machine, puisqu'on a vu que les machines analogiques pouvaient évaluer une fonction en certains de ses arguments sans exécuter un calcul symbolique.

2 La thèse A et sa pertinence philosophique

2.1 Énoncé de la thèse A

En prenant acte de la distinction entre la forme algorithmique et la forme empirique de la thèse de Church-Turing, on peut poser deux nouvelles questions : « *Toute fonction calculable par un algorithme est-elle calculable par une machine* ? » et « *Toute fonction calculable par une machine est-elle calculable par un algorithme* ? » Pour nombre d'auteurs [12], une réponse affirmative à la première question fait partie de nos intuitions fondamentales sur la notion d'algorithme. Selon cette vision courante, l'exécution d'un algorithme est un processus de pure manipulation symbolique, ne nécessitant aucune ingéniosité et aucune intuition sémantique. Il pourrait donc, à tout le moins en droit, être réalisé par un dispositif machinique ne disposant nullement de l'intuition humaine. On peut même voir la possibilité d'être exécuté par une machine comme un *critère* permettant de reconnaître qu'un algorithme est bien explicité. Un algorithme étant censé définir une procédure parfaitement explicite pour l'exécution d'un calcul, la possibilité de l'implémenter sur une machine montrerait qu'on a bien atteint cet objectif et qu'aucune référence implicite à notre intuition ne s'est glissée dans notre travail :

> (...) an algorithm must be specified to such a degree that even a computer can follow the directions. [Knuth 1997, 6]

Une réponse affirmative à la seconde question constitue la thèse A : « *Toute fonction calculable par une machine est calculable par un algorithme.* » Celle-ci ne constitue en aucun cas une intuition commune sur la notion d'algorithme, ou de machine. Notre conception d'un algorithme ne définit aucune limite évidente sur les capacités des machines. Les fonctions calculables par un algorithme sont calculables par un calcul symboliquement descriptible et symboliquement exécutable : aucune mention n'est faite de processus empiriques dans l'énoncé de ces propriétés. Comme le signale à juste titre David Deutsch, il n'y a aucune raison *a priori* pour que l'ensemble des fonctions exprimables par un algorithme soit exactement le même que celui calculable par une procédure empirique :

12. On peut trouver un exemple d'une telle position chez Rosser : "An effective method of solving certain sets of problems exists if one can build a machine which will then solve any problem of the set with no human intervention beyond inserting the question and (later) reading the answer" [Rosser 1939]. Comme nous l'avons déjà souligné ci-dessus, la dernière condition donnée est trop stricte : le sujet humain peut intervenir à d'autres étapes que l'insertion de la question et la lecture de la réponse, tant qu'il n'a pas accès à l'information encodée dans le dispositif.

> (. . .) There is no *a priori* reason why physical laws should respect the limitations of the mathematical processes we call 'algorithms' (. . .). [Deutsch 1985, 4]

Ensuite, comme nous l'avons vu précédemment, il existe des dispositifs empiriques, comme les machines analogiques, qui permettent d'implémenter le calcul d'une fonction sans implémenter un algorithme. Il n'est donc pas évident d'expliciter les raisons pour lesquelles ces dispositifs, qui n'implémentent pas une machine de Turing universelle, ne pourraient pas calculer une fonction qui n'est pas calculable par une machine de Turing ou un modèle équivalent.

Si les notions de « calcul effectué selon un algorithme » et de « calcul effectué par une machine » sont sémantiquement distinctes, cela n'exclut nullement qu'elles soient extensionnellement équivalentes. À la lumière des arguments que nous venons de présenter, il ne semble cependant pas *a priori* évident de montrer que ce soit le cas. Autrement dit, la thèse A, si elle peut sembler plausible, n'est ni une simple remarque sémantique, ni une trivialité.

Il pourrait être tentant de qualifier d'hypercalcul la possibilité ici envisagée. On désigne usuellement comme hypercalcul tout modèle de calcul permettant le calcul d'une fonction non-récursive. Une telle possibilité théorique souffre d'une certaine ambiguïté par rapport à la thèse A. Si la forme *algorithmique* de la thèse de Church-Turing est vraie, une machine qui calculerait une fonction non-récursive calculerait une fonction incalculable par un algorithme et invaliderait la thèse A. Si la forme algorithmique est fausse, il ne serait pas certain qu'un tel modèle de calcul contredirait la thèse A. Pour bien le comprendre, imaginons qu'un physicien de génie réussisse à concevoir une machine permettant de calculer une fonction non-récursive. Cela n'impliquerait pas nécessairement que la thèse A soit fausse. Il se pourrait que, quelques mois plus tard, quelque informaticien de génie conçoive un langage de programmation permettant l'expression de fonctions non-récursives. Dans un tel cas de figure, la thèse A demeurerait correcte, bien que les deux formes de la thèse de Church-Turing soient fausses. La question de la correction de la thèse A peut donc être posée, indépendamment de celles de ces deux thèses. Il est donc préférable de ne pas confondre la possibilité d'une fonction implémentable sans être calculable par un algorithme avec celle de l'hypercalcul. Si j'ai formulé la thèse A à partir d'une discussion des deux formes de la thèse de Church-Turing, il est important de souligner que la question *philosophique* soulevée par cette thèse est bien formulée, indépendamment de toute hypothèse sur la vérité de ces deux propositions.

En revanche, d'un point de vue *scientifique,* il serait vain de réfléchir sur la thèse A sans admettre la thèse de Church-Turing algorithmique. Que l'on admette ou non les démonstrations proposées de la thèse de Church-Turing algorithmique, force est de reconnaître que nous n'avons à ce jour aucun indice que la thèse de Church-Turing algorithmique puisse être fausse, et aucune idée de ce à quoi pourrait ressembler un langage de programmation permettant

d'exprimer des fonctions non-récursives. On est donc stratégiquement contraint de raisonner *modulo* la thèse de Church-Turing algorithmique, si l'on souhaite raisonner scientifiquement sur les limites de la calculabilité. Certains auteurs essayent ainsi de démontrer la forme physique de la thèse de Church-Turing à partir de principes physiques raisonnables, tandis que d'autres tâchent de montrer qu'une machine calculant une fonction non-récursive est une impossibilité méthodologique (voir [Gandy 1980], [Sieg 2002], [Sieg 2008], [Arrighi & Dowek 2011] pour la première approche, et [Davis 2004], pour la seconde). Je ne me prononcerai pas ici sur ces travaux scientifiques. L'argument philosophique que je cherche à développer ici ne dépend pas même de cette hypothèse et pourrait s'accommoder de l'existence d'un langage de programmation plus expressif que tous les langages existant à ce jour.

2.2 Langages formels et description de la nature

Admettant que la thèse A constitue une proposition bien définie, je vais tâcher à présent d'en montrer la pertinence philosophique. La thèse A peut être conçue comme une thèse épistémologique, portant sur la capacité de nos langages de programmation à simuler certains processus naturels.

La thèse A affirme en effet la capacité de nos langages formels à simuler toutes les procédures de manipulation *empirique* de l'information permettant de déterminer la valeur d'une fonction en des arguments. Pour le comprendre, imaginons que la thèse A soit fausse. Un physicien pourrait alors construire une machine permettant le calcul d'une fonction, qu'aucun algorithme ne nous permettrait d'évaluer. Pour fixer les idées, admettons la vérité de la thèse de Church-Turing algorithmique et imaginons que cette machine puisse calculer une fonction non-Turing calculable, comme la fonction d'arrêt. Par le biais de cette machine, nous pourrions connaître la valeur de la fonction d'arrêt en ses arguments. Aucun algorithme ne décrirait la procédure mathématique menant à ce résultat, puisque l'indécidabilité du problème de l'arrêt, joint à la thèse de Church-Turing algorithmique, prohibe l'existence d'un tel algorithme. Par conséquent, aucun texte formel ne décrirait non plus l'exécution pas à pas de ce calcul, tout comme dans le cas des machines analogiques. Mais à la différence de ces dernières, aucun algorithme, dans aucun langage de programmation, ne permettrait de décrire le calcul effectué et de simuler ainsi le fonctionnement de cette machine.

Une telle éventualité n'est nullement interdite par l'indécidabilité du problème de l'arrêt, ou par un autre résultat similaire d'indécidabilité. Ceux-ci reposent sur la vérité de la thèse de Church-Turing algorithmique, tout d'abord, et sont fondés sur des raisonnements portant uniquement sur les algorithmes, et non sur des machines physiques, ensuite. L'inexistence d'un algorithme calculant une certaine fonction n'implique pas l'inexistence d'une machine pouvant calculer cette fonction, sauf si précisément on admet la thèse A.

Rappelons encore une fois qu'une machine de calcul, d'après notre définition, n'implémente pas nécessairement les étapes d'exécution d'un algorithme. Comment se fait-il donc qu'à chaque fois qu'on a constaté qu'un phénomène naturel suivait une certaine fonction, cette fonction était calculable par un certain algorithme ? Ceci semble attribuer à nos langages formels, au sein desquels sont décrits et exécutés les algorithmes, un pouvoir de description des évolutions naturelles qui n'est absolument pas évident. Comme nous venons de le voir, ceci peut être interprété d'un point de vue méthodologique : lorsque nous prenons la décision théorique de modéliser un phénomène naturel par un certain objet mathématique, nous faisons en sorte d'employer des mathématiques calculables, afin d'obtenir ce pouvoir de description. On peut *a contrario* l'interpréter comme un authentique principe de la physique, comme l'ont défendu nombre d'auteurs, en particulier David Deutsch [Deutsch 1985], [Deutsch 1997], G. Dowek [Dowek 2007, chap. 5] et J. Barrow [Barrow 1992, 86–87]).

Sans me prononcer sur cette question complexe, je vais à présent tâcher de développer les conséquences de cette question pour l'épistémologie du calcul, en particulier pour l'idée que le calcul constitue une connaissance *a priori* (section 4). Mais avant d'en arriver à ce problème précis, il nous faut poser quelques jalons préliminaires sur l'épistémologie du calcul, en définissant les critères de correction d'un calcul en fonction des moyens employés (section 3.1), les relations de simulabilité entre ces différents modèles (section 3.2) et les relations de la thèse A à ces deux questions (section 3.3).

3 Remarques sur l'épistémologie du calcul

3.1 Les critères de correction du calcul

Comment sait-on qu'un calcul est correct ? Avant de commencer à répondre à cette question, il nous faut distinguer entre une méthode mathématique pour résoudre un problème de calcul et un moyen permettant d'effectuer ce calcul, ou moyen de calcul. Un algorithme est une méthode mathématique permettant de résoudre uniformément toutes les instances d'un problème. C'est un objet idéal, qui peut s'appliquer à l'ensemble de toutes les instances d'un problème décidable donné, y compris lorsque cet ensemble est de cardinalité infinie. Mais pour connaître effectivement les valeurs d'une fonction, lorsqu'elle est évaluée sur une entrée donnée, il faut encore disposer d'un moyen par lequel on puisse exécuter concrètement le calcul. Le calcul papier-crayon, les machines digitales programmables, les machines analogiques et les hypothétiques machines violant la thèse A constituent tous des moyens pour évaluer la fonction et obtenir des résultats de calcul ou valeurs. Certains procèdent par l'exécution d'un algorithme, et d'autres non.

Si différents qu'ils puissent être les uns des autres, ces moyens se distinguent fondamentalement en nature de la méthode mathématique représentée par l'al-

gorithme, en ceci qu'ils sont des objets concrets, soumis à des contraintes de ressources. Nous disposons par exemple d'algorithmes simples nous permettant de calculer la somme de deux nombres entiers. Le calcul papier-crayon constitue un moyen pour exécuter cet algorithme sur des entrées particulières, qui s'applique en droit à une entrée de taille arbitraire. Cependant, dans les faits, on sera incapable d'exécuter le calcul sur des entrées de taille 10^{250}, puisqu'il s'agit d'un chiffre supérieur au nombre de particules dans l'univers. Lorsqu'on considère un moyen pour effectuer un calcul, on doit donc distinguer entre ce que ce moyen nous permet de faire *de jure,* et ce qu'il nous permet de faire *de facto.* Cette distinction n'aurait pas de sens pour la méthode, qui est d'emblée un objet idéal, échappant aux contraintes concrètes de ressources.

Comme on va le voir immédiatement, la question de la correction du calcul se pose à la fois au niveau de la méthode et au niveau des moyens. Cependant, la réponse variera essentiellement en fonction des moyens de calcul employés. Je vais donc distinguer trois cas : le calcul papier-crayon, les machines digitales programmables et les machines analogiques.

Dans les deux premiers cas, deux questions indépendantes doivent être résolues pour montrer que le résultat d'un calcul est correct :

- *l'algorithme employé est-il correct ?* On s'assure par là que la méthode de calcul employée résout bien le problème considéré.
- *l'algorithme est-il correctement exécuté ?* Une fois convaincu de la correction de son algorithme, il faut encore s'assurer de la fidélité de son exécution pas à pas sur l'entrée considérée [13].

Si les questions posées demeurent les mêmes pour le calcul papier-crayon et les machines digitales programmables, les moyens d'y répondre varient. Cela est manifeste dans le cas du contrôle de l'exécution. L'exécution d'un calcul papier-crayon est vérifiée par un être humain qui doit s'assurer pas à pas d'avoir effectué les bonnes manipulations de symboles. En revanche, comme on l'a vu plus haut [14], l'exécution d'un calcul sur un ordinateur n'est pas constituée dans les faits par une série de transformations sur des symboles, mais par la manipulation de signaux électroniques. Aucun texte formel n'est rédigé ou lu par un mathématicien, pour s'assurer que le résultat obtenu par la machine digitale est correct. La correction de l'exécution du calcul dépend de la fiabilité de notre machine. Celle-ci est appuyée sur les principes physiques régissant notre modèle physique de calcul, sur l'ingénierie du modèle de machine que nous employons et sur les opérations de maintenance et de contrôle qui nous garantissent que la machine singulière, qui se trouve devant nous, effectue bien correctement le calcul que nous lui avons demandé d'exécuter. En termes simples, ces considérations garantissent que notre ordinateur fonctionne bien.

13. Ceci a déjà été remarqué par I. Pitowski dans [Pitowski 1990, 85].
14. Voir section 1.2.2.

La première question peut, quant à elle, être traitée de la même manière dans le cas du calcul papier-crayon [15] et du calcul sur machine digitale programmable. Une partie respectable de l'énergie des informaticiens est dédiée à la conception de techniques mathématiques permettant de prouver la correction de programmes de plus en plus longs et sophistiqués [16]. L'informatique permet cependant l'emploi de techniques autres que la rédaction d'une démonstration sur le papier pour vérifier la correction d'un programme. On peut ainsi remarquer l'emploi par ces mêmes informaticiens de moyens automatisés pour vérifier certaines propriétés de leurs programmes, comme les logiciels contrôlant la correction de la syntaxe ou du typage d'un programme, voire pour prouver la correction du programme. La fiabilité des réponses données par ces logiciels dépend alors de la correction de *leur* programme et du bon fonctionnement de notre ordinateur, en sorte que la correction de notre calcul en vient à dépendre de la correction d'un *autre* calcul.

Cette situation n'est pas en elle-même totalement inédite. Dans la pratique du papier-crayon, on peut ainsi employer de nombreuses techniques de tests partiels pour vérifier la correction d'un résultat, comme, par exemple, la vérification de l'homogénéité d'une équation. La correction d'un calcul est alors vérifiée par la réalisation d'un autre calcul, plus simple, dont on doit encore se demander s'il a été bien exécuté et si l'algorithme employé est correct. L'informatique moderne a surtout innové en automatisant ces tâches et en les appliquant au niveau de la vérification des algorithmes eux-mêmes. Dans le cas des machines digitales programmables, la question de la correction de l'algorithme employé peut donc elle aussi dépendre du bon fonctionnement de notre ordinateur. Cette dépendance est cependant une dépendance de fait, tandis qu'elle est une dépendance de droit dans le cas de l'exécution du calcul. La vérification de la correction d'un algorithme peut donc dépendre du bon fonctionnement de notre ordinateur, tandis que la correction de l'exécution sur une machine digitale programmable est une question portant sur ce bon fonctionnement.

La situation est totalement différente dans le troisième cas, à savoir celui des machines analogiques. Comme on l'a vu ci-dessus, leurs principes de fonctionnement sont différents de ceux d'une machine digitale programmable, et

15. Notre propos est bien évidemment fortement idéalisé et vise à l'examen des considérations théoriques de droit, et non à une description de la pratique de fait. Dans la pratique ordinaire du calcul papier-crayon, on emploie nombre d'algorithmes sans jamais avoir vérifié leur correction, et même sans s'être jamais posé la question. D'un point de vue historique, c'est l'essor de l'informatique qui a engendré l'étude systématique de cette question de la correction des algorithmes.

16. Les questions du sens et du statut de ces preuves de correction, par comparaison avec les preuves mathématiques usuelles, sont des enjeux centraux de la philosophie de l'informatique. Ne pouvant aborder ici ces problèmes difficiles, je considérerai que, pour les questions soulevées dans cet article, on peut assimiler les preuves de correction aux preuves mathématiques usuelles.

les critères de correction d'un calcul donné sur une machine analogique sont donc différents. Puisqu'aucun algorithme n'est exécuté, il n'y a ni à démontrer qu'un algorithme est correct, ni à s'assurer que celui-ci est bien exécuté par notre machine. Pour s'assurer que le résultat donné par notre machine effectué est correct, il faut pouvoir garantir que a) l'évolution dynamique de notre système permet bien d'implémenter la fonction désirée ; b) la préparation du système et sa mesure permettent d'encoder et de lire entrées et sorties avec suffisamment de précision. Ces critères relèvent à la fois de la théorie physique, qui décrit la dynamique de notre système et de l'ingénierie de notre dispositif particulier, qui assure que nous avons un contrôle suffisant de notre dispositif. Enfin, il faut effectuer un ensemble d'opérations de maintenance et de contrôle, afin de s'assurer que l'exemplaire particulier de machine qui se trouve devant nous n'est pas victime d'un vice de fabrication ou d'une perturbation quelconque.

Supposons à présent que nous disposions d'une machine censée calculer une fonction qui ne soit pas calculable par un algorithme. Comment saurions-nous qu'un calcul effectué par cette machine est correct ? Nos raisons de croire que notre machine calcule bien une fonction donnée seraient à première vue similaires à celles employées pour une machine analogique. Nous devrions nous assurer que la dynamique du système considéré permet bien d'évaluer la fonction voulue, que notre contrôle de la préparation et de la mesure de notre système permet d'encoder les entrées et de lire les sorties avec la précision requise [17] et que le dispositif situé en face de nous n'est pas victime d'un défaut ou d'une perturbation particulière.

3.2 Relations de simulabilité entre ces différents modèles

Quelles sont les relations entre ces différents moyens de calcul, lorsqu'il s'agit d'examiner les limites de la calculabilité ? Les machines de calcul, qu'elles soient analogiques ou digitales, nous permettent de calculer des fonctions que nous sommes strictement incapables de calculer mentalement ou par un simple calcul papier-crayon : c'est ce qui justifie leur mise au point. Cette incapacité est cependant de nature *pratique.* Elle est due, tout d'abord, à la faible fiabilité des calculateurs humains, qui sont fortement sujets à commettre des erreurs, même sur des calculs de quelques pages. La fiabilité d'un calcul extrêmement long serait donc quasi nulle. Elle est ensuite due, comme nous l'avons déjà remarqué, aux difficultés liées aux ressources, qui rendraient pratiquement irréalisables certains calculs. Même si l'on disposait d'un vaste groupe

17. Parmi les critiques adressées aux modèles censés violer la thèse de Church-Turing physique, on peut d'ailleurs trouver une critique d'ordre pratique, selon laquelle certains des modèles envisagés exigeraient une précision d'encodage et de mesure totalement irréaliste (voir notamment [Hodges 2005]).

de calculateurs humains infaillibles, le calcul papier-crayon est trop lent et trop gourmand en espace pour permettre l'exécution de certains calculs aux coûts très élevés en ces deux ressources.

Cependant, aucun de ces modèles de calcul ne permet de calculer une fonction qu'il serait *en droit* impossible de calculer par un calcul papier-crayon. Nos machines digitales programmables sont des implémentations de machines de Turing universelles. Elles sont donc *en droit* simulables par un calcul papier-crayon : il suffirait de faire exécuter la table d'instructions par un humain écrivant et effaçant entrées et sorties sur une bande de papier. Nous *pourrions* donc effectuer tous les calculs effectués par nos machines digitales, en ignorant les difficultés pratiques que nous venons de mentionner. Pour les machines digitales programmables, la forme physique de la thèse de Church-Turing est donc trivialement vraie, et donc la thèse A est vraie. Dans le cas des machines analogiques, aucun argument théorique général ne garantit à l'heure actuelle qu'elles ne puissent exécuter un calcul qui ne soit pas exécutable par une machine de Turing, même s'il existe des résultats en ce sens (pour une présentation des modèles analogiques, voir [Bournez & Campagnolo 2008], [Bournez, Dershowitz & Falkovich 2012]). Il n'est donc pas évident que la forme physique de la thèse de Church-Turing soit vraie pour ces machines, et par conséquent il n'est pas évident que la thèse A s'applique à ces machines. Cependant, d'un point de vue pratique, on n'a jamais implémenté avec ces machines de fonctions qui ne soient pas Turing-calculables. Dans les faits, les machines analogiques sont donc elles aussi simulables par un calcul papier-crayon.

La relation de simulabilité envisagée dans ce dernier cas est cependant différente de celle prévalant pour les machines digitales programmables. Il est possible, en suivant le même algorithme que celui exécuté par la machine, d'opérer une simulation pas à pas du calcul effectué par une machine digitale programmable. Pour une machine analogique, il n'est pas possible de définir une notion équivalente de pas de calcul : la simulation porte donc uniquement sur le comportement entrées-sorties de la fonction. Cette distinction n'affectera pas la suite de notre discussion, et je continuerai à parler simplement de « simulation », sans distinguer entre la « simulation pas à pas » et la « simulation entrées-sorties [18] ».

La thèse A garantit donc que tous les calculs effectués sur machine demeurent *de jure* simulables par un calcul papier-crayon. Il convient de préciser le sens de cette dernière remarque. L'inexistence d'un algorithme pour résoudre un problème n'implique pas qu'on ne puisse pas résoudre des instances de ce problème. L'inexistence d'un algorithme signifie que nous ne disposons pas de méthode générale pour résoudre un problème donné : elle ne signifie pas que

18. La notion de simulation pas à pas est une notion complexe, notamment à cause de la grande difficulté qu'il y a à définir ce qu'est un pas de calcul élémentaire. Je ne rentrerai pas ici dans ces difficultés, puisqu'elles n'affectent pas la distinction faite usuellement entre simulation pas à pas et simulation entrées-sorties.

nous ne puissions résoudre des instances de ce problème, par d'autres biais. Ainsi, nous pouvons connaître de nombreuses valeurs de la fonction d'arrêt, bien que le problème de l'arrêt soit indécidable [19]. À chaque fois qu'un algorithme, codé par l'entier n, s'arrête sur une entrée donnée m, nous connaissons la valeur correspondante de la fonction d'arrêt : $h(n, m) = 1$. Une telle approche ne nous permet que de connaître un nombre fini, quoique non borné, de valeurs de la fonction d'arrêt. En revanche, en démontrant qu'une fonction est totale [20], on peut déduire une infinité de valeurs de la fonction d'arrêt. Par exemple, si n est le code de la fonction successeur, on sait que, pour tout entier m, $h(n, m) = 1$. En démontrant qu'une fonction n'est pas définie en un de ses arguments, on peut en déduire un ensemble de valeurs de la fonction d'arrêt pour tous les programmes calculant cette fonction. Ainsi, si l'entier n' est le code d'un programme calculant une fonction f et qu'on peut démontrer que f n'est pas définie en un de ses arguments m', alors on sait que $h(n', m') = 0$. Chacune des approches que nous venons de décrire constitue une méthode pour résoudre une classe d'instances du problème de l'arrêt. On peut les qualifier de méthodes particulières, en ceci qu'elles ne permettent pas de résoudre le problème dans le cas général, par opposition à la méthode générale qu'aurait constitué un algorithme résolvant le problème de l'arrêt.

La démonstration de l'indécidabilité d'un problème interdit de trouver une méthode permettant de résoudre toutes les instances d'un problème : elle n'empêche pas de connaître la solution du problème pour un nombre indéfini d'instances. Elle n'implique pas non plus qu'il existerait des instances de ce problème qu'on ne saurait résoudre par aucune méthode particulière [21]. En d'autres termes, l'inexistence d'un algorithme pour calculer une fonction donnée n'interdit pas de connaître des valeurs de cette fonction, et elle n'interdit pas même de connaître une valeur donnée de cette fonction : elle rend seulement la recherche de ces valeurs beaucoup plus difficile.

Il serait donc en droit possible, bien que pratiquement très difficile, de vérifier que des valeurs données par la machine violant la thèse A sont correctes. Mais aucun algorithme ne pourrait résoudre toutes les instances que notre modèle résoudrait. Celui-ci représenterait donc un surcroît d'efficacité, ce qui est précisément ce qu'on attend d'une machine : mais ce surcroît d'efficacité ne se traduirait pas seulement par un gain éventuel de temps, d'espace et de fiabilité, mais aussi par un gain en généralité.

Nous disposerions alors d'un moyen de calcul permettant de connaître plus de valeurs d'une fonction qu'aucune méthode mathématique ne le per-

19. Ceci a déjà été remarqué par J. Copeland dans [Copeland 2004, 256].

20. Une fonction f d'un ensemble E dans un ensemble F est totale ssi f est partout définie sur E. Si le domaine de définition de f est un sous-ensemble propre de E, la fonction est partielle.

21. Soient f une fonction, p, p' des programmes. On note $exec(p, n)$ l'exécution du programme p sur une entrée n. $\nexists p \forall n((f(n) = y) \leftrightarrow (exec(p, n) = y))$ n'implique pas $\exists n \nexists p'((f(n) = y) \leftrightarrow (exec(p', n) = y))$.

mettrait. Cette situation serait paradoxale, dans la mesure où, en considérant nos moyens présents, on penserait plutôt qu'un moyen de calcul permettrait au plus, si l'on ignorait les contingences liées aux problèmes de ressources, de nous donner accès à toutes les valeurs de la fonction que l'algorithme nous permettrait de calculer. Nous reviendrons sur ce point essentiel, lorsque nous conclurons la partie suivante de nos réflexions sur la notion d'*a priori*.

4 La thèse A et le statut *a priori* du calcul

4.1 Le calcul papier-crayon comme connaissance *a priori*

La pertinence de la notion philosophique d'*a priori* en général, et en particulier en philosophie des mathématiques, a été l'objet d'intenses débats dans la littérature récente (voir l'ouvrage collectif [Boghossian & Peacocke 2000]). Certains auteurs, dont la plupart se placent dans la continuité de la critique de W. V. O. Quine, vont jusqu'à dénier toute pertinence à cette notion, en général ou en particulier en philosophie des mathématiques, tandis que d'autres ont choisi de se faire les avocats de ce concept vénérable (voir respectivement les contributions de Philip Kitcher et Penelope Maddy, d'une part, et Paul Boghossian, d'autre part). La simple exposition des différentes positions en présence, sans même parler de leur discussion, dépasse de loin la portée de cet article. Sans prendre position au sein de ce vaste débat, je vais me contenter de montrer ce que l'examen de la thèse A peut apporter à la discussion.

Tout d'abord, il faut distinguer deux définitions de la notion d'*a priori*. La première, ou définition *épistémique*, stipule qu'une proposition est *a priori* (respectivement, a posteriori, ou empirique) si et seulement si la connaissance de sa valeur de vérité est justifiée par des considérations indépendantes de l'expérience (respectivement, dépendantes de l'expérience). La seconde, ou définition *métaphysique*, stipule qu'une proposition est *a priori* (respectivement, a posteriori, ou empirique) si et seulement si sa valeur de vérité ne dépend pas de considérations empiriques (respectivement, dépend de considérations empiriques). Cette seconde définition est qualifiée de métaphysique, dans la mesure où elle ne prétend pas prendre en compte les conditions particulières sous lesquelles nous parvenons à la connaissance d'une vérité, mais ce qui la rend objectivement vraie, indépendamment de la connaissance que nous en avons.

Ces définitions courantes ne sont pas sans quelque imprécision. Quelle est la nature exacte de cette indépendance à l'égard de l'expérience ? Sans prétendre résoudre cette question difficile, on peut tenter de préciser quelque peu la définition de la manière suivante : une proposition est *a priori* si et seulement si on ne peut réviser sa valeur de vérité suite à de nouvelles expériences. Une

proposition est au contraire empirique ou *a posteriori* si et seulement si sa valeur de vérité peut être révisée suite à de nouvelles expériences. Je n'entends pas substituer cette définition à la définition plus générale donnée auparavant, mais je la conçois simplement comme un cas particulier de cette dernière, jouïssant d'une précision légèrement supérieure, dont je ferai un usage ponctuel dans le cours de mon analyse.

Qu'est-ce qui est censé compter comme expérience ? Par exemple, les sensations internes d'un sujet, établies par l'introspection, doivent-elles compter comme expérience (voir [Boghossian & Peacocke 2000, 2–3]) ? L'ambition de donner une définition à la fois rigoureuse, générale et pertinente de la notion d'*a priori* va bien au-delà des objectifs de notre analyse présente. Je vais me contenter d'une compréhension minimale de cette notion, qui permet de comprendre la vision historique courante selon laquelle les propositions mathématiques en général, et celles portant sur le calcul en particulier, sont *a priori*. À cette fin, je me servirai uniquement de la conception épistémique de l'*a priori*, sans aborder la question complexe de son rapport à la conception métaphysique.

La connaissance obtenue par le calcul est-elle *a priori* ? Pour répondre à cette question, il est nécessaire d'appliquer le concept d'*a priori* aux méthodes et aux moyens de calcul. Une méthode peut être qualifiée d'*a priori*, dans la mesure où les considérations établissant sa correction sont *a priori*. Un moyen de calcul est *a priori*, dans la mesure où les considérations établissant la correction de l'exécution du calcul sont *a priori*. Je dirai enfin que la connaissance obtenue par un moyen de calcul est *a priori*, si la valeur de vérité des propositions exprimant les résultats de calculs obtenus par ce moyen particulier, de type « $f(n) = m$ » ne dépend que de considérations *a priori*. Lorsque je parlerai de manière générale des différents moyens de calcul, je parlerai simplement de connaissance obtenue par le calcul, en partant de l'idée élémentaire que le but du calcul est de connaître les valeurs d'une fonction en certains de ses arguments.

Dans le cas du calcul papier-crayon, la correction d'un calcul dépend de la démonstration de correction de l'algorithme et de la vérification pas à pas de l'exécution de cet algorithme. Ces deux étapes sont réalisées par un sujet calculateur, qui vérifie la correction de ses calculs et se convainc de la justesse de la méthode employée. Toutes ces considérations seront usuellement qualifiées d'*a priori*. La connaissance obtenue par le calcul papier-crayon semble donc un candidat naturel, au titre de connaissance *a priori*, si seulement une connaissance doit mériter ce titre.

Si l'on doit préciser quelque peu l'intuition sous-jacente à notre dernière affirmation, on peut faire la remarque suivante. Lorsqu'il établit un résultat de calcul, le mathématicien idéal procède à une lecture de textes formels : le texte décrivant l'algorithme, le texte démontrant la correction de cet algorithme et le texte détaillant l'exécution de cet algorithme sur une entrée

donnée. Le mathématicien qui comprend ces textes dispose de l'intégralité des considérations permettant de reconnaître la vérité de la proposition discutée, de la forme « $f(n) = m$ ». Il n'a nullement besoin de sortir de son bureau pour observer les étoiles, le vol des oiseaux migratoires ou la croissance des populations de ratons-laveurs, pas plus qu'il n'a besoin de s'enfermer dans un laboratoire pour y réaliser des expériences. Une fois convaincu d'avoir déterminé la valeur de vérité de cette proposition, le mathématicien pourrait être amené à la réviser suite à la découverte d'une faute de raisonnement ou de calcul, mais nulle expérience nouvelle ne pourra l'amener à opérer une telle révision : les propositions exprimant les résultats de calcul sont donc *a priori* au second sens que nous avons défini.

Par contraste, la lecture et la compréhension du texte constituant un article de biologie, par exemple, ne suffisent nullement à justifier la vérité des propositions qui y sont défendues. Il faut encore que les résultats d'expériences et d'observations qui y sont rapportés soient corrects, ce qui, pour être établi, exige d'autres activités que la simple lecture et compréhension de l'article, notamment la reproduction des résultats expérimentaux. On peut exprimer l'intuition à la base de la croyance que les résultats de calcul constituent une connaissance *a priori* de la manière suivante : *les propositions exprimant les résultats de calcul seront qualifiées de propositions* a priori, *dans la mesure où la connaissance de leur valeur de vérité est justifiée par la simple lecture et compréhension de textes formels finis.* Cette conception se généralise immédiatement aux théorèmes, puisque les démonstrations sont également des textes formels finis [22]. En reprenant le vocabulaire défini à la section 1.2.2, on peut donc dire que les propositions exprimant un résultat de calcul sont *a priori* parce que le calcul effectué selon un algorithme est symboliquement descriptible et symboliquement exécutable.

Les considérations du paragraphe précédent ne visent nullement à défendre la conception aprioriste des mathématiques en général, et du calcul en particulier. Le philosophe hostile à la notion d'*a priori* pourra parfaitement s'accommoder de nos dernières remarques. Il pourra ainsi défendre, dans une perspective néo-millienne, que les axiomes de ZFC ou les règles d'inférence du calcul des prédicats de premier ordre naissent par généralisation de faits de l'expérience. Ou l'on pourra encore, avec David Deutsch, défendre que les manipulations de symboles doivent en dernier recours compter comme une expérience, et que donc que les vérités mathématiques, si elles sont *métaphysiquement a priori*, ne le sont pas *épistémiquement* (voir [Deutsch 1997, 157–163 ; 293–299]). Sans prendre position dans ces débats épineux, il s'agit juste ici d'admettre que les résultats de calcul constituent une connaissance

22. Comme nous l'avons déjà signalé précédemment, il s'agit ici d'une description idéalisée, et non d'une description de la pratique mathématique courante. Dans la pratique, on utilise des algorithmes dont on n'a pas forcément démontré la correction, et les calculs et démonstrations ne sont le plus souvent que partiellement formalisés.

a priori dans un certain sens « naïf », inanalysé, et de voir qu'on peut en tirer des conséquences philosophiques intéressantes. Pour le dire en d'autres termes : admettons naïvement l'intuition courante selon laquelle les résultats de calcul papier-crayon sont une connaissance *a priori* et voyons qu'on peut néanmoins étudier l'épistémologie du calcul avec intérêt.

4.2 Le calcul sur machines et la notion d'*a priori*

La situation est sensiblement différente lorsqu'on examine le calcul effectué par d'autres moyens. Considérons tout d'abord le cas des calculs effectués sur machine digitale programmable. La démonstration de correction d'un algorithme compte toujours comme un candidat raisonnable au titre de connaissance *a priori*, si l'on ignore les complications introduites par l'emploi d'assistants automatisés. Mais la vérification de la correction de l'exécution ne peut plus être effectuée par un sujet humain. Comme on l'a vu ci-dessus, la correction de l'exécution dépend désormais des principes empiriques régissant la machine, de son ingénierie particulière et de toutes les opérations de contrôle et de maintenance nous assurant que notre exemplaire particulier de machine fonctionne convenablement. Il s'agit là d'un ensemble de considérations hétérogènes, mais elles seront toutes exprimées par des propositions méritant d'être qualifiées d'empiriques, au sens élémentaire défini plus haut : on peut réviser leur valeur de vérité suite à de nouvelles expériences. Toutes les considérations garantissant le bon fonctionnement d'un ordinateur sont empiriques en ce sens, y compris les principes fondamentaux de la physique (voir [Deutsch 1985, 4–5]). Savoir qu'un ordinateur fonctionne bien est donc globalement une question empirique. Les critères de correction d'un calcul effectué sur une machine digitale programmable mélangent donc des considérations qui, à tout le moins, peuvent être *a priori* — la démonstration de la correction de l'algorithme — et des considérations *a posteriori* — le bon fonctionnement de notre ordinateur. Pour qu'une proposition soit qualifiée d'*a priori*, il faut cependant qu'aucune des considérations établissant sa valeur de vérité ne dépende de l'expérience. Lorsque nous employons une machine digitale programmable, la connaissance que nous avons de la valeur de certaines fonctions en certains de leurs arguments peut donc être qualifiée de connaissance empirique ou connaissance *a posteriori*.

Pour ce qui est des machines analogiques, le caractère empirique de la connaissance obtenue est plus flagrant encore, puisque n'y figure plus la démonstration de correction d'un algorithme. Connaître la dynamique d'un système, sa préparation dans un état et les résultats des mesures effectuées sur lui : toutes ces considérations sont de nature empirique. La connaissance obtenue par le biais d'une machine analogique est donc également une connaissance empirique.

Cependant, comme nous l'avons expliqué ci-dessus, nos machines actuelles, digitales comme analogiques, sont *en droit* simulables par un calcul papier-crayon. Nous *pourrions* donc effectuer tous les calculs effectués par nos machines digitales et analogiques, en ignorant les difficultés pratiques que nous venons de mentionner. Si nous atteignons *de facto* une partie de notre connaissance mathématique par des moyens empiriques, nous sommes toujours capables *de jure* de l'atteindre par l'exécution symbolique du calcul. À l'heure actuelle, tous les calculs que nous effectuons sur machine sont donc *a priori de jure*.

Si l'on adopte une définition épistémique de la connaissance *a priori*, il est donc possible qu'une même proposition soit *a priori* et *a posteriori*. Ceci a déjà été remarqué par Elliott Sober à propos du problème de Plateau (voir [Sober 1993], [Sober 2000]). Le mathématicien belge Joseph Plateau, pour résoudre le problème de la surface minimum adhérente à une courbe, plongeait une courbe formée par un fil de fer dans un bassin de savon. En partant de l'hypothèse *physique* que la surface du film de savon, en minimisant l'énergie potentielle, serait la surface minimale adhérente à la courbe, Plateau pouvait donc résoudre une instance du problème, pour la courbe fermée représentée par le fil de fer.

Je rajouterai seulement deux remarques à la réflexion de Sober. Tout d'abord, sa réflexion au sujet du problème de Plateau se généralise à tous les résultats de calculs obtenus à l'aide d'une machine analogique. Le dispositif de Plateau peut d'ailleurs lui-même être conçu comme un cas très particulier de machine analogique [23]. Ensuite, lorsqu'on affirme qu'une même connaissance peut être à la fois *a priori* et *a posteriori*, il faut distinguer deux cas. Dans le cas du problème de Plateau, comme de ces calculs qu'on effectue à la main avant de les vérifier sur une calculatrice, la connaissance obtenue est en réalité *a priori* et *a posteriori de facto*. Dans le cas de calculs très coûteux en espace et en temps dont on ne connaîtra jamais, dans les faits, le résultat sans l'aide d'une machine, la connaissance obtenue est *a priori de jure* et *a posteriori de facto*. Pour reprendre l'exemple du problème de Plateau, un mathématicien peut en effet réussir à démontrer les résultats obtenus par l'emploi du fil de fer plongé dans le savon. Ce n'est pas le cas de tous les résultats obtenus à l'aide de machines : certains résultats obtenus aujourd'hui à l'aide d'ordinateurs puissants seraient pratiquement impossibles à retrouver par un calcul papier-crayon. Il convient donc de distinguer deux acceptions de la notion d'*a priori* et de préciser laquelle on emploie pour éviter de faire croire qu'on atteint une même connaissance par deux moyens différents, alors qu'en pratique on ne l'atteindra parfois que par un d'entre eux. En toute généralité, la connaissance

23. Ceci pourrait d'ailleurs constituer une piste permettant d'expliquer en partie pourquoi les physiciens sont fréquemment capables de formuler des conjectures mathématiques que les mathématiciens ont les plus grandes difficultés à démontrer. Les physiciens peuvent en effet se servir des systèmes physiques qu'ils étudient comme des machines analogiques résolvant des instances de problèmes mathématiques complexes, ce qui permet plus aisément de formuler des conjectures prometteuses.

obtenue par le calcul avec nos machines modernes est donc *a priori* seulement *de jure,* et le plus souvent *a posteriori de facto.*

4.3 La thèse A et la connaissance *a priori*

Qu'adviendrait-il si la thèse A s'avérait être fausse ? Quel serait le statut épistémologique des calculs opérés à l'aide d'une machine violant cette thèse ? Pour le comprendre, il faut nous rappeler nos remarques sur l'impossibilité de simuler une machine violant la thèse A avec un algorithme. L'impossibilité de simuler la fonction calculée par notre machine à l'aide d'un algorithme n'interdit nullement de connaître des valeurs de cette fonction par d'autres méthodes de moindre généralité. Toute la connaissance mathématique obtenue empiriquement par notre machine pourrait donc, en droit, être obtenue par des moyens *a priori.* En aucun cas notre machine ne nous donnerait accès à une connaissance inatteignable par des méthodes mathématiques. En revanche, notre machine nous fournirait un moyen *a posteriori* pour résoudre un problème, là où les techniques mathématiques ne nous permettraient que de résoudre des cas particuliers. La thèse A garantit donc que tout problème mathématique soluble par un moyen empirique peut être résolu par une méthode *a priori.*

Lorsqu'un moyen de calcul respecte la thèse A, ce sont essentiellement les contingences empiriques liées aux ressources qui limitent notre capacité à connaître des valeurs de la fonction, tandis que la méthode *a priori* donnée par l'algorithme nous permet en droit de connaître toutes ces valeurs. Une machine violant la thèse A pourrait, elle aussi, être soumise à certaines limitations empiriques de ressources, qui l'empêcheraient d'évaluer la fonction en certains de ses arguments. Elle nous permettrait cependant de connaître plus de valeurs d'une fonction qu'aucune méthode *a priori* donnée. Au lieu que les contingences du monde empirique limitent la généralité idéale conférée par la connaissance *a priori,* nos ressources empiriques nous permettraient donc de gagner en généralité sur toute méthode *a priori.*

Conclusion

La forme algorithmique et la forme empirique de la thèse de Church-Turing ne sont pas trivialement équivalentes. Non seulement les concepts de machine et d'algorithme ne sont pas sémantiquement équivalents, mais il existe en outre des dispositifs empiriques permettant d'évaluer un objet en certains de ses arguments sans passer par un calcul symbolique. Ceci légitime la question soulevée par la thèse A : toutes les fonctions calculables par une machine sont-elles calculables par un algorithme ?

La forme physique de la thèse de Church-Turing, et à travers elle la thèse A, fait l'objet d'une investigation scientifique en cours. Sans vouloir me prononcer sur l'avancement de ces travaux, j'ai voulu montrer les enjeux philosophiques soulevés par cette thèse, en particulier autour de la notion d'*a priori*.

Pour ce faire, j'ai étudié le statut épistémologique du calcul, en distinguant les différents moyens employés pour calculer : calcul papier-crayon, machines digitale programmables, machines analogiques. Si l'on adopte une définition épistémique de l'*a priori*, le statut *a priori* ou *a posteriori* de la connaissance obtenue par le calcul dépend du moyen employé.

La simulabilité des machines par le calcul papier-crayon permet de comprendre comment une même proposition peut être à la fois *a posteriori* et *a priori*. En revanche, l'indispensabilité pratique des machines pour obtenir certains résultats de calcul justifie l'introduction d'une distinction entre *a priori de facto* et *a priori de jure*. Toutes ces considérations montrent la grande différence de perspective offerte par les acceptions métaphysique et épistémique de l'*a priori*.

Une machine violant la thèse A ne serait plus simulable par un calcul papier-crayon et par un calcul symbolique en général. Ceci n'interdirait pas qu'on puisse retrouver les résultats donnés par cette machine à l'aide d'une pluralité de méthodes mathématiques. Mais il existerait alors un moyen empirique pour résoudre un problème mathématique, qu'aucune méthode *a priori* ne pourrait résoudre dans le cas général. Bien qu'elle puisse être soumise à des contraintes empiriques de ressources, une machine violant la thèse A permettrait de connaître plus de valeurs d'une fonction qu'une quelconque méthode *a priori* donnée.

Mes résultats ont été formulés à l'aide d'une acception naïve de la notion d'*a priori*. Ils bénéficieraient grandement de l'approfondissement permis par l'emploi d'acceptions plus fines. Néanmoins, même si leur état présent d'élaboration ne permet pas de prendre position dans les débats sur la pertinence épistémologique de cette notion, il me semble qu'ils montrent tout l'intérêt que ces questionnements pourraient tirer d'une étude approfondie de l'épistémologie du calcul. Le calcul, et en particulier la notion de simulation, sont des lieux privilégiés pour interroger la vénérable notion d'*a priori*.

Remerciements

Je tiens à remercier mes directeurs de thèse, Jean-Baptiste Joinet et Alexei Grinbaum, ainsi que Gilles Dowek, Adrien Guatto, Giulio Guerrieri, Alberto Naibo, Mattia Petrolo, pour leurs corrections, discussions et commentaires de versions précédentes de ce travail.

Bibliographie

Arrighi, Pablo & Dowek, Gilles

2011 The physical Church-Turing thesis and the principles of quantum theory, accepté par *Int. J. Found. of Computer Science*. Preprint: arXiv:1102.1612.

Barrow, John D.

1992 *Perché il mondo è matematico ?*, Rome, Bari : Laterza, *Pourquoi le monde est-il mathématique ?*, trad. B. P. Marzi, Paris : Odile Jacob, 1996.

Boghossian, Paul A. & Peacocke, Christopher

2000 *New Essays on the A Priori*, Oxford; New York : Oxford University Press.

Bournez, Olivier & Campagnolo, Manuel L.

2008 *New Computational Paradigms. Changing Conceptions of What is Computable*, New York : Springer-Verlag, chap. A Survey on Continuous Time Computations, 383–423. `http://www.lix.polytechnique.fr/~bournez/i.php/Main/Publicationsbyyear?action=bibentry&bibfile=/perso.bib&bibref=CIEChapter2007`.

Bournez, Olivier, Dershowitz, Nachum & Falkovich, Evgenia

2012 Towards an axiomatization of simple analog algorithms, dans *Theory and Applications of Models of Computation – 9th Annual Conference, TAMC 2012, Beijing, China, May 16-21, 2012. Proceedings*, édité par Agrawal, M., Cooper, S. B. & Li, A., Springer-Verlag, *Lecture Notes in Computer Science*, t. 7287, 525–536.

Copeland, Jack

2003 Computation, dans *The Blackwell Guide to the Philosophy of Computing and Information*, édité par Floridi, L., Oxford : Blackwell, 3–17.

2004 Hypercomputation, *Theoretical Computer Science*, 317, 251–267.

Davis, Martin

2004 The myth of hypercomputation, dans *Alan Turing: The life and legacy of a great thinker*, édité par Teuscher, C., Berlin : Springer, 195–210.

Dershowitz, Nachum & Gurevich, Yuri

2008 A natural axiomatization of computability and proof of Church's thesis, *The Bulletin of Symbolic Logic*, 14(3), 299–350.

Deutsch, David

1985 Quantum theory, the Church-Turing Principle and the universal quantum computer, dans *Proceedings of the Royal Society of London*, t. 400, 1818, 97–117.

1997 *The Fabric of Reality*, London : Penguin Books, *L'Étoffe de la réalité,* trad. J. Balibar, Paris : Cassini, 2003.

Dowek, Gilles

2007 *Les Métamorphoses du calcul*, Paris : Le Pommier.

Farge, Marie

2007 Numerical experimentation: A third way to study nature, dans *Frontiers of Computational Sciences*, édité par Kaneda, Y., Kawamura, H. & Sasai, M., Heidelberg : Springer, 15–30.

Gandy, Robin

1980 Church's thesis and principles of mechanism, dans *The Kleene Symposium (Proceedings of the Symposium, University of Wisconsin, Madison, 1978)*, édité par Barwise, J., Kreisel, H. J. & Kunen, K., Amsterdam : North Holland Publishing, t. 101, 123–148.

Gurevich, Yuri

2012a What is an algorithm?, dans *SOFSEM 2012*, édité par Bieliková, M., Friedrich, G., Gottlob, G., Katzenbeisser, S. & Turán, G., Springer LNCS, 31–42, doi:http://dx.doi.org/10.1007/978-3-642-27660-6_3.

2012b Foundational analyses of computation, dans *CiE 2012*, édité par Cooper, S. Barry, Dawar, Anuj & Löwe, Benedikt, *Springer LNCS*, t. 7318, 264–275, doi:http://dx.doi.org/10.1007/978-3-642-30870-3.

Hodges, Andrew

2005 Can quantum computing solve classically unsolvable problems?. `ArXivPreprint:arXiv:quant-ph/0512248v1`.

Knuth, Donald E.

1997 *The Art of Computer Programming*, Reading : Addison-Wesley Professional.

Piccinini, Gualtiero

2011 The physical Church-Turing thesis: Modest or bold?, *British Journal for the Philosophy of Science*, 62, 733–769.

Pitowski, Itamar

1990 The physical Church thesis and physical computational complexity, *A Jerusalem Philosophical Quaterly*, 39(January), 81–99.

Pour-El, Marian B. & Richards, J. Ian

1989 *Computability in Analysis and Physics*, Berlin : Springer Verlag.

Rosser, John Barkley

1939 An informal exposition of proofs of Gödel's Theorem and Church's Theorem, *The Journal of Symbolic Logic*, 4, 53–60.

Sieg, Wilfried

2002 Calculations by Man & Machine: A mathematical presentation, dans *Proceedings of the Cracow International Congress of Logic, Methodology and Philosophy of Science*, Dordrecht : Kluwer Academic Publishers, 245–260.

2008 Church without dogmas: Axioms for computability, dans *New Computational Paradigms: Changing conceptions of what is computable*, édité par Lowe, B., Sorbi, A. & Cooper, B., New York : Springer, 139–152.

Smith, Peter

2007 *An introduction to Gödel's theorems*, Cambridge : Cambridge University Press.

Sober, Elliott

1993 Mathematics and indispensability, *The Philosophical Review*, 102(1), 35–57.

2000 Quine's two dogmas, dans *Proceedings of the Aristotelian Society, Supplementary Volume*, Blackwell Publishing, *1*, t. 74, 237–280.