Machine de Turing

Cursus – André-Guy Bruneau M.Sc. IT – Octobre 2025

Équivalence Fondamentale de Toute Computation

Au cœur de l'ère numérique se trouve un paradoxe saisissant : un supercalculateur de pointe comme El Capitan, capable d'exécuter des quintillions d'opérations par seconde, et le smartphone que nous tenons dans notre main, bien que séparés par des ordres de grandeur en termes de vitesse, de mémoire et d'architecture, sont fondamentalement équivalents sur le plan de la calculabilité. En principe, si on leur accordait des ressources de temps et de mémoire illimitées, ils pourraient résoudre exactement la même classe de problèmes. Cette équivalence conceptuelle ne se limite pas aux technologies actuelles ; elle s'étend à toute machine de calcul imaginable, passée, présente ou future. Qu'il s'agisse des calculateurs mécaniques du passé, des ordinateurs quantiques de demain ou même d'hypothétiques technologies extraterrestres, tous partagent un horizon commun de ce qui est calculable.

Cette universalité stupéfiante est cependant contrebalancée par l'existence de limites intrinsèques, non pas technologiques, mais profondément logiques. Ces frontières ne sont pas des obstacles que nous pourrons un jour franchir avec des processeurs plus rapides ou une mémoire plus vaste. Elles sont inscrites dans la nature même de ce qu'est un "algorithme", une procédure mécanique de calcul. Les exemples les plus célèbres de ces barrières infranchissables sont le "problème de l'arrêt" — l'impossibilité de créer un programme général capable de prédire si n'importe quel autre programme finira par s'arrêter — et l'existence d'une infinité de "fonctions non-calculables", des problèmes mathématiques bien définis pour lesquels aucune solution algorithmique ne pourra jamais exister. Ces limites sont universelles et absolues.

Ce livre blanc se propose de guider le lecteur à travers ce paysage fondamental de l'informatique théorique. Nous commencerons par explorer les origines historiques de ces concepts, nés d'une crise profonde des fondements des mathématiques au début du XXe siècle. Nous disséquerons ensuite la machine de Turing, le modèle abstrait qui a permis de formaliser ces idées, en passant de sa description conceptuelle à sa définition mathématique rigoureuse. Puis, nous examinerons comment ce simple modèle a pu devenir le fondement de toute l'informatique moderne, à travers la thèse de Church-Turing et le concept de machine universelle. Forts de cette compréhension, nous aborderons de front les limites infranchissables de la calculabilité, en démontrant leur existence et en explorant leurs manifestations dans divers domaines. Enfin, nous connecterons cette théorie abstraite au monde physique, en discutant des implications pour la physique, l'intelligence artificielle et la philosophie de l'esprit, clarifiant la distinction cruciale entre ce qui est calculable en principe et ce qui est réalisable en pratique. Ce voyage au cœur de la calculabilité révélera la double nature de l'héritage d'Alan Turing : une vision de l'universalité du calcul, et la découverte de ses limites éternelles.

Partie I : Les Origines Intellectuelles de la Machine

L'invention de la machine de Turing en 1936 n'est pas le fruit d'une ambition d'ingénierie visant à construire un calculateur physique. Elle est, au contraire, une construction purement intellectuelle, une réponse à une crise fondamentale qui secouait les fondements mêmes des mathématiques. Pour comprendre la portée de cette invention, il est indispensable de la replacer dans le contexte du débat intense qui animait la communauté des logiciens et des mathématiciens au début du XXe siècle.

Chapitre 1 : La Crise des Fondements et le Programme de Hilbert

Au tournant du XXe siècle, les mathématiques étaient empreintes d'un profond optimisme, incarné par la figure de David Hilbert. Ce dernier était convaincu que tout problème mathématique bien posé devait admettre une solution. Lors du Congrès international des mathématiciens de 1900, il présenta sa célèbre liste de 23 problèmes, guidant la recherche pour les décennies à venir.[1] Plus tard, dans les années 1920, il formula ce que l'on a appelé le "programme de Hilbert", une tentative ambitieuse de placer les mathématiques sur des fondations axiomatiques inébranlables. Ce programme visait trois objectifs majeurs :

1. Prouver que les mathématiques sont **complètes**, c'est-à-dire que toute affirmation mathématique vraie peut être prouvée à partir d'un ensemble fini d'axiomes.
2. Prouver que les mathématiques sont **cohérentes** (ou consistantes), c'est-à-dire qu'il est impossible de prouver à la fois une affirmation et sa négation à partir des mêmes axiomes.
3. Prouver que les mathématiques sont **décidables**, c'est-à-dire qu'il existe une procédure effective pour déterminer la vérité ou la fausseté de n'importe quelle proposition mathématique.[3]

C'est ce troisième objectif qui nous intéresse particulièrement. Il fut formalisé en 1928 par Hilbert et son élève Wilhelm Ackermann sous le nom d'*Entscheidungsproblem*, ou "problème de la décision".[5] La question était la suivante : existe-t-il un "procédé mécanique" ou un "algorithme" qui, pour n'importe quel énoncé formulé dans le langage de la logique du premier ordre, pourrait déterminer en un nombre fini d'étapes si cet énoncé est universellement valide, c'est-à-dire s'il est un théorème logique?.[3] Une réponse positive à cette question aurait eu des conséquences monumentales. Elle aurait signifié que la recherche de la vérité mathématique pouvait être entièrement automatisée, transformant le processus de démonstration en une simple procédure de calcul, certes potentiellement longue, mais garantie de réussir.[5]

Cependant, cet édifice optimiste fut profondément ébranlé avant même que l'on puisse s'attaquer sérieusement à l'*Entscheidungsproblem*. En 1931, le logicien autrichien Kurt Gödel publia ses deux théorèmes d'incomplétude. Le premier théorème démontrait que tout système axiomatique cohérent, suffisamment puissant pour formaliser l'arithmétique des entiers, est nécessairement incomplet : il existe des énoncés vrais sur les entiers qui ne peuvent pas être prouvés au sein de ce système.[3] Cette découverte porta un coup fatal aux deux premiers piliers du programme de Hilbert, prouvant que l'ambition d'une axiomatisation complète et prouvablement cohérente des mathématiques était vaine. La question de la décidabilité, l'*Entscheidungsproblem*, restait cependant ouverte, bien que l'onde de choc de Gödel ait semé le doute sur sa possible résolution.

Chapitre 2 : "On Computable Numbers" - La Révolution de Turing

C'est dans ce climat intellectuel, marqué par la remise en cause des certitudes mathématiques, qu'un jeune mathématicien de 24 ans de l'Université de Cambridge, Alan Turing, décida de s'attaquer à l'*Entscheidungsproblem*.[3] Son article de 1936, intitulé "On Computable Numbers, with an Application to the Entscheidungsproblem" ("Sur les nombres calculables, avec une application au problème de la décision"), allait non seulement fournir une réponse définitive à la question de Hilbert, mais aussi, de manière presque accidentelle, jeter les bases d'une toute nouvelle discipline : l'informatique théorique.[3]

Le coup de génie de Turing fut de réaliser que pour répondre à la question de l'existence d'un "procédé mécanique", il fallait d'abord définir rigoureusement ce que ce terme signifiait. Le mot "algorithme" était utilisé de manière intuitive depuis des siècles, mais il n'avait pas de définition mathématique formelle.[6] Plutôt que de se perdre dans des abstractions logiques, Turing choisit une approche radicalement originale : il modélisa le processus mental et physique d'un

"calculateur humain" en train de résoudre un problème mathématique avec un crayon et du papier. Il décomposa cette activité en ses actions les plus élémentaires et irréductibles :

- Lire un symbole sur une case d'une feuille de papier (le ruban).
- Écrire un nouveau symbole sur cette case.
- Changer son "état d'esprit" (par exemple, se souvenir de l'étape en cours du calcul).
- Déplacer son attention vers une case adjacente, à gauche ou à droite.[3]

De cette analyse, il tira un modèle abstrait, une machine imaginaire qu'il nomma "automatic machine" (ou a-machine). Cette machine, plus tard baptisée "machine de Turing" par le logicien américain Alonzo Church, était constituée d'un ruban infini servant de mémoire, d'une tête de lecture/écriture capable de se déplacer le long du ruban, et d'un mécanisme de contrôle fini possédant un nombre limité d'"états" internes.[3]

Avec ce modèle en main, Turing put formaliser la notion de "nombres calculables" : ce sont les nombres réels dont les décimales peuvent être énumérées, une par une, par une machine de Turing.[3] L'étape suivante fut de démontrer, par une construction logique ingénieuse, qu'il existait des problèmes bien définis qu'*aucune* machine de Turing ne pourrait jamais résoudre. Il y parvint en formulant une question spécifique, aujourd'hui connue sous le nom de "problème de l'arrêt" : peut-on concevoir une machine de Turing qui, recevant la description de n'importe quelle autre machine et son entrée, pourrait prédire si cette machine finira par s'arrêter ou si elle continuera à calculer indéfiniment?[5]

Turing prouva par l'absurde que la réponse était non. En montrant que l'existence d'une telle machine "décideuse" mènerait à une contradiction logique, il établit que le problème de l'arrêt est "indécidable". Finalement, il démontra que si l'on pouvait résoudre l'*Entscheidungsproblem*, on pourrait alors résoudre le problème de l'arrêt. Puisque le problème de l'arrêt est insoluble, il s'ensuit logiquement que l'*Entscheidungsproblem* l'est aussi.[5] La réponse à la question de Hilbert était donc négative : il n'existe pas d'algorithme universel pour décider de la vérité de toutes les propositions mathématiques.

Il est à noter qu'Alonzo Church, travaillant indépendamment à l'Université de Princeton, était parvenu à la même conclusion quelques mois plus tôt en utilisant un formalisme différent, le lambda-calcul.[5] Cependant, c'est le modèle de Turing, avec sa simplicité conceptuelle et sa ressemblance frappante avec un processus physique, qui a eu l'impact le plus durable.

Le processus intellectuel de Turing révèle une vérité profonde sur la nature de l'informatique. La question de Hilbert était formulée en termes vagues de "procédé mécanique". Pour la résoudre, une définition mathématique rigoureuse était nécessaire. Turing n'a pas cherché à concevoir une machine physique, mais a créé un modèle *abstrait* en analysant les actions cognitives minimales d'un calculateur humain. Cette abstraction a transformé une question de faisabilité pratique en un problème de logique mathématique pure. En résolvant ce problème, l'abstraction elle-même — la machine de Turing — est devenue un objet mathématique d'une richesse inouïe, le fondement sur lequel toute l'informatique théorique allait se construire. L'informatique n'est donc pas née de l'ingénierie ou de la construction de calculateurs, mais d'un effort purement intellectuel pour définir les limites de la raison algorithmique. La machine de Turing n'est pas un prototype d'ordinateur ; l'ordinateur moderne est une incarnation physique des principes universels découverts par Turing.

Partie II : Anatomie d'une Machine Abstraite

Pour apprécier pleinement la puissance et les limites du modèle de Turing, il est essentiel de dépasser la description

conceptuelle et de plonger dans sa structure formelle. Cette partie dissèque les composants de la machine, formalise son fonctionnement et illustre son mécanisme à travers un exemple concret de calcul.

Chapitre 3 : Définition Formelle de la Machine de Turing

Une machine de Turing, bien qu'abstraite, est définie par un ensemble de composants précis et interdépendants. Sa puissance réside dans la simplicité et la clarté de ces éléments.[14]

Les Composants Essentiels

1. **Le Ruban (Tape) :** Il s'agit d'une bande théoriquement infinie, divisée en une succession de cases discrètes. Ce ruban sert de mémoire à la machine, pour l'entrée, le calcul intermédiaire et la sortie. Chaque case peut contenir un unique symbole, issu d'un alphabet fini. L'infinité du ruban est une idéalisation mathématique cruciale : elle ne signifie pas que le ruban est physiquement infini, mais que la machine peut toujours demander plus d'espace si nécessaire, garantissant qu'un calcul ne s'arrête jamais par manque de mémoire.[12] Dans la plupart des modèles, le ruban est infini dans une seule direction (généralement vers la droite), mais des variantes avec un ruban infini dans les deux sens existent et sont équivalentes en puissance de calcul.[14]
2. **La Tête de Lecture/Écriture (Head) :** C'est un dispositif qui, à tout instant, est positionné au-dessus d'une unique case du ruban. Ses fonctions sont triples : elle peut **lire** le symbole présent sur la case, **écrire** un nouveau symbole à sa place (en effaçant l'ancien), et **se déplacer** d'une case, soit vers la gauche, soit vers la droite.[15] L'action de la tête est l'opération atomique de la machine.
3. **Le Registre d'États (State Register) :** Il s'agit du "cerveau" de la machine, une unité de contrôle qui peut se trouver dans un état parmi un ensemble fini et prédéfini d'états. Ces états représentent la "mémoire interne" ou le contexte du calcul en cours. Parmi ces états, on distingue un **état initial** (q_0), dans lequel la machine commence toujours son calcul, et un ou plusieurs **états d'arrêt** (ou états finaux). Typiquement, on définit un **état d'acceptation** (q_{accept}) et un **état de rejet** (q_{reject}), qui signalent la fin du calcul et son résultat.[12]

La Fonction de Transition (δ) : Le "Programme"

Le comportement de la machine est entièrement déterminé par sa fonction de transition, souvent notée δ. Cette fonction est l'incarnation du "programme" de la machine. Elle prend en entrée deux informations : l'**état actuel** de l'unité de contrôle et le **symbole actuellement lu** par la tête sur le ruban. En sortie, elle dicte de manière déterministe la prochaine action de la machine, sous la forme d'un triplet :

1. Le **nouvel état** dans lequel l'unité de contrôle doit passer.
2. Le **nouveau symbole** à écrire sur la case actuelle du ruban.
3. Le **mouvement** de la tête (Gauche, Droite, ou parfois Reste sur place).

Cette fonction peut être représentée sous forme d'une table, où chaque ligne correspond à une instruction du type : "Si tu es dans l'état q_i et que tu lis le symbole S_j, alors passe dans l'état q_k, écris le symbole S_l, et déplace la tête dans la direction D".[12] L'ensemble de ces règles, fini et explicite, constitue l'algorithme que la machine exécute.

Définition Formelle en 7-uplet

Pour permettre une analyse mathématique rigoureuse, une machine de Turing est formellement définie comme un 7-

uplet $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$. Cette notation encapsule tous les aspects fonctionnels de la machine dans un seul objet mathématique, ce qui est essentiel pour prouver des théorèmes sur ses capacités et ses limites.[18]

Table 1 : Définition Formelle d'une Machine de Turing (7-uplet)

Symbole	Nom	Description
Q	Ensemble des états	Un ensemble fini et non vide d'états dans lesquels l'unité de contrôle peut se trouver.
Σ	Alphabet d'entrée	Un ensemble fini de symboles autorisés dans la chaîne d'entrée initiale. Il ne contient pas le symbole blanc.
Γ	Alphabet du ruban	Un ensemble fini de symboles que la tête peut lire et écrire sur le ruban. Il doit contenir l'alphabet d'entrée ($\Sigma \subseteq \Gamma$) et un symbole spécial, le symbole "blanc" ($\square \in \Gamma \setminus \Sigma$).
δ	Fonction de transition	La fonction qui définit le programme de la machine. C'est une application de la forme $\delta: Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$, où L signifie "déplacement à gauche" et R "déplacement à droite".
q_0	État initial	L'état dans lequel la machine commence son calcul. $q_0 \in Q$.
q_{accept}	État d'acceptation	Un état final qui, une fois atteint, provoque l'arrêt de la machine et signifie que l'entrée a été acceptée. $q_{accept} \in Q$.
q_{reject}	État de rejet	Un état final qui, une fois atteint, provoque l'arrêt de la machine et signifie que l'entrée a été rejetée. $q_{reject} \in Q$ et $q_{reject} \neq q_{accept}$.

Sources : Synthèse de.[14]

Le passage de la description informelle (ruban, tête) à ce formalisme mathématique est une étape cruciale. Il permet de raisonner sur l'ensemble de toutes les machines de Turing possibles et de prouver des résultats universels, tels que l'indécidabilité du problème de l'arrêt, qui s'appliquent non seulement à une machine particulière, mais à la totalité du modèle de calcul.

Chapitre 4 : La Machine en Action : Configurations et Calculs

Une fois la structure de la machine définie, il faut décrire précisément comment elle opère dans le temps. Le concept de "configuration" est central pour capturer l'évolution d'un calcul.

Le Concept de Configuration

Une configuration est un "instantané" complet de l'état de la machine à un instant t. Pour connaître l'état global du système et prédire son état à l'instant $t+1$, trois informations sont nécessaires et suffisantes :

1. L'état actuel de l'unité de contrôle ($q \in Q$).
2. Le contenu complet du ruban (la chaîne de symboles).
3. La position de la tête de lecture/écriture sur le ruban.

Formellement, une configuration est souvent représentée par une chaîne de caractères ou un triplet. Une notation courante est uqv, où q est l'état actuel, la chaîne u représente la partie du ruban à gauche de la tête, et la chaîne v représente la partie du ruban commençant sous la tête et s'étendant vers la droite. Par convention, la tête scanne le premier symbole de v. Cette notation unique capture les trois éléments essentiels de manière compacte.[14]

Déroulement d'un Calcul

Un calcul est simplement une séquence de configurations $C_0, C_1, C_2, \ldots$.

- La configuration initiale, C_0, est définie par l'état initial q_0, la chaîne d'entrée w écrite sur le ruban (le reste étant rempli de symboles blancs), et la tête positionnée sur le premier symbole de w.
- Le passage d'une configuration C_i à la suivante C_{i+1} est appelé une **transition** ou un **mouvement**. Ce passage est entièrement déterminé par l'application de la règle unique de la fonction de transition δ correspondant à l'état et au symbole scanné dans la configuration C_i.
- Le calcul se termine, ou **s'arrête**, si la machine atteint une configuration dont l'état est soit q_{accept}, soit q_{reject}. Si la machine n'atteint jamais un tel état, elle continue à calculer indéfiniment ; on dit alors qu'elle **boucle**.[15]

Exemple Détaillé : L'Addition Unaire

Pour rendre ces concepts concrets, examinons comment une machine de Turing peut effectuer une opération arithmétique simple : l'addition de deux nombres entiers. Nous utiliserons la représentation unaire, où un nombre n est représenté par une séquence de n symboles identiques (par exemple, des '0'). Le problème est de calculer $m+n$. L'entrée sur le ruban sera une séquence de m zéros, suivie d'un symbole séparateur 'c', suivi de n zéros. Par exemple, pour calculer 2+3, l'entrée initiale sera 00c000. Le résultat souhaité sur le ruban est 00000.[24]

L'algorithme intuitif pour résoudre ce problème est le suivant :

1. Parcourir le ruban vers la droite pour trouver le séparateur 'c'.
2. Remplacer le 'c' par un '0'.
3. Continuer à se déplacer vers la droite jusqu'à la fin de la seconde chaîne de '0' (c'est-à-dire jusqu'à trouver un symbole blanc).
4. Se déplacer d'une case vers la gauche.
5. Effacer le dernier '0' (le remplacer par un symbole blanc).

6. Le ruban contient maintenant $m+n$ zéros, et le calcul est terminé.

Cet algorithme peut être traduit directement en une table de transition pour une machine de Turing. Les états de la machine correspondront aux étapes logiques de l'algorithme : "chercher 'c'", "chercher le blanc final", "revenir en arrière et effacer", etc.

Table 2 : Exemple de Table de Transition pour l'Addition Unaire ($2+3 \to 5$)

Alphabet du ruban $\Gamma = \{0, c, \square\}$ (où $\square$ est le symbole blanc)
États $Q = \{q_0, q_1, q_2, q_3, q_{accept}\}$

État Actuel	Symbole Lu	Nouvel État	Symbole Écrit	Mouvement	Description de l'Action
q_0	0	q_0	0	R	Parcourt les premiers '0' vers la droite.
q_0	c	q_1	0	R	Trouve 'c', le remplace par '0' et passe à l'état suivant.
q_1	0	q_1	0	R	Parcourt les seconds '0' vers la droite.
q_1	$\square$	q_2	$\square$	L	Trouve la fin, commence à revenir en arrière.
q_2	0	q_3	$\square$	L	Efface le dernier '0'.
q_3	0	q_3	0	L	Revient au début du ruban.
q_3	$\square$	q_{accept}	$\square$	R	Atteint le début, s'arrête et accepte.

Note : Cette table est une implémentation simplifiée de l'algorithme. D'autres implémentations sont possibles.[24]

Trace d'Exécution pour l'entrée 00c000

Voici la séquence de configurations que la machine suivrait :

1. $(q_0, \underline{0}0c000\square\ldots)$
2. $(q_0, 0\underline{0}c000\square\ldots)$
3. $(q_0, 00\underline{c}000\square\ldots)$

4. $(q_1, 000\underline{0}00\square\ldots)$
5. $(q_1, 0000\underline{0}0\square\ldots)$
6. $(q_1, 00000\underline{0}\square\ldots)$
7. $(q_1, 000000\underline{\square}\ldots)$
8. $(q_2, 00000\underline{0}\square\ldots)$
9. $(q_3, 0000\underline{0}\square\square\ldots)$
   10.... la machine continue de se déplacer à gauche...
10. $(\underline{\square}00000\square\square\ldots, q_3)$
11. $(\square\underline{0}0000\square\square\ldots, q_{accept})$ -> **HALT**

Le ruban final contient 00000, ce qui représente bien le nombre 5. Cet exemple, bien que simple, démontre comment une tâche arithmétique peut être décomposée en une séquence d'opérations symboliques purement mécaniques. La présentation de la table de transition rend tangible le concept abstrait de "programme" d'une machine de Turing, illustrant la traduction d'un algorithme en un ensemble fini de règles déterministes.

Partie III : L'Universalité du Modèle de Turing

La véritable puissance du modèle de Turing ne réside pas dans sa capacité à résoudre des problèmes spécifiques comme l'addition, mais dans son universalité. Les concepts développés par Turing et ses contemporains ont révélé que ce modèle simple capture l'essence de tout ce qui est calculable, jetant ainsi les bases de l'informatique moderne.

Chapitre 5 : La Thèse de Church-Turing

Dans les années 1930, plusieurs mathématiciens, travaillant indépendamment, ont cherché à formaliser la notion intuitive d'algorithme. Outre la machine de Turing, deux autres modèles majeurs ont émergé :

- Le **lambda-calcul**, développé par Alonzo Church, est un système formel basé sur la notion de fonction, d'application et de substitution de variables. Il est à l'origine des langages de programmation fonctionnels.[4]
- Les **fonctions récursives générales**, étudiées par Kurt Gödel, Jacques Herbrand et Stephen Kleene, définissent une classe de fonctions sur les entiers construites à partir de fonctions de base (successeur, constante, projection) et de règles de composition, de récursion et de minimisation.[4]

Ces trois approches semblaient radicalement différentes : l'une était mécanique (Turing), l'autre fonctionnelle (Church), et la troisième arithmétique (Gödel/Kleene). Pourtant, un résultat mathématique fondamental a été prouvé : ces trois modèles sont **formellement équivalents**. Toute fonction pouvant être calculée par une machine de Turing peut également être exprimée dans le lambda-calcul et définie comme une fonction récursive générale, et vice-versa.[26]

Cette convergence remarquable a conduit Church et Turing à formuler ce qui est aujourd'hui connu sous le nom de **thèse de Church-Turing**. Son énoncé le plus courant est le suivant :

> Toute fonction qui est "effectivement calculable" au sens intuitif (c'est-à-dire pour laquelle il existe un algorithme ou une procédure mécanique) est calculable par une machine de Turing.[26]

Cette thèse agit comme un pont entre le monde informel de l'intuition humaine sur ce que signifie "suivre une recette" et le monde formel et rigoureux des mathématiques. Elle affirme que le modèle de la machine de Turing capture avec succès et complètement notre concept d'algorithme.

Il est crucial de comprendre pourquoi il s'agit d'une "thèse" et non d'un "théorème". Un théorème se prouve à l'intérieur d'un système formel. Or, la thèse de Church-Turing relie un terme formel ("calculable par une machine de Turing") à un terme informel et intuitif ("effectivement calculable"). On ne peut pas prouver mathématiquement qu'une définition formelle correspond parfaitement à une intuition préexistante.[26] Les arguments en sa faveur sont donc de nature inductive et empirique :

1. **L'équivalence des modèles :** Tous les formalismes raisonnables et suffisamment puissants proposés pour capturer la notion de calcul se sont avérés équivalents à la machine de Turing.
2. **L'absence de contre-exemple :** Après plus de 80 ans de recherche intensive en informatique et en mathématiques, personne n'a jamais découvert de procédure que l'on considérerait intuitivement comme un algorithme mais qui ne pourrait pas être implémentée par une machine de Turing.[27]

La thèse est aujourd'hui universellement acceptée et constitue le postulat fondamental de l'informatique théorique. Elle nous autorise à identifier la notion de "calculable" avec celle de "Turing-calculable".

Chapitre 6 : La Machine de Turing Universelle (UTM)

Peu après avoir défini sa machine, Turing fit une découverte encore plus profonde. Il réalisa qu'il n'était pas nécessaire de concevoir une nouvelle machine matérielle pour chaque nouveau problème à résoudre. Il est possible de construire une *seule et unique* machine, la **Machine de Turing Universelle (UTM)**, qui est capable de simuler le comportement de n'importe quelle autre machine de Turing.[3]

Le principe de fonctionnement de l'UTM est la **simulation**. Pour simuler une machine spécifique M exécutant une entrée w, l'UTM reçoit sur son propre ruban une description codée de la machine M (sa table de transition, représentée comme une longue chaîne de symboles) suivie de l'entrée w.[34] L'UTM opère alors en plusieurs étapes répétitives :

1. Elle examine l'état actuel simulé de M et le symbole simulé sous la tête de M (qu'elle stocke sur une partie de son propre ruban).
2. Elle parcourt la description de M (la partie "programme" de son ruban) pour trouver la règle de transition qui correspond à l'état et au symbole actuels de M.
3. Elle applique cette règle en mettant à jour l'état simulé de M et en modifiant le ruban simulé de M en conséquence.
4. Elle répète ce cycle, simulant ainsi pas à pas l'exécution de M sur w.[35]

Cette idée, bien que purement théorique en 1936, est l'une des plus importantes de l'histoire de la technologie. L'acte d'encoder les instructions de la machine (M) sous forme de données sur le même ruban que les données d'entrée (w) est le concept fondateur du **programme enregistré** (*stored-program concept*).[34]

L'UTM est l'ancêtre théorique direct de l'ordinateur moderne. Dans cette analogie :

- La **Machine de Turing Universelle** correspond au **matériel** de l'ordinateur (le processeur, le CPU), dont la conception est fixe et universelle.
- La **description de la machine M** sur le ruban correspond au **logiciel** ou au **programme** que l'on charge en mémoire.
- L'**entrée w** correspond aux **données** sur lesquelles le programme opère.

Ainsi, l'UTM a démontré pour la première fois qu'une seule machine physique pouvait être rendue programmable pour exécuter une infinité de tâches différentes, simplement en lui fournissant différentes instructions sous forme de données. Ce principe est à la base de l'architecture de von Neumann, qui structure tous les ordinateurs que nous utilisons

aujourd'hui, des serveurs aux smartphones.[34]

Chapitre 7 : La Turing-Complétude

Le concept de machine universelle donne naissance à une classification fondamentale des systèmes de calcul : la **Turing-complétude**. Un système de manipulation de données — que ce soit un langage de programmation, un ensemble d'instructions de processeur, ou même un système plus exotique — est dit **Turing-complet** s'il possède une puissance de calcul suffisante pour simuler une machine de Turing universelle.[41]

En vertu de la thèse de Church-Turing, cela signifie qu'un système Turing-complet peut, en principe, calculer tout ce qui est calculable. La conséquence directe est que la quasi-totalité des langages de programmation d'usage général (Python, C++, Java, Lisp, etc.) sont Turing-complets. D'un point de vue purement théorique de la calculabilité, ils sont tous équivalents. Un programme écrit en Python peut être traduit en un programme équivalent en C++, qui peut à son tour être simulé par une machine de Turing. Les différences manifestes entre ces langages ne résident pas dans ce qu'ils *peuvent* calculer, mais dans leur efficacité, leur niveau d'abstraction, leur sécurité, leur facilité d'utilisation et leur écosystème.[42]

Quels sont les ingrédients minimaux pour qu'un système atteigne ce seuil d'universalité? La recherche en informatique théorique a montré que les exigences sont étonnamment simples. Un système est généralement Turing-complet s'il dispose de deux capacités fondamentales :

1. **La sélection conditionnelle :** La capacité d'exécuter différentes instructions en fonction d'une condition (par exemple, les structures if-then-else).
2. **L'itération ou la récursivité non bornée :** La capacité de répéter une séquence d'instructions un nombre de fois qui n'est pas fixé à l'avance (par exemple, une boucle while qui continue tant qu'une condition est vraie, ou des appels de fonction récursifs).[41]

La simplicité de ces exigences explique pourquoi la Turing-complétude apparaît dans des contextes très inattendus, bien au-delà des langages de programmation traditionnels. Des systèmes comme les feuilles de calcul Excel, le mécanisme de "redstone" dans le jeu *Minecraft*, ou même les règles complexes du jeu de cartes *Magic: The Gathering* se sont révélés être Turing-complets.[41]

Cela met en lumière une dichotomie fondamentale entre la puissance théorique et l'utilité pratique. La Turing-complétude est un concept binaire : un système l'est ou ne l'est pas. C'est le seuil maximal de la calculabilité, et il est atteint par une multitude de systèmes, y compris ceux qui n'ont pas été conçus pour la programmation. Cependant, le fait qu'un système soit Turing-complet ne dit absolument rien sur sa praticité. Programmer une fonction complexe en utilisant les règles de *Magic: The Gathering* est un exercice de logique fascinant, mais ce n'est pas une méthode de développement logiciel viable. La Turing-complétude est un outil de classification théorique essentiel, mais elle est largement orthogonale aux notions d'expressivité, d'efficacité et de pertinence d'un système de calcul dans le monde réel. La distinction entre "pouvoir tout calculer en principe" et "pouvoir calculer quelque chose d'utile en pratique" est fondamentale pour comprendre la portée et les limites de ce concept.

Partie IV : Les Frontières Infranchissables du Calculable

L'universalité du modèle de Turing, qui unifie tous les processus algorithmiques, a une contrepartie inéluctable : en définissant précisément ce qui *est* calculable, elle permet également de démontrer rigoureusement que certaines choses

ne le sont pas. Ces limites ne sont pas des échecs technologiques temporaires, mais des barrières logiques absolues, inhérentes à la nature même du calcul.

Chapitre 8 : L'Existence des Fonctions Non-Calculables

La première preuve de l'existence de problèmes insolubles ne vient pas d'un exemple spécifique, mais d'un argument d'une élégance et d'une puissance redoutables basé sur la comparaison de la taille de deux ensembles infinis.

Le Dénombrement des Machines de Turing

Chaque machine de Turing est entièrement définie par sa table de transition, qui est un ensemble fini de règles utilisant un alphabet fini. Cette description complète peut être encodée sous la forme d'une unique et longue chaîne de caractères. Tout comme n'importe quel texte peut être représenté numériquement (par exemple, en ASCII ou Unicode), cette chaîne de caractères peut être associée à un nombre entier unique.[11] Par conséquent, on peut imaginer une liste ordonnée de toutes les machines de Turing possibles : la machine M_1, la machine M_2, la machine M_3, et ainsi de suite, pour chaque entier positif. Cela signifie que l'ensemble de toutes les machines de Turing est **dénombrable**, c'est-à-dire qu'il a la même "taille" (ou cardinalité) que l'ensemble des nombres naturels $\mathbb{N}$.[11] Par extension, l'ensemble de tous les programmes informatiques possibles, dans n'importe quel langage, est également dénombrable.

L'Argument de la Diagonale de Cantor

En revanche, qu'en est-il de l'ensemble de tous les problèmes que l'on pourrait vouloir résoudre? En informatique, un problème peut être formalisé comme une fonction qui associe une sortie à une entrée. Considérons la classe la plus simple de problèmes de décision : les fonctions qui prennent un nombre entier en entrée et retournent soit 0, soit 1 (équivalent à "non" ou "oui"). L'ensemble de toutes les fonctions possibles de $\mathbb{N}$ vers $\{0, 1\}$ est-il dénombrable?

Le mathématicien Georg Cantor a prouvé, à la fin du XIXe siècle, que la réponse est non. Son argument, connu sous le nom d'**argument de la diagonale**, est une preuve par l'absurde.[51] Supposons que l'on puisse lister toutes ces fonctions : $f_0, f_1, f_2, \ldots$. On peut visualiser cette liste comme un tableau infini où la ligne i représente les valeurs de la fonction f_i pour chaque entrée j :

	0	**1**	**2**	**3**	**...**
f_0	$f_0(0)$	$f_0(1)$	$f_0(2)$	$f_0(3)$	...
f_1	$f_1(0)$	$f_1(1)$	$f_1(2)$	$f_1(3)$	...
f_2	$f_2(0)$	$f_2(1)$	$f_2(2)$	$f_2(3)$	...
f_3	$f_3(0)$	$f_3(1)$	$f_3(2)$	$f_3(3)$	...

...	...	...	...	...	...

Construisons maintenant une nouvelle fonction, appelons-la g, en prenant la "diagonale" de ce tableau et en inversant chaque valeur. Formellement, pour tout entier n, nous définissons $g(n) = 1 - f_n(n)$.

Par sa construction même, la fonction g est différente de chaque fonction f_n de la liste. En effet, $g(n)$ est différent de $f_n(n)$ pour tout n. Par exemple, $g(0) \neq f_0(0)$, $g(1) \neq f_1(1)$, etc. Par conséquent, la fonction g ne peut pas se trouver dans notre liste. Mais nous avions supposé que la liste contenait *toutes* les fonctions possibles. Cette contradiction prouve que notre supposition initiale était fausse : il est impossible de lister toutes les fonctions de $\mathbb{N}$ vers $\{0, 1\}$. L'ensemble de ces fonctions est **non-dénombrable**.[50]

La Conclusion Inéluctable

Nous avons donc deux faits :

1. L'ensemble des outils pour résoudre les problèmes (les programmes, les machines de Turing) est dénombrable.
2. L'ensemble des problèmes à résoudre (les fonctions) est non-dénombrable.

La conclusion est mathématiquement inévitable : il y a "infiniment plus" de problèmes que de programmes pour les résoudre. Il doit donc exister des fonctions pour lesquelles aucun programme, aucune machine de Turing, ne peut être conçu pour les calculer. Ces fonctions sont appelées **fonctions non-calculables**.[49] Loin d'être des curiosités rares, elles constituent en réalité la vaste majorité des fonctions mathématiques. Le monde du calculable est une petite île dans un océan infini de non-calculable.

Chapitre 9 : Le Problème de l'Arrêt (The Halting Problem)

L'argument de cardinalité prouve l'existence de problèmes insolubles, mais il n'en fournit pas d'exemple concret. Alan Turing a été le premier à en exhiber un, qui est devenu le problème indécidable le plus célèbre de l'informatique : le **problème de l'arrêt**.

Énoncé du Problème

La question est la suivante : existe-t-il un algorithme universel, une machine de Turing que nous appellerons HaltChecker, qui puisse prendre en entrée la description de n'importe quel autre programme P et n'importe quelle entrée I pour ce programme, et qui puisse toujours répondre correctement à la question : "Le programme P, lorsqu'il est exécuté avec l'entrée I, finira-t-il par s'arrêter?".[53]

Les conditions sont strictes :

- HaltChecker doit fonctionner pour **tous** les couples (P, I).
- HaltChecker doit lui-même **toujours s'arrêter** et fournir une réponse claire : "oui, il s'arrête" ou "non, il boucle à l'infini".[56]

La Preuve par l'Absurde de Turing

La preuve de Turing est un chef-d'œuvre de logique qui utilise une forme d'auto-référence pour créer un paradoxe. Elle

se déroule en quatre étapes.[57]

1. **L'Hypothèse :** Supposons, pour les besoins de l'argumentation, qu'un tel programme HaltChecker(P, I) existe. Il s'agit d'une "boîte noire" magique qui résout le problème de l'arrêt.
2. **La Construction du "Contrariant" :** Si HaltChecker existe, alors nous pouvons l'utiliser comme une brique pour construire un nouveau programme, que nous appellerons Reverser. Ce programme Reverser prend en entrée la description d'un seul programme, P. Voici ce qu'il fait :
    - Il utilise HaltChecker pour analyser ce qui se passerait si P était exécuté avec sa propre description comme entrée, c'est-à-dire qu'il calcule HaltChecker(P, P).
    - Il inverse ensuite le résultat :
        - Si HaltChecker(P, P) répond "oui" (c'est-à-dire que P s'arrête lorsqu'il est exécuté sur lui-même), alors Reverser entre délibérément dans une boucle infinie.
        - Si HaltChecker(P, P) répond "non" (c'est-à-dire que P boucle à l'infini lorsqu'il est exécuté sur lui-même), alors Reverser s'arrête immédiatement.[54]
3. **Le Paradoxe :** Le piège se referme lorsque l'on pose la question : que se passe-t-il si nous donnons à Reverser sa propre description comme entrée? Que fait l'exécution de Reverser(Reverser)?
    - À l'intérieur de Reverser(Reverser), l'appel HaltChecker(Reverser, Reverser) est effectué.
    - **Cas 1 : Supposons que HaltChecker(Reverser, Reverser) réponde "oui".** Cela signifie que Reverser s'arrête sur sa propre entrée. Mais, par définition de Reverser, si HaltChecker répond "oui", alors Reverser doit entrer dans une boucle infinie. Donc, Reverser ne s'arrête pas. C'est une contradiction.
    - **Cas 2 : Supposons que HaltChecker(Reverser, Reverser) réponde "non".** Cela signifie que Reverser ne s'arrête pas sur sa propre entrée. Mais, par définition de Reverser, si HaltChecker répond "non", alors Reverser doit s'arrêter. Donc, Reverser s'arrête. C'est une nouvelle contradiction.
4. **La Conclusion :** Dans les deux cas possibles, nous aboutissons à une contradiction logique insurmontable. La seule façon de résoudre ce paradoxe est de conclure que notre hypothèse de départ était fausse. Un programme comme HaltChecker ne peut pas exister. Le problème de l'arrêt est donc **indécidable**.[53]

Cette preuve ne démontre pas seulement qu'un problème est insoluble ; elle révèle une limite fondamentale de l'analyse algorithmique. Aucun programme ne peut analyser de manière exhaustive et toujours correcte le comportement d'un autre programme arbitraire, car le programme analysé peut toujours utiliser la prédiction de l'analyseur pour la contredire.

Chapitre 10 : Un Panorama de l'Indécidable

Le problème de l'arrêt n'est que la pointe de l'iceberg. Sa découverte a ouvert la voie à l'identification de toute une classe de problèmes fondamentalement insolubles.

Le Théorème de Rice

Le théorème de Rice est une généralisation puissante et stupéfiante de l'indécidabilité du problème de l'arrêt. Il affirme que **toute propriété sémantique non-triviale des programmes est indécidable**.[56] Décortiquons cette affirmation :

- Une **propriété sémantique** est une propriété qui concerne le *comportement* du programme (ce que la fonction calcule), et non sa structure syntaxique (le texte du code). Par exemple, "le programme contient-il une boucle for?" est une question syntaxique (décidable), mais "le programme s'arrêtera-t-il un jour?" est une question sémantique.
- Une propriété est **non-triviale** si elle est vraie pour au moins un programme et fausse pour au moins un autre. Par exemple, "le programme calcule-t-il une fonction?" est trivial (vrai pour tous), mais "le programme s'arrête-t-il sur

l'entrée 0?" est non-trivial.

Le théorème de Rice implique donc qu'il est impossible de créer un algorithme général pour décider de questions telles que :

- Ce programme s'arrête-t-il pour toutes les entrées? (Le problème de la totalité)
- Ce programme calcule-t-il la fonction identité?
- Le langage accepté par ce programme est-il vide, fini, infini, régulier, ou context-free?
- Ce programme est-il équivalent à un autre programme donné?

Toutes ces questions, et une infinité d'autres, sont indécidables.[57]

Le Problème de Correspondance de Post (PCP)

Le PCP est un autre exemple célèbre de problème indécidable, souvent utilisé dans les preuves de réduction en raison de sa formulation plus simple que celle du problème de l'arrêt. Il peut être visualisé comme un puzzle de dominos. On dispose d'une collection finie de types de dominos, chaque domino ayant une chaîne de caractères sur sa moitié supérieure et une autre sur sa moitié inférieure. La question est : peut-on trouver une séquence de ces dominos (avec répétitions autorisées) de sorte que la chaîne formée par la concaténation des moitiés supérieures soit identique à celle formée par les moitiés inférieures?[62]

Malgré son apparence ludique, ce problème est indécidable. La preuve consiste à montrer que l'on peut construire un ensemble de dominos PCP qui simule l'historique de calcul d'une machine de Turing arbitraire sur une entrée donnée. Une solution (une "correspondance") au PCP n'existera que si et seulement si la machine de Turing s'arrête et accepte son entrée. Puisque la décision de l'acceptation par une TM est indécidable (une variante du problème de l'arrêt), le PCP doit l'être aussi.[62]

Le Dixième Problème de Hilbert

Ce problème, posé par David Hilbert en 1900, est antérieur à la théorie de la calculabilité. Il demande s'il existe un algorithme général pour déterminer si une **équation diophantienne** — une équation polynomiale à coefficients entiers — a des solutions entières.[1] Par exemple, $x^2 + y^2 = z^2$ a des solutions (par exemple, 3, 4, 5), mais $x^3 + y^3 = z^3$ n'en a pas pour des entiers non nuls.

Pendant 70 ans, le problème est resté ouvert. Finalement, en 1970, le mathématicien russe Yuri Matiyasevich, en s'appuyant sur les travaux fondamentaux de Martin Davis, Hilary Putnam et Julia Robinson (formant le théorème MRDP), a prouvé qu'un tel algorithme ne peut pas exister. Le dixième problème de Hilbert est indécidable.[1] La preuve, extrêmement complexe, établit un lien profond entre la théorie des nombres et la calculabilité, en montrant que tout ensemble récursivement énumérable (la classe des ensembles dont les membres peuvent être listés par une machine de Turing) peut être représenté comme l'ensemble des solutions d'une équation diophantienne.

Ces exemples, issus de domaines aussi variés que la logique, les puzzles combinatoires et la théorie des nombres, ne sont pas des cas isolés. Ils révèlent une propriété structurelle profonde de la computation. Le problème de l'arrêt n'est pas une simple anomalie ; il est la source primitive de l'indécidabilité. Le théorème de Rice montre que l'incapacité à prédire le comportement est la norme pour toute question sémantique intéressante. Les preuves d'indécidabilité pour des problèmes comme le PCP ou le dixième problème de Hilbert passent presque invariablement par une **réduction** à partir du problème de l'arrêt, montrant que si l'on pouvait résoudre ces problèmes, on pourrait résoudre le problème de l'arrêt.

L'indécidabilité n'est donc pas une collection de curiosités, mais une propriété fondamentale et unificatrice qui émerge dès qu'un système formel devient suffisamment puissant pour effectuer des calculs universels et pour s'auto-référencer. La structure logique du problème de l'arrêt "infecte" de vastes domaines des mathématiques et de l'informatique, traçant une limite définitive à ce que la connaissance algorithmique peut atteindre.

Partie V : Incarnations Physiques et Portée Philosophique

Après avoir exploré le monde abstrait de la calculabilité et de ses limites, il est temps de le reconnecter au monde physique des ordinateurs réels et aux questions fondamentales qu'il soulève sur la nature de l'univers et de l'esprit.

Chapitre 11 : Calculabilité vs. Efficacité : Le Monde Physique

Il est essentiel de réaffirmer une distinction cruciale : l'équivalence en termes de **calculabilité** ne signifie pas l'équivalence en termes d'**efficacité**.

Le Supercalculateur et le Smartphone : Équivalents mais Pas Égaux

Comme nous l'avons établi, un supercalculateur et un smartphone sont tous deux Turing-complets. Ils peuvent, en théorie, résoudre la même classe de problèmes. Cependant, dans la pratique, leurs performances sont radicalement différentes. Le supercalculateur CRAY-2 de 1985, une merveille de technologie à son époque, atteignait une performance de pointe de 1.9 gigaflops (milliards d'opérations en virgule flottante par seconde). Un iPhone 12 moderne, en comparaison, peut effectuer environ 11 téraflops (milliers de milliards d'opérations par seconde), soit plus de 5000 fois la vitesse du CRAY-2.[66] Cette différence abyssale n'est pas une question de calculabilité, mais de **complexité algorithmique** et d'ingénierie. Elle concerne le temps et les ressources (mémoire, énergie) nécessaires pour exécuter un calcul, des aspects que la théorie de la calculabilité ignore délibérément en postulant des ressources illimitées.

La Thèse de Church-Turing "Étendue" (ou "Forte")

Pour aborder la question de l'efficacité, les informaticiens ont formulé une version plus forte de la thèse de Church-Turing, souvent appelée la **thèse étendue** ou la **thèse de la complexité**. Elle ne postule pas seulement que tout ce qui est calculable peut l'être par une machine de Turing, mais que tout ce qui est calculable par un modèle de calcul "raisonnable" peut l'être *efficacement* par une machine de Turing (probabiliste). Le terme "efficacement" est généralement interprété comme "en temps polynomial", ce qui signifie que le temps de calcul ne croît pas de manière exponentielle avec la taille de l'entrée.[68] Cette thèse suggère que, bien que certains modèles de calcul puissent être plus rapides que d'autres, ils ne le sont que par un facteur polynomial, et qu'aucune machine physique ne pourrait résoudre "efficacement" des problèmes considérés comme "difficiles" (par exemple, la classe NP-complète) pour les ordinateurs classiques.

L'Informatique Quantique : Un Défi à l'Efficacité, pas à la Calculabilité

L'informatique quantique est souvent mal comprise comme un moyen de "dépasser" les limites de Turing. Ce n'est pas le cas.

- **Le Principe :** Les ordinateurs quantiques ne sont pas des "hypercalculateurs" qui calculent des fonctions non-calculables. Ils exploitent les principes de la mécanique quantique, tels que la superposition et l'intrication, pour explorer un vaste espace de solutions potentielles simultanément.[71]
- **Simulabilité et Limites :** Tout calcul effectué par un ordinateur quantique peut, en principe, être simulé par un ordinateur classique (une machine de Turing). La simulation serait extraordinairement lente (généralement exponentiellement plus lente), mais elle est possible. Par conséquent, les ordinateurs quantiques **n'étendent pas la classe des problèmes calculables**. Ils ne peuvent pas résoudre le problème de l'arrêt ou tout autre problème indécidable.[73]
- **Le Défi à la Thèse Étendue :** Le véritable impact de l'informatique quantique se situe au niveau de la complexité. Pour certains problèmes spécifiques, les algorithmes quantiques offrent une accélération spectaculaire. L'algorithme de Shor, par exemple, peut factoriser de grands nombres en temps polynomial, une tâche qui est considérée comme intraitable pour les meilleurs algorithmes classiques connus (qui prennent un temps super-polynomial).[73] De même, l'algorithme de Grover permet une recherche dans une base de données non structurée avec une accélération quadratique.[80] Ces résultats suggèrent fortement que la thèse de Church-Turing étendue est **fausse**. Il semble exister des processus physiques (quantiques) qui ne peuvent pas être simulés *efficacement* par une machine de Turing classique. La classe de complexité **BQP** (Bounded-error Quantum Polynomial time), qui contient les problèmes résolubles efficacement par un ordinateur quantique, semble être strictement plus grande que la classe **P** (problèmes résolubles efficacement par un ordinateur classique).[70]

Chapitre 12 : La Thèse de Church-Turing Physique et ses Débats

La connexion entre le calcul et le monde physique peut être poussée encore plus loin, transformant la thèse de Church-Turing en un principe physique fondamental.

La Thèse comme Principe Physique

La **Physical Church-Turing Thesis (PCTT)** est l'affirmation selon laquelle tout processus physique finiment réalisable peut être simulé par une machine de Turing universelle.[83] Cette thèse est beaucoup plus forte que l'originale. Elle ne prétend pas seulement modéliser le calcul humain, mais l'univers lui-même. Si la PCTT est vraie, cela implique que l'univers est, à un certain niveau, calculable. C'est une idée centrale de la "physique numérique", qui postule que l'univers pourrait être assimilé à un vaste automate cellulaire ou à un ordinateur.

Les Défis Théoriques : L'Hypercalcul

L'hypercalcul est le domaine spéculatif qui étudie des modèles de calcul hypothétiques, ou **hypercalculateurs**, capables de calculer des fonctions non-Turing-calculables, comme la résolution du problème de l'arrêt.[86] Bien que ces modèles ne soient pas considérés comme physiquement réalisables, ils servent de tests de pensée pour explorer les limites de la PCTT. Parmi les modèles proposés, on trouve :

- **Les machines à oracle :** Des machines de Turing dotées d'une "boîte noire" capable de répondre instantanément à une question indécidable. C'est un modèle purement abstrait.[86]
- **Les machines de Zénon :** Des machines qui exécuteraient une infinité d'étapes de calcul en un temps fini, en divisant

par deux la durée de chaque étape successive. Ce modèle se heurte à des paradoxes physiques et logiques évidents.[86]

- **Les modèles relativistes :** Certaines solutions exotiques des équations de la relativité générale (impliquant des trous noirs en rotation ou des courbes temporelles fermées) pourraient théoriquement permettre à un ordinateur d'effectuer un calcul infini tout en envoyant le résultat à un observateur en un temps fini. Cependant, la stabilité et l'existence de tels espaces-temps dans notre univers sont hautement improbables.[87]

La quasi-totalité de ces modèles d'hypercalcul se heurtent à des obstacles physiques qui semblent insurmontables, comme la nécessité d'une énergie infinie, d'une précision de mesure infinie, ou de conditions cosmologiques qui ne correspondent pas à nos observations. Ils restent donc largement du domaine de la spéculation théorique.[86]

La Question de l'Informatique Interactive

Un débat plus terre-à-terre concerne la pertinence du modèle de Turing pour l'informatique moderne. La machine de Turing classique modélise un calcul comme une fonction : une entrée est fournie, un traitement est effectué, une sortie est produite, et le processus s'arrête. Or, de nombreux systèmes informatiques contemporains (un serveur web, un système d'exploitation, un robot) ne fonctionnent pas de cette manière. Ils sont des **systèmes ouverts** qui interagissent en continu avec leur environnement, recevant des entrées et produisant des sorties de manière asynchrone et potentiellement infinie. Certains chercheurs, comme Peter Wegner, ont soutenu que ce paradigme de **calcul interactif** n'est pas entièrement capturé par le modèle de la machine de Turing et pourrait représenter une forme de calcul plus puissante, échappant aux conclusions de la thèse classique.[91] Ce débat reste ouvert, bien que la plupart des théoriciens considèrent que ces interactions peuvent encore être modélisées dans le cadre de Turing, bien que de manière plus complexe.

Chapitre 13 : La Machine de Turing et l'Esprit

L'une des implications les plus profondes et les plus controversées du modèle de Turing se trouve à l'intersection de l'informatique, de la psychologie et de la philosophie.

La Théorie Computationnelle de l'Esprit (CTM)

La CTM est une thèse philosophique majeure qui postule que l'esprit humain est un système de traitement de l'information, et que la cognition (la pensée, le raisonnement, la perception) est une forme de calcul. Dans cette perspective, le cerveau est le "matériel" (*hardware*) qui implémente les "programmes" mentaux (*software*).[93] Cette théorie s'appuie explicitement sur le modèle de la machine de Turing pour affirmer que les processus mentaux sont des manipulations de symboles (représentations mentales) basées sur des règles syntaxiques, tout comme une machine de Turing manipule des symboles sur son ruban selon sa table de transition.

L'Argument de Lucas-Penrose contre l'IA Forte

En opposition directe à la CTM, le philosophe J.R. Lucas et le physicien mathématicien Roger Penrose ont développé un argument célèbre basé sur le théorème d'incomplétude de Gödel pour affirmer que l'esprit humain ne peut pas être un système formel, et donc ne peut pas être une machine de Turing.[98]

Leur raisonnement est le suivant :

1. Supposons que l'esprit d'un mathématicien humain puisse être entièrement modélisé par une machine de Turing M.

2. Cette machine M est équivalente à un système formel F. Si nous supposons que le mathématicien est cohérent, alors F doit être un système cohérent.
3. Selon le premier théorème de Gödel, il existe une phrase, la phrase de Gödel $G(F)$, qui affirme "Je ne suis pas prouvable dans le système F".
4. Puisque F est cohérent, il ne peut pas prouver $G(F)$. Par conséquent, la machine M est incapable de "produire" ou de "connaître" la vérité de $G(F)$.
5. Cependant, le mathématicien humain, en examinant le système F et la construction de $G(F)$, peut "voir" par un raisonnement méta-mathématique que $G(F)$ est vraie.
6. Conclusion : L'esprit humain peut connaître une vérité que la machine M ne peut pas. Par conséquent, l'esprit humain n'est pas équivalent à la machine M. Comme cet argument peut être répété pour n'importe quelle machine M, l'esprit humain transcende les capacités de n'importe quelle machine de Turing.[98]

Critiques et État du Débat

L'argument de Lucas-Penrose a été largement critiqué et est rejeté par la majorité des logiciens, informaticiens et philosophes.[99] Les principales objections sont :

- **La cohérence de l'esprit humain :** L'argument suppose que l'esprit humain est mathématiquement cohérent. C'est une affirmation très forte et probablement fausse. Les humains font des erreurs de logique, se contredisent et croient en des paradoxes. Si l'esprit est un système incohérent, alors il peut "prouver" n'importe quoi, y compris sa phrase de Gödel, et l'argument s'effondre.[98]
- **La connaissance du système :** L'argument suppose que l'humain peut connaître la spécification exacte de sa propre "machine" M pour construire la phrase $G(M)$. Il est tout à fait possible que le système computationnel de l'esprit soit si complexe que nous ne puissions jamais en avoir une description complète, nous empêchant ainsi de formuler notre propre phrase de Gödel.[98]

Bien que l'argument soit généralement considéré comme non valide, il continue de stimuler le débat en soulignant les questions profondes et non résolues concernant la nature de la conscience, de l'intuition mathématique et la différence entre la manipulation syntaxique de symboles et la compréhension sémantique.

Conclusion : L'Héritage de Turing et les Limites de la Connaissance Algorithmique

L'héritage d'Alan Turing, cristallisé dans le concept de sa machine, est profondément double. D'une part, il nous a légué une vision d'une universalité stupéfiante. Le modèle de la machine de Turing universelle a révélé que le calcul n'est pas une collection de techniques disparates, mais un domaine unifié par un ensemble de principes fondamentaux. L'idée qu'une seule machine, avec un mécanisme fixe, puisse exécuter n'importe quel algorithme en lisant simplement des instructions sous forme de données, est le fondement théorique sur lequel repose toute l'informatique moderne. De l'architecture de von Neumann aux langages de programmation Turing-complets, chaque ordinateur, chaque programme que nous utilisons aujourd'hui est une incarnation de cette universalité. Cette vision a unifié le champ du calculable, montrant que malgré les apparences, du supercalculateur au smartphone, tous les dispositifs informatiques partagent la même puissance théorique fondamentale.

D'autre part, et c'est là le paradoxe central de son œuvre, c'est précisément cette formalisation rigoureuse de l'universalité qui a permis à Turing de découvrir ses limites absolues. En définissant ce qu'est un algorithme, il a pu prouver

mathématiquement qu'il existe des problèmes qu'aucun algorithme ne pourra jamais résoudre. Le problème de l'arrêt et l'infinité des fonctions non-calculables ne sont pas des défis technologiques à surmonter, mais des frontières logiques infranchissables. Ces limites ne sont pas propres à une architecture ou à une technologie particulière ; elles sont une propriété intrinsèque de tout système suffisamment puissant pour être universel. Aucune avancée future, qu'elle soit quantique, biologique ou autre, ne pourra contourner ces barrières, car elles ne sont pas physiques mais logiques.

La machine de Turing est donc bien plus qu'un simple modèle pour les ordinateurs. Elle est un outil épistémologique fondamental. Elle trace une ligne de démarcation claire et immuable entre ce qui est connaissable par un processus algorithmique et ce qui se situe au-delà de sa portée. Cette frontière n'est pas une limitation de notre ingéniosité, mais une caractéristique fondamentale de l'univers logique dans lequel nous opérons, aussi immuable que les lois de la physique qui régissent notre monde matériel. L'héritage de Turing est de nous avoir donné à la fois la clé du pouvoir computationnel universel et la carte de ses limites éternelles.

Annexe : La Turing-Complétude en Action (Études de Cas Ludiques)

Le concept de Turing-complétude, bien qu'abstrait, se manifeste dans des systèmes surprenants et souvent ludiques. Ces exemples illustrent de manière concrète comment des règles simples, combinées de manière ingénieuse, peuvent donner naissance à une capacité de calcul universelle.

Minecraft et la Logique Redstone

Le jeu vidéo *Minecraft* permet aux joueurs de construire des circuits en utilisant un matériau appelé "redstone", qui se comporte comme un fil électrique. En combinant des torches de redstone (qui agissent comme des inverseurs logiques, ou portes NON), de la poudre de redstone (fils) et d'autres composants, les joueurs peuvent construire des portes logiques de base.[101] Par exemple, deux torches alimentant le même fil créent une porte NOR. Il est bien connu en logique booléenne que la porte NOR est **fonctionnellement complète**, ce qui signifie que n'importe quel autre circuit logique (ET, OU, etc.) peut être construit uniquement à partir de portes NOR.[102] En assemblant ces portes, les joueurs ont réussi à construire des circuits de plus en plus complexes, allant de simples additionneurs à des unités de mémoire (loquets, bascules) et même des processeurs 8-bits complets et des machines de Turing universelles.[102] Bien que ces ordinateurs virtuels soient extrêmement lents et limités par la taille du monde du jeu et les performances de l'ordinateur hôte, ils démontrent que le système de règles de la redstone possède les ingrédients nécessaires (sélection conditionnelle et mémoire) pour la Turing-complétude.[41]

Factorio et la Logique des Convoyeurs et des Trains

Factorio est un jeu de gestion et d'automatisation où les joueurs construisent des usines complexes. La Turing-complétude y émerge de plusieurs manières. La plus directe est l'utilisation du "réseau de circuits", qui permet de connecter des machines et des convoyeurs avec des fils logiques pour lire leur contenu et contrôler leur comportement, permettant ainsi la construction directe de circuits logiques.[105] Une approche plus créative et visuelle est celle du "Turing Train". Dans cette implémentation, un long chemin de fer droit représente le ruban de la machine de Turing. Des gares espacées régulièrement le long de la voie correspondent aux cases du ruban, et un coffre à chaque gare contient un objet représentant le symbole de cette case. Une locomotive à double tête, capable de se déplacer dans les deux sens, joue le rôle de la tête de lecture/écriture. Le "programme" est encodé dans un réseau de combinateurs logiques situé à l'écart, qui lit l'état du train et le contenu du coffre de la gare actuelle, puis dicte au train sa prochaine destination (la gare de gauche, la gare de droite, ou rester sur place) et quelle action effectuer sur le coffre.[106] Cette construction est une

magnifique incarnation physique et dynamique du modèle abstrait de Turing.

Magic: The Gathering, la Machine Déterministe

Le jeu de cartes à collectionner *Magic: The Gathering* est réputé pour ses règles complexes et les interactions innombrables entre ses milliers de cartes uniques. Des chercheurs ont prouvé que le jeu est Turing-complet d'une manière particulièrement fascinante. Ils ont démontré qu'il est possible de construire deux decks et d'établir un état de jeu à partir duquel la partie se déroule de manière entièrement déterministe, sans qu'aucun des deux joueurs n'ait de choix à faire.[108] Dans cette configuration, les "capacités déclenchées" des cartes (des effets qui se produisent automatiquement lorsque certaines conditions sont remplies, du type "Quand X arrive, faites Y") agissent comme les instructions conditionnelles d'un programme. L'état du jeu — les créatures sur le champ de bataille, leurs types, le nombre de jetons, etc. — sert de mémoire, encodant à la fois le ruban et l'état interne de la machine de Turing. Le déroulement forcé du jeu simule l'exécution de la machine étape par étape. La preuve de la Turing-complétude va plus loin : elle montre que déterminer le vainqueur d'une telle partie est équivalent au problème de l'arrêt, ce qui en fait le premier jeu joué dans le monde réel dont il a été prouvé que la stratégie optimale est non-calculable.[108]

La Machine de Turing en LEGO de l'ENS de Lyon

Pour célébrer le centenaire de la naissance d'Alan Turing en 2012, des étudiants de l'École Normale Supérieure de Lyon ont entrepris le projet RubENS : construire une machine de Turing entièrement avec des briques LEGO.[111] L'objectif était purement pédagogique : créer un outil visuel et simple pour expliquer les fondements du calcul à un large public.[112] La machine est une merveille d'ingénierie mécanique. Elle n'utilise aucune électronique ; sa seule source d'énergie est de l'air comprimé qui actionne des pistons pneumatiques LEGO. Le ruban est une chaîne de briques LEGO de différentes couleurs, et la tête de lecture est un capteur de couleur optique LEGO qui déclenche des mécanismes pour déplacer le ruban et écrire de nouvelles couleurs. En rendant le processus de calcul entièrement visible et tangible, sans "boîte noire" électronique, cette machine est peut-être l'incarnation physique la plus pure et la plus fidèle du modèle abstrait original de Turing, démontrant que le calcul est fondamentalement un processus mécanique de manipulation de symboles.[112]

Ouvrages cités

1. Hilbert's tenth problem - Wikipedia, dernier accès : octobre 31, 2025, https://en.wikipedia.org/wiki/Hilbert%27s_tenth_problem
2. Hilbert's Tenth Problem is Unsolvable Author(s): Martin Davis Source: The American Mathematical Monthly, Vol. 80, No. 3 (Mar., 1, dernier accès : octobre 31, 2025, http://math.uchicago.edu/~shmuel/lg-readings/martin%20davis,%20hilbert%2010.pdf
3. Alan Turing Publishes "On Computable Numbers," Describing What ..., dernier accès : octobre 31, 2025, https://www.historyofinformation.com/detail.php?id=619
4. History of the Church–Turing thesis - Wikipedia, dernier accès : octobre 31, 2025, https://en.wikipedia.org/wiki/History_of_the_Church%E2%80%93Turing_thesis
5. Entscheidungsproblem - Wikipedia, dernier accès : octobre 31, 2025, https://en.wikipedia.org/wiki/Entscheidungsproblem
6. The Entscheidungsproblem and Alan Turing - Georgia College, dernier accès : octobre 31, 2025, https://www.gcsu.edu/sites/files/page-assets/node-808/attachments/brodkorb.pdf
7. Touring Turing | American Scientist, dernier accès : octobre 31, 2025, https://www.americanscientist.org/article/touring-turing
8. The Church-Turing Thesis > The Rise and Fall of the Entscheidungsproblem (Stanford Encyclopedia of Philosophy), dernier accès : octobre 31, 2025, https://plato.stanford.edu/entries/church-

turing/decision-problem.html
9. From Hilbert to Turing and beyond, dernier accès : octobre 31, 2025, https://people.scs.carleton.ca/~bertossi/talks/hilbTur08.pdf
10. Turing's Analysis of Hilbert's "Entscheidungsproblem", dernier accès : octobre 31, 2025, https://sites.math.rutgers.edu/~cherlin/History/Papers2002/turing.html
11. Alan Turing and the Countability of Computable Numbers – Feature Column - Math Voices, dernier accès : octobre 31, 2025, https://mathvoices.ams.org/featurecolumn/2021/12/01/alan-turing-computable-numbers/
12. Turing Machines - Stanford Encyclopedia of Philosophy, dernier accès : octobre 31, 2025, https://plato.stanford.edu/entries/turing-machine/
13. Turing's proof - Wikipedia, dernier accès : octobre 31, 2025, https://en.wikipedia.org/wiki/Turing%27s_proof
14. 1 Definition of a Turing machine - CS@Cornell, dernier accès : octobre 31, 2025, https://www.cs.cornell.edu/courses/cs4820/2018sp/handouts/turingm.pdf
15. CS 4810 » Lecture 11: Turing Machines, dernier accès : octobre 31, 2025, https://www.dsteurer.org/toc13/lectures/11/
16. What are the components of a Turing machine and how do they contribute to its functionality? - EITCA Academy, dernier accès : octobre 31, 2025, https://eitca.org/cybersecurity/eitc-is-cctf-computational-complexity-theory-fundamentals/turing-machines/definition-of-tms-and-related-language-classes/examination-review-definition-of-tms-and-related-language-classes/what-are-the-components-of-a-turing-machine-and-how-do-they-contribute-to-its-functionality/
17. Turing machine - Wikipedia, dernier accès : octobre 31, 2025, https://en.wikipedia.org/wiki/Turing_machine
18. formal definition of a Turing machine - PlanetMath.org, dernier accès : octobre 31, 2025, https://planetmath.org/formaldefinitionofaturingmachine
19. 3.1.2: Representing Turing Machines - Humanities LibreTexts, dernier accès : octobre 31, 2025, https://human.libretexts.org/Bookshelves/Philosophy/Logic_and_Reasoning/Sets%2C_Logic%2C_Computation_(Zach)/03%3A_III-_Turing_Machines/3.01%3A_Turing_Machine_Computations/3.1.02%3A_Representing_Turing_Machines
20. Turing Machines - a formal definition, dernier accès : octobre 31, 2025, http://www.doc.ic.ac.uk/~mrc/Computability%20&%20Complexity/Lectures/C240Lec3.pdf
21. Formal Languages, Automata and Computation Turing Machines - andrew.cmu.ed, dernier accès : octobre 31, 2025, https://www.andrew.cmu.edu/user/ko/pdfs/lecture-12.pdf
22. Turing Machines, dernier accès : octobre 31, 2025, https://www.cs.odu.edu/~toida/nerzic/390teched/tm/tm.html-not-used
23. 1 Definition of a Turing machine - Cornell: Computer Science, dernier accès : octobre 31, 2025, https://www.cs.cornell.edu/courses/cs4820/2010sp/handouts/turingm.pdf
24. Turing Machine for addition - GeeksforGeeks, dernier accès : octobre 31, 2025, https://www.geeksforgeeks.org/theory-of-computation/turing-machine-addition/
25. 2013-10-29: Addition on Turing Machines - Jay McCarthy, dernier accès : octobre 31, 2025, https://jeapostrophe.github.io/2013-10-29-tmadd-post.html
26. Church–Turing thesis - Wikipedia, dernier accès : octobre 31, 2025, https://en.wikipedia.org/wiki/Church%E2%80%93Turing_thesis
27. The Church-Turing Thesis - Stanford Encyclopedia of Philosophy, dernier accès : octobre 31, 2025, https://plato.stanford.edu/archives/fall1997/entries/church-turing/
28. Are Turing machines and lambda calculus equivalent in ..., dernier accès : octobre 31, 2025,

https://eitca.org/cybersecurity/eitc-is-cctf-computational-complexity-theory-fundamentals/turing-machines/definition-of-tms-and-related-language-classes/are-turing-machines-and-lambda-calculus-equivalent-in-computational-power/
29. I'm still a little confused. It seems like Turing came up with something that wo... | Hacker News, dernier accès : octobre 31, 2025, https://news.ycombinator.com/item?id=28540583
30. Church-Turing Thesis -- from Wolfram MathWorld, dernier accès : octobre 31, 2025, https://mathworld.wolfram.com/Church-TuringThesis.html
31. AlanTuring.net The Turing-Church Thesis, dernier accès : octobre 31, 2025, https://www.alanturing.net/turing_archive/pages/reference%20articles/The%20Turing-Church%20Thesis.html
32. plato.stanford.edu, dernier accès : octobre 31, 2025, https://plato.stanford.edu/archives/fall2020/entries/church-turing/#:~:text=1.-,The%20Thesis%20and%20its%20History,not%20carry%20their%20everyday%20meaning.
33. Turing Invents the Universal Turing Machine | Research Starters - EBSCO, dernier accès : octobre 31, 2025, https://www.ebsco.com/research-starters/computer-science/turing-invents-universal-turing-machine
34. Universal Turing machine - Wikipedia, dernier accès : octobre 31, 2025, https://en.wikipedia.org/wiki/Universal_Turing_machine
35. Universal Turing Machine - University of Alberta Dictionary of Cognitive Science, dernier accès : octobre 31, 2025, http://www.bcp.psych.ualberta.ca/~mike/Pearl_Street/Dictionary/contents/U/utm.html
36. Alan Turing's Universal Computing Machine | by calhoun137 - Medium, dernier accès : octobre 31, 2025, https://medium.com/@calhoun137/alan-turings-universal-computing-machine-be69c052c6fd
37. Lecture 8 - Universal turing machines - ECE374-B Archive, dernier accès : octobre 31, 2025, https://ecealgo.com/lectures/Lec08.html
38. CSE 460 Universal Turing Machine and the Simulated Turing machine, dernier accès : octobre 31, 2025, https://cse.msu.edu/~pramanik/teaching/courses/cse460/12f/lectures/L8_TM/UnivTM.pdf
39. Universal Turing Machine - MIT, dernier accès : octobre 31, 2025, https://web.mit.edu/manoli/turing/www/turing.html
40. Universal Turing Machine: A Model for all Computational Problems, dernier accès : octobre 31, 2025, https://www.rroij.com/open-access/universal-turing-machine-a-model-for-allcomputational-problems.pdf
41. Turing Completeness, dernier accès : octobre 31, 2025, https://www.cs.odu.edu/~zeil/cs390/latest/Public/turing-complete/index.html
42. Turing completeness - Wikipedia, dernier accès : octobre 31, 2025, https://en.wikipedia.org/wiki/Turing_completeness
43. Turing complete - Simple English Wikipedia, the free encyclopedia, dernier accès : octobre 31, 2025, https://simple.wikipedia.org/wiki/Turing_complete
44. Exploring Turing Completeness | Lenovo US, dernier accès : octobre 31, 2025, https://www.lenovo.com/us/en/glossary/what-is-turing-completeness/
45. What Is Turing Completeness? - ITU Online IT Training, dernier accès : octobre 31, 2025, https://www.ituonline.com/tech-definitions/what-is-turing-completeness/
46. Does Turing completeness mean a thing? - Stack Overflow, dernier accès : octobre 31, 2025, https://stackoverflow.com/beta/discussions/77996304/does-turing-completeness-mean-a-thing
47. Why are some programming languages Turing complete but lack some abilities of other languages? - Computer Science Stack Exchange, dernier accès : octobre 31, 2025, https://cs.stackexchange.com/questions/63961/why-are-some-programming-languages-turing-complete-but-lack-some-abilities-of-ot

48. "Since Python is written in C, absolutely everything that can be done in Python, can be done in C" - is this statement true? : r/learnpython - Reddit, dernier accès : octobre 31, 2025, https://www.reddit.com/r/learnpython/comments/k2oxqh/since_python_is_written_in_c_absolutely/
49. Why is the idea of an uncomputable number a thing? : r/askmath - Reddit, dernier accès : octobre 31, 2025, https://www.reddit.com/r/askmath/comments/1kscguj/why_is_the_idea_of_an_uncomputable_number_a_thing/
50. CSE 311 Lecture 28: Undecidability of the Halting Problem - Washington, dernier accès : octobre 31, 2025, https://courses.cs.washington.edu/courses/cse311/20sp/doc/lecture28.pdf
51. Cantor's diagonal argument - Wikipedia, dernier accès : octobre 31, 2025, https://en.wikipedia.org/wiki/Cantor%27s_diagonal_argument
52. Cantor's Diagonal Proof and various misconceptions as to what it actually proves - Logic, dernier accès : octobre 31, 2025, https://jamesrmeyer.com/infinite/diagonal-proof
53. Halting problem - Wikipedia, dernier accès : octobre 31, 2025, https://en.wikipedia.org/wiki/Halting_problem
54. Undecidable problems | AP CSP (article) | Khan Academy, dernier accès : octobre 31, 2025, https://www.khanacademy.org/computing/ap-computer-science-principles/algorithms-101/solving-hard-problems/a/undecidable-problems
55. Undecidability, dernier accès : octobre 31, 2025, https://www.cs.rochester.edu/u/nelson/courses/csc_173/computability/undecidable.html
56. Lesson 7: Halting Problem and Undecidable Problems | BTU, dernier accès : octobre 31, 2025, https://btu.edu.ge/wp-content/uploads/2023/07/Lesson-7_-Halting-Problem-and-Undecidable-Problems.pdf
57. Rice's theorem - Wikipedia, dernier accès : octobre 31, 2025, https://en.wikipedia.org/wiki/Rice%27s_theorem
58. Halting Problem | Brilliant Math & Science Wiki, dernier accès : octobre 31, 2025, https://brilliant.org/wiki/halting-problem/
59. CS245 Undecidability, dernier accès : octobre 31, 2025, https://cs.uwaterloo.ca/~a23gao/cs245_f17/notes/undecidability_solutions.pdf
60. The Halting Problem: The Unsolvable Problem - YouTube, dernier accès : octobre 31, 2025, https://www.youtube.com/watch?v=VyHbd6sx5Po
61. How do we know The Halting Problem isn't only impossible to implement for itself? - Reddit, dernier accès : octobre 31, 2025, https://www.reddit.com/r/compsci/comments/gi57ku/how_do_we_know_the_halting_problem_isnt_only/
62. Post correspondence problem - Wikipedia, dernier accès : octobre 31, 2025, https://en.wikipedia.org/wiki/Post_correspondence_problem
63. Post Correspondence Problem - Tutorials Point, dernier accès : octobre 31, 2025, https://www.tutorialspoint.com/automata_theory/post_correspondence_problem.htm
64. Post Correspondence Problem - InterviewBit, dernier accès : octobre 31, 2025, https://www.interviewbit.com/blog/post-correspondence-problem/
65. 6.8 The Post Correspondence Problem - CIS UPenn, dernier accès : octobre 31, 2025, https://www.cis.upenn.edu/~jean/gbooks/PCPh04.pdf
66. Exponential Curves: Supercomputer to Smartphone to Singularity | Turn the Lens Ep39, dernier accès : octobre 31, 2025, https://www.turnthelenspodcast.com/episode/exponential-curves-supercomputer-to-smartphone-to-singularity-turn-the-lens-ep39
67. Fast-forward — comparing a 1980s supercomputer to a modern ..., dernier accès : octobre 31, 2025,

https://blog.adobe.com/en/publish/2022/11/08/fast-forward-comparing-1980s-supercomputer-to-modern-smartphone
68. Church-Turing Thesis - Quantum Computing Codex, dernier accès : octobre 31, 2025, https://qc-at-davis.github.io/QCC/Classical-Computation/Church-Turing-Thesis/Church-Turing-Thesis.html
69. The Efficient Church-Turing Thesis - Computational Complexity, dernier accès : octobre 31, 2025, https://blog.computationalcomplexity.org/2006/12/efficient-church-turing-thesis.html
70. Lecture 1, Tues Jan 17: Course Intro, Church ... - Scott Aaronson, dernier accès : octobre 31, 2025, https://www.scottaaronson.com/qclec/1.pdf
71. Dissipative Quantum Church-Turing Theorem | Phys. Rev. Lett., dernier accès : octobre 31, 2025, https://link.aps.org/doi/10.1103/PhysRevLett.107.120501
72. Quantum computing - Wikipedia, dernier accès : octobre 31, 2025, https://en.wikipedia.org/wiki/Quantum_computing
73. The Church-Turing thesis in a quantum world, dernier accès : octobre 31, 2025, https://people.maths.bris.ac.uk/~csxam/presentations/turingtalk.pdf
74. quantumcomputing.stackexchange.com, dernier accès : octobre 31, 2025, https://quantumcomputing.stackexchange.com/questions/30155/what-are-some-examples-of-uncomputability-with-quantum-computers#:~:text=Quantum%20computers%20cannot%20solve%20uncomputable%20problems.&text=However%2C%20there%20are%20many%20other,inefficiently)%20on%20a%20classical%20computer.
75. Are there any problems in Computer Science that *can't* be solved with classical computers but *can* be solved with quantum computers? : r/askscience - Reddit, dernier accès : octobre 31, 2025, https://www.reddit.com/r/askscience/comments/2uo4ld/are_there_any_problems_in_computer_science_that/
76. Why is a quantum computer not capable of solving more problems than a classical computer? [duplicate], dernier accès : octobre 31, 2025, https://cs.stackexchange.com/questions/44823/why-is-a-quantum-computer-not-capable-of-solving-more-problems-than-a-classical
77. Unsolved math problems that can only be solved using a quantum computer? - Reddit, dernier accès : octobre 31, 2025, https://www.reddit.com/r/computerscience/comments/miory3/unsolved_math_problems_that_can_only_be_solved/
78. Could quantum computing solve problems, such as the Halting problem? - Quora, dernier accès : octobre 31, 2025, https://www.quora.com/Could-quantum-computing-solve-problems-such-as-the-Halting-problem
79. complexity theory - What are some examples of uncomputability with ..., dernier accès : octobre 31, 2025, https://quantumcomputing.stackexchange.com/questions/30155/what-are-some-examples-of-uncomputability-with-quantum-computers
80. Quantum algorithm - Wikipedia, dernier accès : octobre 31, 2025, https://en.wikipedia.org/wiki/Quantum_algorithm
81. The Quantum-Extended Church-Turing Thesis in Quantum Field Theory - arXiv, dernier accès : octobre 31, 2025, https://arxiv.org/pdf/2309.09000
82. Quantum complexity theory - Wikipedia, dernier accès : octobre 31, 2025, https://en.wikipedia.org/wiki/Quantum_complexity_theory
83. Quantum theory, the Church–Turing principle and the universal quantum computer | Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences - Journals, dernier accès : octobre 31, 2025, https://royalsocietypublishing.org/doi/10.1098/rspa.1985.0070
84. The Church-Turing Thesis (Stanford Encyclopedia of Philosophy), dernier accès : octobre 31, 2025, https://plato.stanford.edu/entries/church-turing/

85. [1102.1612] The physical Church-Turing thesis and the principles of quantum theory - arXiv, dernier accès : octobre 31, 2025, https://arxiv.org/abs/1102.1612
86. Hypercomputation - Wikipedia, dernier accès : octobre 31, 2025, https://en.wikipedia.org/wiki/Hypercomputation
87. Can somebody explain this to me?: The computability of the laws of physics and hypercomputation - LessWrong, dernier accès : octobre 31, 2025, https://www.lesswrong.com/posts/Rc2Ymaj8sfaG4Crt8/can-somebody-explain-this-to-me-the-computability-of-the
88. Hypercomputation and the Physical Church-Turing Thesis | The British Journal for the Philosophy of Science: Vol 54, No 2, dernier accès : octobre 31, 2025, https://www.journals.uchicago.edu/doi/10.1093/bjps/54.2.181
89. Why is hypercomputation contested? - Philosophy Stack Exchange, dernier accès : octobre 31, 2025, https://philosophy.stackexchange.com/questions/110920/why-is-hypercomputation-contested
90. Hypercomputation and The Limits of Computing - Medium, dernier accès : octobre 31, 2025, https://medium.com/@noah_h/hypercomputation-and-the-limits-of-computing-4e10c533880b
91. The Church-Turing Thesis: Breaking the Myth | Lambda the Ultimate, dernier accès : octobre 31, 2025, http://lambda-the-ultimate.org/node/1038
92. Applicability of Church-Turing thesis to interactive models of computation, dernier accès : octobre 31, 2025, https://cstheory.stackexchange.com/questions/12377/applicability-of-church-turing-thesis-to-interactive-models-of-computation
93. The Computational Theory of Mind - Cambridge University Press, dernier accès : octobre 31, 2025, https://www.cambridge.org/core/elements/computational-theory-of-mind/A56A0340AD1954C258EF6962AF450900
94. Computational theory of mind - Wikipedia, dernier accès : octobre 31, 2025, https://en.wikipedia.org/wiki/Computational_theory_of_mind
95. Computational Theory of Mind | Internet Encyclopedia of Philosophy, dernier accès : octobre 31, 2025, https://iep.utm.edu/computational-theory-of-mind/
96. The Computational Theory of Mind (Stanford Encyclopedia of ..., dernier accès : octobre 31, 2025, https://plato.stanford.edu/entries/computational-mind/
97. Focus Article Article title: Computational Modeling of the Mind: What Role for Mental Representation? - Philosophy - UCLA, dernier accès : octobre 31, 2025, https://philosophy.ucla.edu/wp-content/uploads/2016/08/Computation-Modeling.pdf
98. Lucas-Penrose Argument about Gödel's Theorem | Internet ..., dernier accès : octobre 31, 2025, https://iep.utm.edu/lp-argue/
99. Penrose–Lucas argument - Wikipedia, dernier accès : octobre 31, 2025, https://en.wikipedia.org/wiki/Penrose%E2%80%93Lucas_argument
100. Lucas and Penrose vs. Computationalism: - Lund University Publications, dernier accès : octobre 31, 2025, https://lup.lub.lu.se/student-papers/record/9061763/file/9061925.pdf
101. Is Minecraft Turing complete? - Quora, dernier accès : octobre 31, 2025, https://www.quora.com/Is-Minecraft-Turing-complete
102. Minecraft is a Turing complete video game - Efe's Blog, dernier accès : octobre 31, 2025, https://efeozbatur.com/minecraft-is-a-turing-complete-video-game/
103. Redstone (in theory) - Reddit, dernier accès : octobre 31, 2025, https://www.reddit.com/r/redstone/comments/1d6fl2n/redstone_in_theory/
104. Universal Turing Machine implemented in Minecraft redstone logic - YouTube, dernier accès : octobre 31, 2025, https://www.youtube.com/watch?v=1X21HQphy6I
105. Turing completeness - Factorio Forums, dernier accès : octobre 31, 2025,

https://forums.factorio.com/viewtopic.php?t=12487
106. Turing Trains - Circuits without circuits - YouTube, dernier accès : octobre 31, 2025, https://www.youtube.com/watch?v=b1PlPG2vUjY
107. The Turing Train (aka fully functional Turing machine using trains ..., dernier accès : octobre 31, 2025, https://forums.factorio.com/viewtopic.php?t=125093
108. Magic: the Turing Machine. Magic is a game that has been around ..., dernier accès : octobre 31, 2025, https://medium.com/henngeblog/magic-the-turing-machine-4ea52bb399a3
109. Magic: the Gathering is Turing Complete - toothycat.net, dernier accès : octobre 31, 2025, https://www.toothycat.net/~hologram/Turing/
110. [1904.09828] Magic: The Gathering is Turing Complete - arXiv, dernier accès : octobre 31, 2025, https://arxiv.org/abs/1904.09828
111. The Turing Machine Comes True - Vidéo Dailymotion, dernier accès : octobre 31, 2025, https://www.dailymotion.com/video/xrmfie
112. RUBENS, dernier accès : octobre 31, 2025, http://rubens.ens-lyon.fr/
113. The LEGO Turing machine - CWI Amsterdam, dernier accès : octobre 31, 2025, https://www.cwi.nl/en/stories/the-lego-turing-machine/
114. Turing Machines - COT 6315 - UF CISE, dernier accès : octobre 31, 2025, https://www.cise.ufl.edu/~nemo/cot6315/tm.html