De la logique avant toute chose!
Du programme de Hilbert à la naissance de la science informatique

Xavier Leroy

Conférence Sciences et Société, Nancy, 2024-05-16

Collège de France, chaire de sciences du logiciel
xavier.leroy@college-de-france.fr

ART POÉTIQUE

De la musique avant toute chose,
Et pour cela préfère l'Impair
Plus vague et plus soluble dans l'air,
Sans rien en lui qui pèse ou qui pose.

(Paul Verlaine, 1874)

Un peu de logique

La logique d'Aristote

Un discours dans lequel, certaines choses étant posées, quelque chose d'autre que ces données en résulte nécessairement par le seul fait de ces données.

Exemple : le syllogisme. ✔

Tous les hommes sont mortels.

Socrate est un homme.

Donc Socrate est mortel.

La logique d'Aristote

Un discours dans lequel, certaines choses étant posées, quelque chose d'autre que ces données en résulte nécessairement par le seul fait de ces données.

Exemple : le syllogisme. ✔

Tous les hommes sont mortels.
Socrate est un homme.
Donc Socrate est mortel.

Contre-exemple : le paralogisme (sophisme). ✘

Tous les chats sont mortels.
Socrate est mortel.
Donc Socrate est un chat.

Que signifie ≪en résulte nécessairement≫ ?

Exemple difficile : l'argument ontologique.

(Anselme de Cantorbéry, 1077)

> *L'idée de Dieu exprime celle d'un Être parfait.*
> *Un Être qui n'existe pas ne peut être parfait.*
> *Donc Dieu existe nécessairement.*

Longuement étudié par Descartes, Spinoza, Leibniz, …

Méticuleusement réfuté par Kant.

Démontré par Gödel vers 1970
(en logique modale et avec une définition bien choisie de ≪parfait≫).

Gottfried Wilhelm Leibniz : le *calculus ratiocinator* (1666–1670)

Quelle: Deutsche Fotothek

Dans *De arte combinatoria*, Leibniz imagine

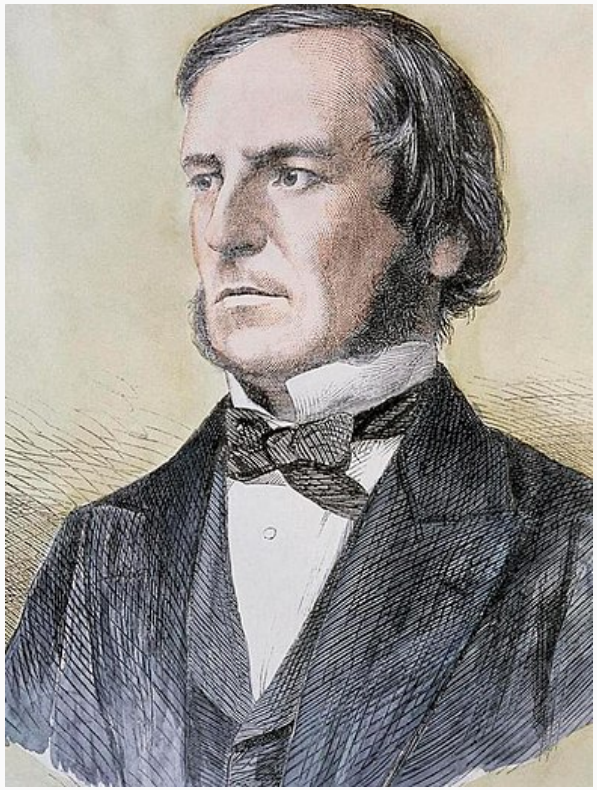
- une langue philosophique universelle, *characteristica universalis*, pour définir les concepts avec la précision des définitions mathématiques ;
- une méthode systématique, voire mécanisable, *calculus ratiocinator*, pour démêler le vrai du faux en appliquant juste des règles de calcul.

Calculemus!

Quando orientur controversiae, non magis disputatione opus erit inter duos philosophus, quam inter duos computistas. Sufficiet enim calamos in manus sumere sedereque ad abacos, et sibi mutuo (accito si placet amico) dicere : calculemus.

Alors, il ne sera plus besoin entre deux philosophes de discussions plus longues qu'entre deux comptables, puisqu'il suffira qu'ils saisissent leur plume, qu'ils s'asseyent à leur table de calcul (en faisant appel, s'ils le souhaitent, à un ami) et qu'ils se disent l'un à l'autre : «calculons!»

George Boole : les lois de la pensée (1847–1854)



AN INVESTIGATION

OF

THE LAWS OF THOUGHT,

ON WHICH ARE FOUNDED

THE MATHEMATICAL THEORIES OF LOGIC
AND PROBABILITIES.

BY

GEORGE BOOLE, LL.D.

PROFESSOR OF MATHEMATICS IN QUEEN'S COLLEGE, CORK.

LONDON:
WALTON AND MABERLY,
UPPER GOWER-STREET, AND IVY-LANE, PATERNOSTER-ROW.
CAMBRIDGE: MACMILLAN AND CO.
1854.

George Boole : les lois de la pensée (1847–1854)

La première formalisation du calcul des propositions :

- énoncés élémentaires («il pleut», «$2 + 2 = 4$», etc.)
- reliés par des connecteurs logiques $\wedge$, $\vee$, $\neg$, etc.

Définition précise des connecteurs par leurs tables de vérité

x	$\neg x$
0	1
1	0

x	y	$x \wedge y$	$x \vee y$
0	0	0	0
1	0	0	1
0	1	0	1
1	1	1	1

ou en termes d'autres connecteurs

$$x \Rightarrow y \stackrel{def}{=} y \vee \neg x \qquad x \Leftrightarrow y \stackrel{def}{=} (x \wedge y) \vee (\neg x \wedge \neg y)$$

L'algèbre booléenne

Une riche structure algébrique avec de nombreuses lois :

$$x \vee (y \vee z) = (x \vee y) \vee z \qquad x \vee y = y \vee x$$

$$x \wedge (y \wedge z) = (x \wedge y) \wedge z \qquad x \wedge y = y \wedge x \qquad x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$$

$$x \vee 0 = x \qquad x \wedge 1 = x \qquad x \wedge 0 = 0$$

$$x \vee 1 = 1 \qquad x \vee x = x \qquad x \wedge x = x$$

$$x \wedge (x \vee y) = x \qquad x \vee (x \wedge y) = x \qquad x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$$

$$x \wedge \neg x = 0 \qquad x \vee \neg x = 1 \qquad \neg\neg x = x$$

$$\neg x \wedge \neg y = \neg(x \vee y) \qquad \neg x \vee \neg y = \neg(x \wedge y)$$

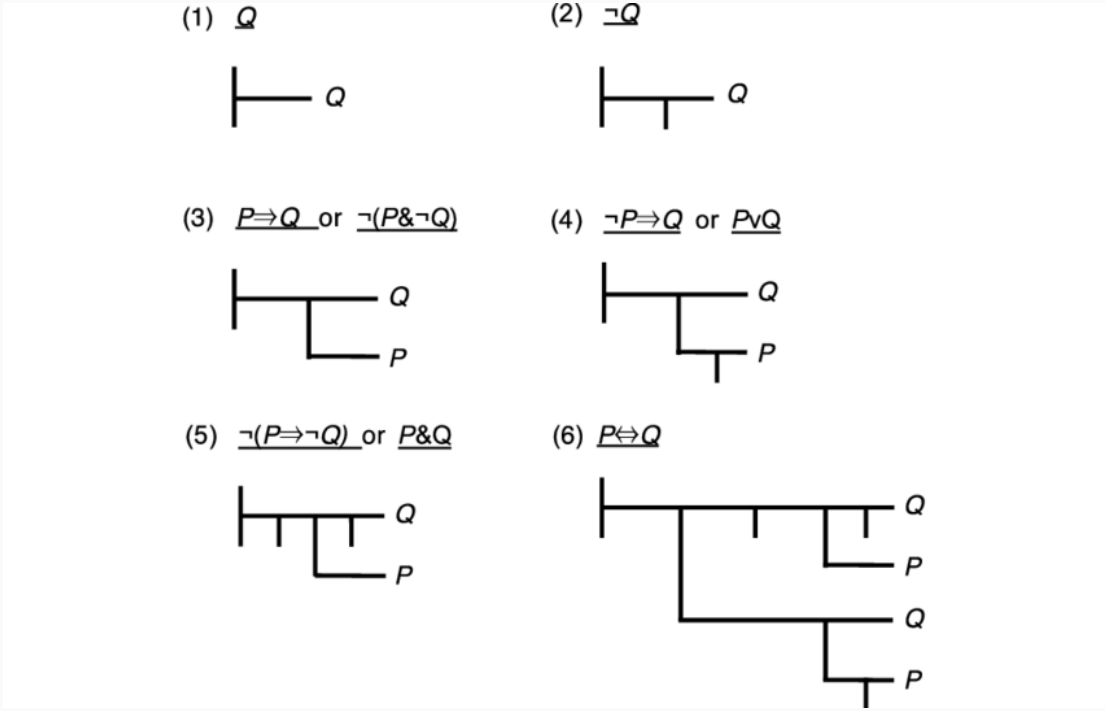
Calculemus !

S'il pleut je prends mon parapluie ;
s'il ne pleut pas, je prends mon parapluie.

$\rightsquigarrow$

Je prends mon parapluie.

Gottlob Frege : l'Idéographie (1879) ; les Fondements de l'arithmétique (1884)



Gottlob Frege : l'Idéographie (1879) ; les Fondements de l'arithmétique (1884)

La première formalisation du calcul des prédicats :

- propositions impliquant des variables $x, y, f, \ldots$
- variables pouvant être quantifiées (pour tout, il existe)

Exemple de prédicat :

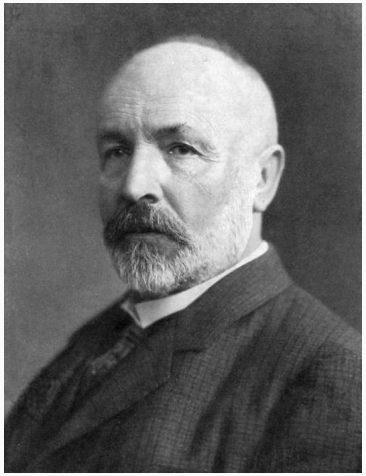
$$lim(u, \ell) \stackrel{def}{=} \forall \varepsilon,\ \exists p,\ \forall n \geq p,\ |u_n - \ell| \leq \varepsilon$$

De nombreuses lois, p.ex.

$$(\exists x, P) \Leftrightarrow \neg(\forall x, \neg P) \qquad (\forall x, P \wedge Q) \Leftrightarrow (\forall x, P) \wedge (\forall x, Q)$$

Les débuts de la théorie des ensembles

Bernard Bolzano (1837)

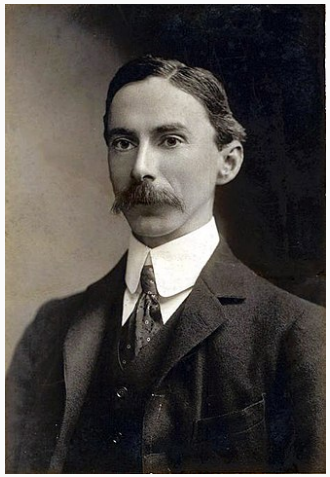


Georg Cantor (1880)

Des liens entre le calcul des prédicats et la théorie naissante des ensembles :

$$\begin{aligned} \text{prédicat } P(x) &\rightsquigarrow \text{ensemble } \{x \mid P(x)\} \\ \text{ensemble } A &\rightsquigarrow \text{prédicat d'appartenance } x \in A \\ \text{égalité d'ensembles } A = B &\rightsquigarrow \text{équivalence } \forall x,\ x \in A \Leftrightarrow x \in B \end{aligned}$$

Le paradoxe de Russell (1901)



Bertrand Russell

Ernst Zermelo

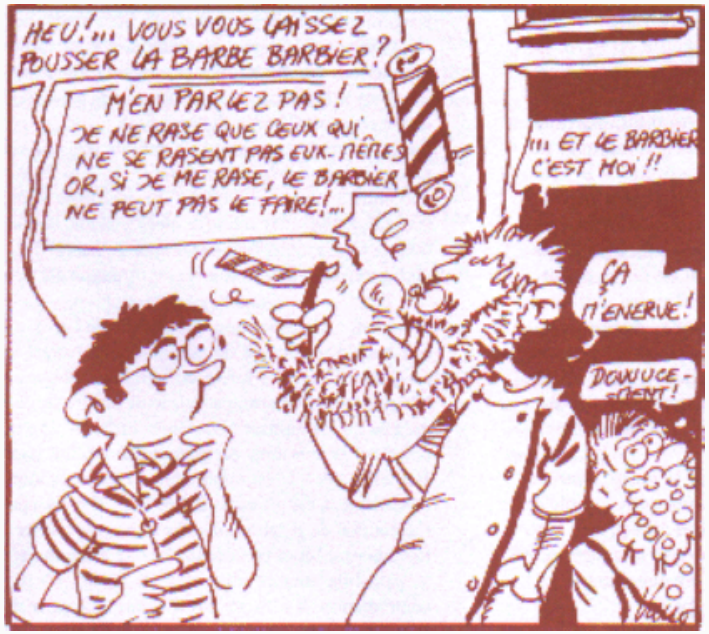
Soit A l'ensemble de tous les ensembles n'appartenant pas à eux-même :

$$A \stackrel{def}{=} \{x \mid x \notin x\}$$

Si $A \in A$, par définition de A, on a $A \notin A$.
Si $A \notin A$, par définition de A, on a $A \in A$.

Aussi connu comme le paradoxe du barbier

Le barbier du village rase toutes les personnes qui ne se rasent pas elles-même. Qui rase le barbier ?



Incohérence de la théorie des ensembles naïve

Le calcul des prédicats de Frege / la théorie des ensembles de Cantor sont donc incohérentes car elles démontrent l'absurde :

de la contradiction $A \in A \wedge \neg A \in A$,

on déduit l'absurde 0 (la proposition toujours fausse);

de l'absurde 0,

on déduit n'importe quel énoncé P.

La réaction de Frege

Addendum au volume 2 des *Fondements de l'arithmétique*.

> *Il ne peut rien arriver de plus malheureux à un homme de science que de voir un des fondements de son édifice ébranlé une fois son œuvre achevée. Telle est la situation où m'a placé une lettre de M. Bertrand Russell quand ce volume était sous presse.*
>
> *L'effondrement d'une de mes lois, auquel aboutit le Paradoxe de Russell, semble miner non seulement les fondements de mon Arithmétique mais aussi les seuls fondements possibles de l'Arithmétique en tant que telle.*

Réparer la théorie des ensembles

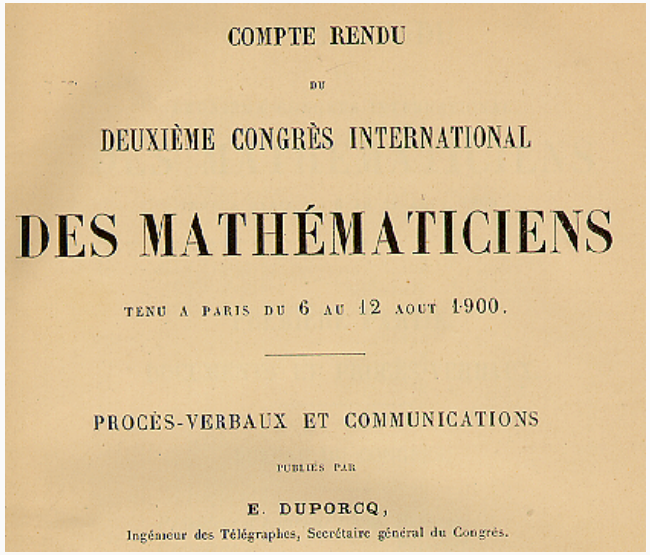
Deux ≪correctifs≫ qui empêchent le paradoxe de Russell :

- La théorie des ensembles de Zermelo-Fraenkel (1908, 1922) : restreindre les ensembles construits par compréhension à $\{x \in A \mid P(x)\}$, où A est un ensemble, au lieu de $\{x \mid P(x)\}$.
- La théorie des types de Russell et Whitehead : un principe ≪le tout est plus grand que la partie≫ qui empêche d'écrire $x \notin x$, formalisé dans *Principia Mathematica* (1910-1913).

La question reste ouverte : ces deux théories sont-elles cohérentes ? ou cachent-elles encore d'autres paradoxes ?

Du programme de Hilbert à la naissance de la science informatique

Le deuxième des 23 problèmes ouverts de David Hilbert (1900)


COMPTE RENDU

DU

DEUXIÈME CONGRÈS INTERNATIONAL

DES MATHÉMATICIENS

TENU A PARIS DU 6 AU 12 AOUT 1900.

PROCÈS-VERBAUX ET COMMUNICATIONS

PUBLIÉS PAR

E. DUPORCQ,

Ingénieur des Télégraphes, Secrétaire général du Congrès.

2. Peut-on prouver la cohérence de l'arithmétique ?
   En d'autres termes, peut-on démontrer que les axiomes de l'arithmétique ne sont pas contradictoires ?

(Démontré par Gerhard Gentzen en 1936, mais avec une preuve par récurrence transfinie plus complexe que ce que Hilbert souhaitait.)

Le programme de Hilbert (1900–1936)

Formaliser l'arithmétique par un système déductif

axiomes : $\vdash x + 0 = x$
$\dots$

règles de déduction : si $\vdash P \Rightarrow Q$ et $\vdash P$, alors $\vdash Q$

et démontrer qu'il satisfait trois propriétés essentielles :

Cohérence : on n'a jamais $\vdash P$ et $\vdash \neg P$
(pas de paradoxes)

Complétude : on a toujours soit $\vdash P$ soit $\vdash \neg P$
(pas de ≪on ne sait pas≫)

Décidabilité (*Entscheidungsproblem*) :
il existe un algorithme qui, étant donné P,
détermine si $\vdash P$ ou si $\vdash \neg P$.

≪Nous devons savoir ; nous saurons≫

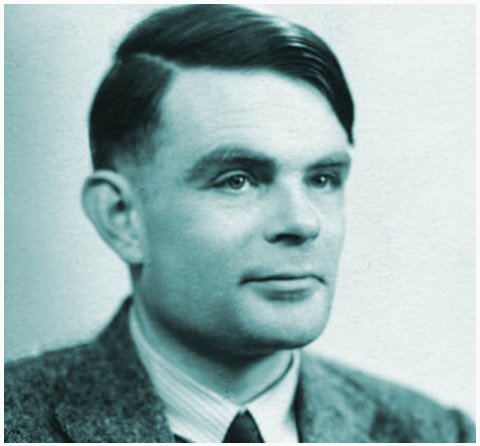
Wir müssen wissen ; wir werden wissen.

(Conclusion d'une conférence de Hilbert radiodiffusée en 1930, et épitaphe sur sa tombe.)

L'effondrement du programme de Hilbert

Kurt Gödel

Alonzo Church



Alan Turing

Kurt Gödel (1931) : premier théorème d'incomplétude.

Toute formalisation cohérente de l'arithmétique contient un énoncé P tel que ni $\vdash P$ ni $\vdash \neg P$.

Alonzo Church (1936) puis **Alan Turing (1936) :**

Indécidabilité du problème de l'arrêt d'un calcul
$\Rightarrow$ non-existence d'un algorithme pour le *Entscheidungsproblem*.

L'indécidabilité du problème de l'arrêt

≪Tant que je termine, je recommence !≫

La machine de Turing et le lambda-calcul de Church sont deux formalismes de calcul où un programme peut prendre un programme comme donnée d'entrée (modulo un codage).

Supposons un programme *TERM* qui décide l'arrêt :

TERM(P) termine sur la valeur 1 si le programme P termine ;

TERM(P) termine sur la valeur 0 si le programme P diverge.

On forme un programme T qui se comporte comme

```
while (TERM(T) == 1) { }
```

Si *TERM*(T) $=$ 1, le programme T diverge.

Si *TERM*(T) $=$ 0, le programme T termine.

Donc *TERM* ne décide pas le problème de l'arrêt.

Une métaphore culinaire

Le problème de la recette (*Rezeptproblem*) :

Toute nourriture peut-elle être produite en suivant une recette de cuisine ?

Si on veut montrer qu'il n'y a pas de recette pour p.ex. l'ambroisie, on va devoir développer une théorie de la ≪cuisinabilité≫ qui est intéressante en elle-même !

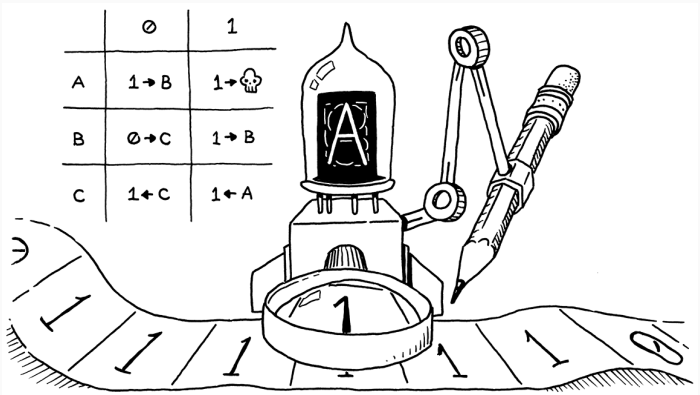
La naissance de la science informatique

Numérotation de Gödel

CODAGE

MACHINE

Machines de Turing



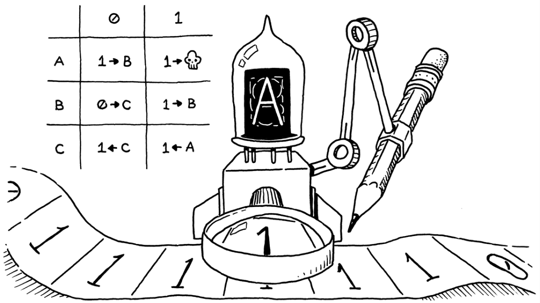
LANGAGE

lambda-calcul de Church

$$\lambda f.\ (\lambda x.\ f\ (x\ x))\ (\lambda x.\ f\ (x\ x))$$

Machines de Turing, lambda-calcul, et calculabilité

Une machine de Turing



Une bande portant des symboles (typiquement : 0, 1, ·).

Un état courant *A*, *B*, *C*, . . .

Une table de transitions (fixée une fois pour toutes)

(état courant, symbole courant) $\rightarrow$

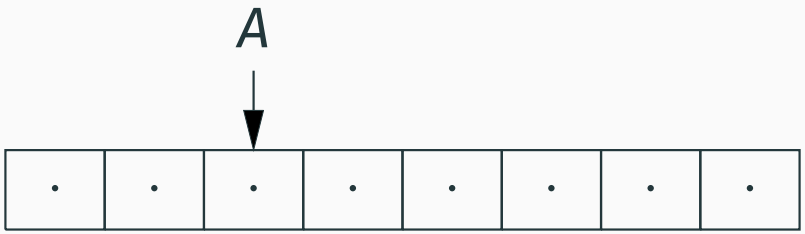
(nouveau symbole, déplacement, état suivant)

Ma première machine : remplir la bande avec 0, 1, 0, 1, ...

Transitions :

	$\cdot$
A	$0, \rightarrow, B$
B	$1, \rightarrow, A$

Exécution :

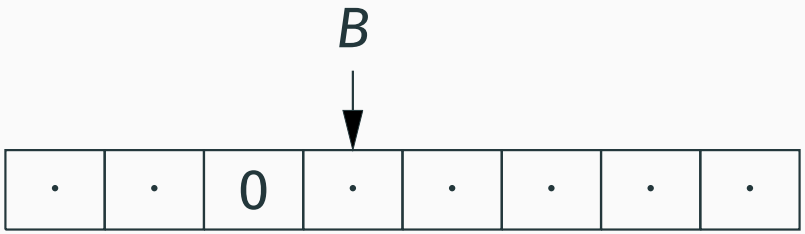


Ma première machine : remplir la bande avec 0, 1, 0, 1, ...

Transitions :

	$\cdot$
A	$0, \rightarrow, B$
B	$1, \rightarrow, A$

Exécution :

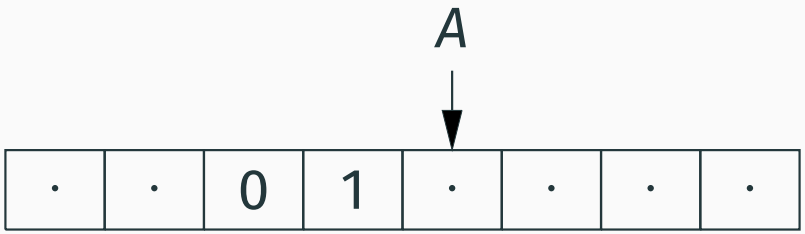


Ma première machine : remplir la bande avec 0, 1, 0, 1, ...

Transitions :

	$\cdot$
A	$0, \rightarrow, B$
B	$1, \rightarrow, A$

Exécution :

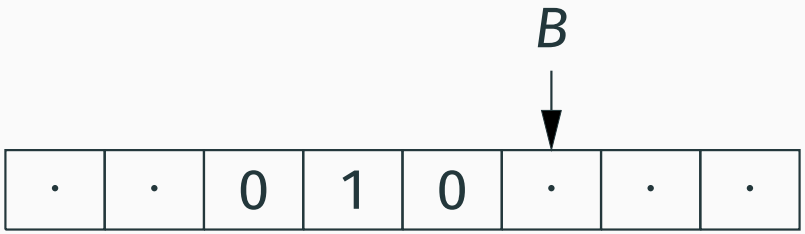


Ma première machine : remplir la bande avec 0, 1, 0, 1, ...

Transitions :

	$\cdot$
A	$0, \rightarrow, B$
B	$1, \rightarrow, A$

Exécution :



Ma première machine : remplir la bande avec 0, 1, 0, 1, ...

Transitions :

	$\cdot$
A	$0, \rightarrow, B$
B	$1, \rightarrow, A$

Exécution :

Incrémenter un entier en base 2

Transitions :

	0	1	·
A	$1, \rightarrow, FIN$	$0, \rightarrow, A$	$1, \rightarrow, FIN$

Exécution :

Notation ≪petit-boutiste≫ : poids faibles en premier.

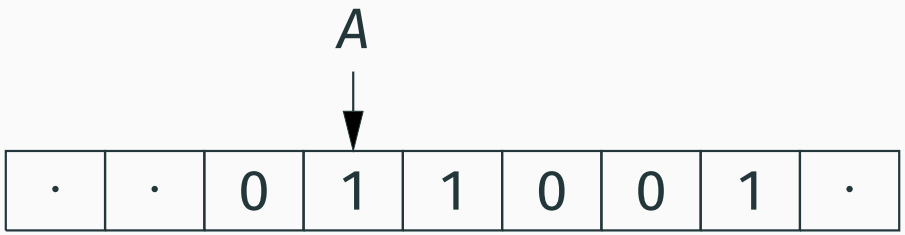
Valeur initiale : `0b100111` = 39,

Incrémenter un entier en base 2

Transitions :

	0	1	·
A	$1, \rightarrow, FIN$	$0, \rightarrow, A$	$1, \rightarrow, FIN$

Exécution :



Notation ≪petit-boutiste≫ : poids faibles en premier.

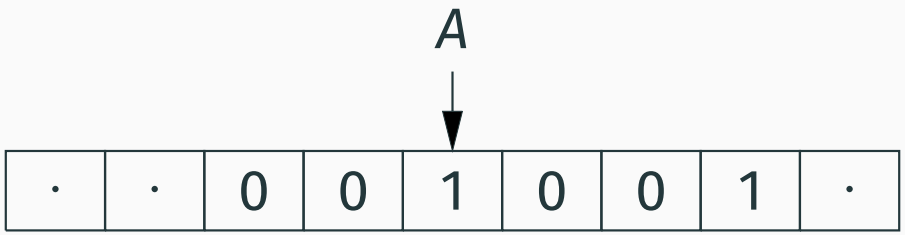
Valeur initiale : `0b100111` = 39,

Incrémenter un entier en base 2

Transitions :

	0	1	·
A	$1, \rightarrow, FIN$	$0, \rightarrow, A$	$1, \rightarrow, FIN$

Exécution :



Notation ≪petit-boutiste≫ : poids faibles en premier.

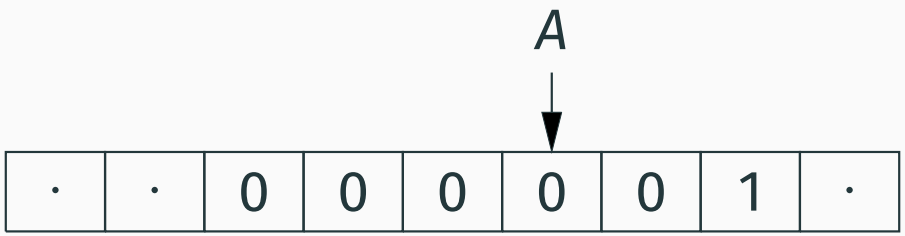
Valeur initiale : `0b100111` = 39,

Incrémenter un entier en base 2

Transitions :

	0	1	·
A	$1, \rightarrow, FIN$	$0, \rightarrow, A$	$1, \rightarrow, FIN$

Exécution :



Notation ≪petit-boutiste≫ : poids faibles en premier.

Valeur initiale : `0b100111` = 39,

Incrémenter un entier en base 2

Transitions :

	0	1	·
A	$1, \rightarrow, FIN$	$0, \rightarrow, A$	$1, \rightarrow, FIN$

Exécution :

Notation ≪petit-boutiste≫ : poids faibles en premier.

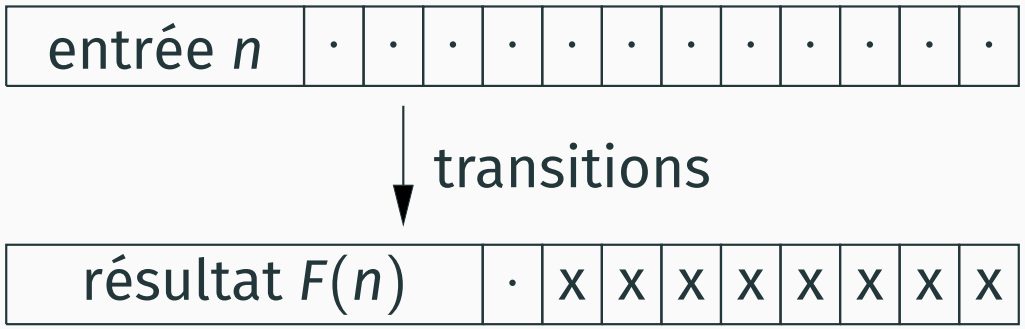
Valeur initiale : `0b100111` = 39, finale : `0b101000` = 40.

Calculer des fonctions entières avec des machines de Turing

Les machines de Turing peuvent effectuer

- les opérations arithmétiques de base ($+$, $-$, $\times$, $\div$, etc.)
- les tests `if ... then ... else`
- l'itération `while ... do`.

Elles peuvent donc calculer des fonctions partielles de $\mathbb{N}$ dans $\mathbb{N}$:



Toutes les fonctions récursives générales (Herbrand, 1931 ; Gödel, 1934 ; Kleene, 1936) peuvent être calculées par une machine de Turing.

Le lambda-calcul

Church, vers 1930 : une notation unifiée pour les fonctions.

$$\lambda x.e \overset{def}{=} \text{la fonction qui à } x \text{ associe } e$$

(La notation de Bourbaki $x \mapsto e$ apparaît bien plus tard.)

Règle de calcul évidente : $(\lambda x.\ x + x)\ (2)$ se simplifie en $2 + 2$.

Le lambda-calcul

Church, vers 1930 : une notation unifiée pour les fonctions.

$$\lambda x.e \overset{def}{=} \text{la fonction qui à } x \text{ associe } e$$

(La notation de Bourbaki $x \mapsto e$ apparaît bien plus tard.)

Règle de calcul évidente : $(\lambda x.\ x + x)\ (2)$ se simplifie en $2 + 2$.

Church, 1932, 1933 : une logique (calcul des prédicats d'ordre supérieur) basée sur la lambda-notation.

Rapidement abandonnée car complètement incohérente.

Le lambda-calcul

Church, vers 1930 : une notation unifiée pour les fonctions.

$$\lambda x.e \overset{def}{=} \text{la fonction qui à } x \text{ associe } e$$

(La notation de Bourbaki $x \mapsto e$ apparaît bien plus tard.)

Règle de calcul évidente : $(\lambda x.\ x + x)\ (2)$ se simplifie en $2 + 2$.

Church, 1932, 1933 : une logique (calcul des prédicats d'ordre supérieur) basée sur la lambda-notation.

Rapidement abandonnée car complètement incohérente.

Church, Kleene et Rosser, 1935 : le lambda-calcul ≪pur≫.

On enlève tous les connecteurs logiques et on ne garde que les fonctions !

Le lambda-calcul pur

Un lambda-terme M est ...

$$\begin{array}{rll} M ::= & x & \text{ou bien une variable } x, y, z, \ldots \\ | & \lambda x.\, M & \text{ou bien une définition de fonction} \\ | & M_1\, M_2 & \text{ou bien une application de fonction.} \end{array}$$

Exemple de terme :

$$(\lambda f.\, f\, f)\, (\lambda x.\, x)$$

Les étapes de son calcul :

$$(\lambda f.\, f\, f)\, (\lambda x.\, x) \to (\lambda x.\, x)(\lambda x.\, x) \to \lambda x.\, x$$

Coder les booléens et les entiers

Les valeurs de vérité $\top$ et $\bot$ sont des fonctions qui sélectionnent un de leurs deux arguments :

$$\top = \lambda x.\, \lambda y.\, x$$

$$\bot = \lambda x.\, \lambda y.\, y$$

$$\texttt{if } M \texttt{ then } M_1 \texttt{ else } M_2 = M\, M_1\, M_2$$

L'entier n est la composition de fonction itérée $f \mapsto \overbrace{f \circ f \circ \cdots \circ f}^{n \text{ fois}}$.

$$0 = \lambda f.\, \lambda x.\, x \qquad 1 = \lambda f.\, f \qquad 2 = \lambda f.\, \lambda x.\, f\ (f\ x)$$

$$M + N = \lambda f.\, \lambda x.\, M\, f\ (N\, f\ x)$$

$$M \times N = \lambda f.\, \lambda x.\, M\ (N\, f)\ x$$

$$M^N = N\, M$$

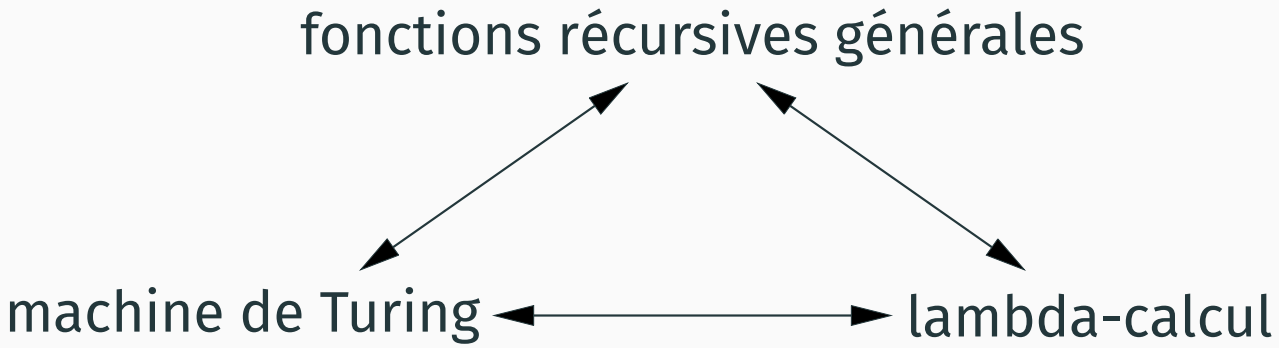
Calculer des fonctions entières avec des lambda-termes

Avec ces codages plus celui d'un ≪combinateur de points fixes≫, on obtient un petit langage de programmation fonctionnelle dans lequel on peut définir p.ex.

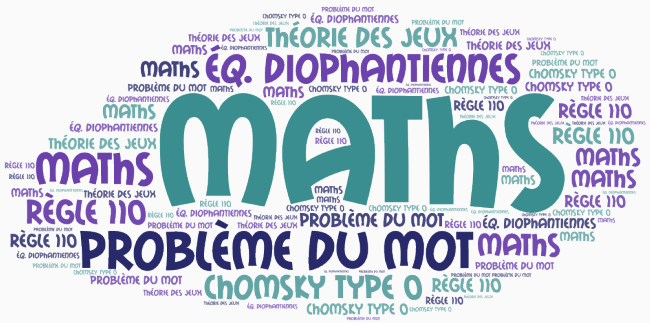
$$\texttt{fact} = \lambda n.\ \texttt{if}\ n = 0\ \texttt{then}\ 1\ \texttt{else}\ n \times \texttt{fact}\ (n - 1)$$

et toutes les fonctions récursives générales sur les entiers.

Une notion universelle de calculabilité ?

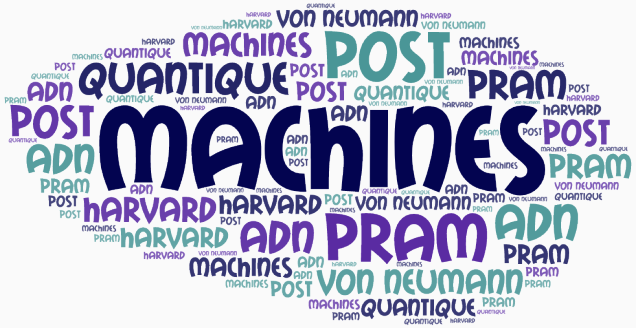


Une notion universelle de calculabilité ?

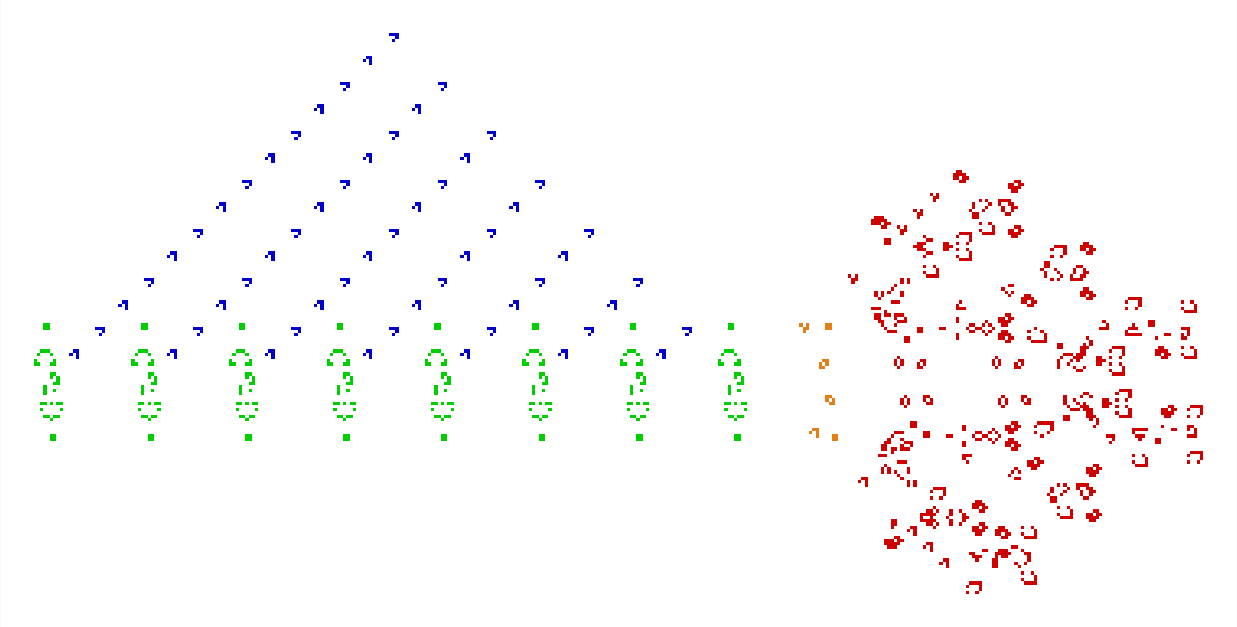


fonctions récursives générales

machine de Turing ⟷ lambda-calcul



Une notion universelle de calculabilité ?



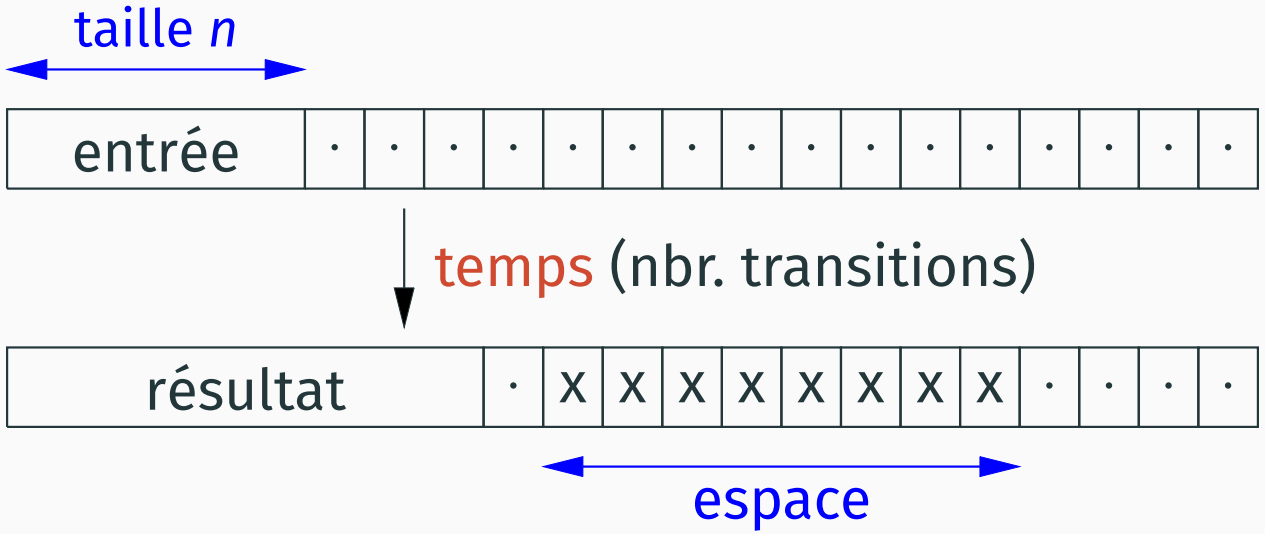
Du jeu de la vie de Conway au calculateur quantique, beaucoup de modèles de calculs sont ≪Turing-complets≫
(ont la puissance d'expression d'une machine de Turing).

Mais on ne connait aucun modèle ≪naturel≫ de calcul
qui calculerait plus de fonctions que les machines de Turing.

De la calculabilité à la complexité

La complexité d'un calcul

Étant donné une machine de Turing et une entrée de taille n :



La complexité en temps du calcul est l'ordre de grandeur du nombre de transitions effectuées par la machine.

La complexité en espace du calcul est l'ordre de grandeur du nombre de cases mémoire supplémentaires utilisées.

Ex : incrémenter un nombre de n bits : temps linéaire en n, espace constant.

Quelques classes de complexité

P Problèmes résolubles en temps polynomial.

EXPTIME Problèmes résolubles en temps exponentiel.

PSPACE Problèmes résolubles en espace polynomial.

NP Problèmes dont les solutions peuvent être vérifiées en temps polynomial.

Un problème NP : la satisfiabilité d'une formule booléenne

$$(m \vee \neg c \vee k) \wedge (w \vee v \vee \neg e) \wedge (\neg s \vee \neg x \vee \neg e) \wedge (\neg r \vee \neg w \vee \neg q) \wedge (x \vee \neg q \vee \neg m) \wedge (e \vee \neg v \vee b) \wedge (s \vee z \vee \neg r) \wedge (t \vee j \vee \neg m) \wedge (\neg x \vee u \vee \neg j) \wedge (\neg t \vee a \vee y) \wedge (c \vee r \vee n) \wedge (k \vee q \vee k) \wedge (c \vee w \vee \neg b) \wedge (q \vee \neg m \vee \neg x) \wedge (\neg c \vee \neg a \vee \neg y)$$

Vérifier qu'une affectation variable $\mapsto$ valeur booléenne satisfait la formule :

facile ! (temps linéaire)

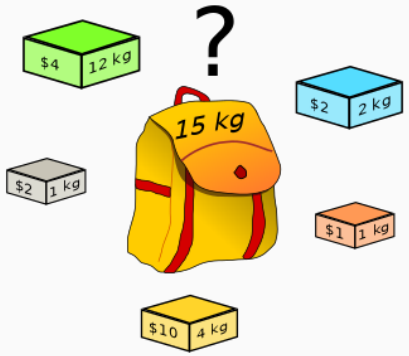
Trouver une affectation variable $\mapsto$ valeur booléenne qui satisfait la formule :

- par énumération naïve : exponentiel
- meilleurs algorithmes connus : exponentiels aussi.

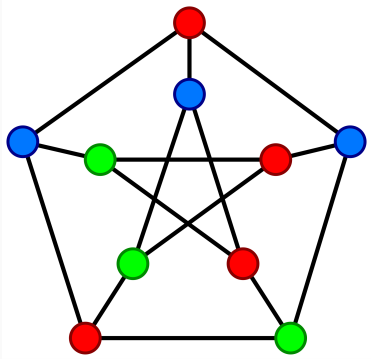
Des problèmes NP-complets

$$(m \vee \bar{c} \vee k) \wedge (w \vee v \vee \bar{e}) \wedge (\bar{s} \vee \bar{x} \vee \bar{e}) \wedge (\bar{r} \vee \bar{w} \vee \bar{q}) \wedge (x \vee \bar{q} \vee \bar{m}) \wedge (e \vee \bar{v} \vee b) \wedge (s \vee z \vee \bar{r}) \wedge (t \vee j \vee \bar{m}) \wedge (\bar{x} \vee u \vee \bar{j}) \wedge (\bar{t} \vee a \vee y) \wedge (c \vee r \vee n) \wedge (k \vee q \vee k) \wedge (c \vee w \vee \bar{b}) \wedge (q \vee \bar{m} \vee \bar{x}) \wedge (\bar{c} \vee \bar{a} \vee \bar{y})$$

(Satisfiabilité)



(Optimisation linéaire)



(Coloriage de graphe)

Des centaines de problèmes utiles en pratique :

- ils sont dans NP ;
- tous les algorithmes connus sont en temps non polynomial ;
- si un de ces problème avait un algorithme polynomial, tous les problèmes NP en auraient un aussi.

Un problème ouvert : P = NP ?

(Le plus grand problème ouvert en informatique théorique ;
un des 7 problèmes du millénaire de l'institut Clay.)

Si P $\neq$ NP :

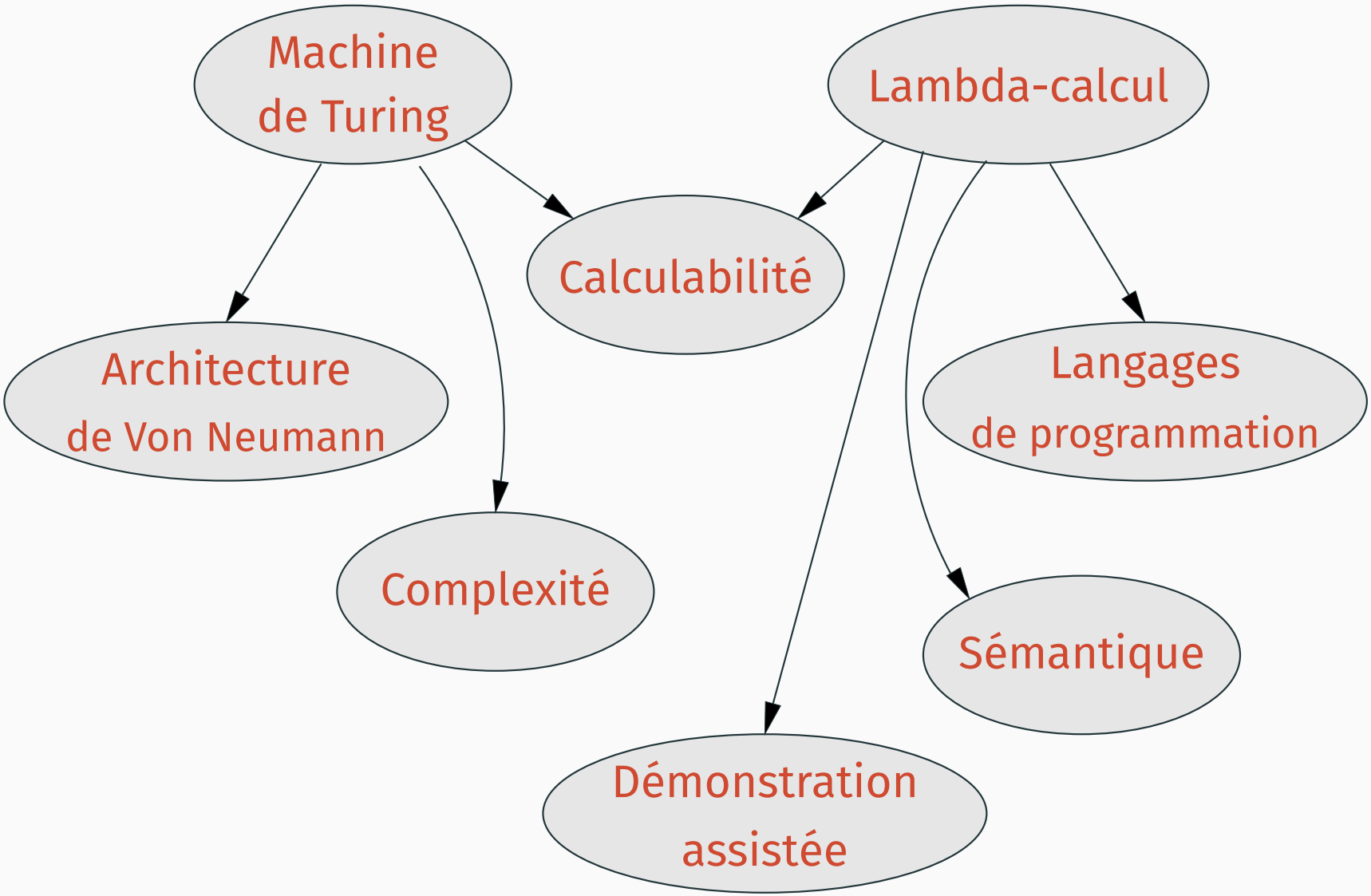
Il existe un problème facile à vérifier (dans NP)
mais qui n'a aucun algorithme en temps polynomial
(pas dans P).

Si P $=$ NP :

Tous les problèmes faciles à vérifier (dans NP)
sont faciles à résoudre (en temps polynomial, dans P).

Conclusion

Un résultat négatif des plus féconds !



Et l'ordinateur quantique dans tout ça ?

Dans le même état que l'ordinateur classique en 1940 :

- théorie déjà bien comprises (modèles du calcul quantique, algorithmes quantiques utiles, etc.);
- réalisation particulièrement difficile.

Questions de complexité ouvertes : BPP = BQP ? BQP = NP ?

(On pense qu'il y a un ≪avantage quantique≫, donc BQP $\not\subseteq$ BPP, mais qu'il ne suffit pas à résoudre tous les problèmes NP.)

Et l'intelligence artificielle dans tout ça ?

L'IA du XX^{e} siècle :

- Approches déductives et combinatoires.
- Succès limités.

Et l'intelligence artificielle dans tout ça ?

L'IA du XX^{e} siècle :

- Approches déductives et combinatoires.
- Succès limités.

L'IA du début du XXI^{e} siècle :

- Approches par apprentissage statistique.
- Progrès foudroyants sur les tâches de perception et de génération (d'images, de textes, …)
- Capacités de raisonnement quasi nulles.

Et l'intelligence artificielle dans tout ça ?

L'IA du XX^{e} siècle :

- Approches déductives et combinatoires.
- Succès limités.

L'IA du début du XXI^{e} siècle :

- Approches par apprentissage statistique.
- Progrès foudroyants sur les tâches de perception et de génération (d'images, de textes, …)
- Capacités de raisonnement quasi nulles.

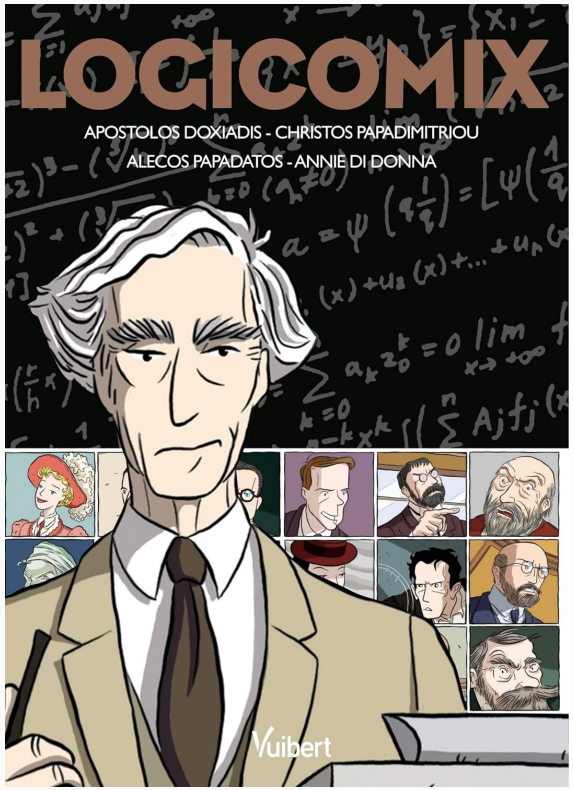
Une approche mixte ?

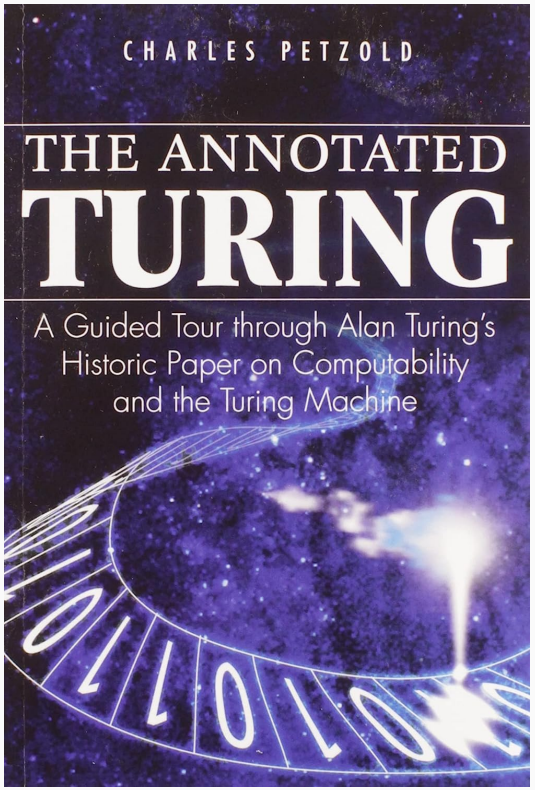
- L'IA générative produit une solution (partielle) qui serait ensuite vérifiée / affinée par des méthodes classiques.

De la logique avant toute chose…

… et des statistiques juste après ?

Pour en savoir plus





A. Doxiadis, C. Papadimitriou, A. Papadatos, A. Di Donna : *Logicomix*, Vuibert, 2022.

C. Petzold, *The Annotated Turing : A Guided Tour Through Alan Turing's Historic Paper on Computability and the Turing Machine*, Wiley, 2008.