La machine de Turing, entre logique et informatique

Emmanuel Beffara

Équipe Logique de la Programmation
Institut de mathématiques de Marseille
Université d'Aix-Marseille
+
Équipe Plume
Laboratoire de l'informatique du parallélisme
École normale supérieure de Lyon

19 mars 2015
IREM de Brest

Plan

Une page d'histoire

La crise des fondements et le calcul

À l'aube du XXème siècle

Hilbert est le parrain des mathématiques au début du XXème siècle, c'est l'époque de la *crise des fondements*.
Parmi ses 23 grands problèmes, on trouve :

- Le 1er : l'hypothèse du continu.
- Le 2ème : la cohérence de l'arithmétique.
- Le 10ème : trouver une méthode algorithmique pour résoudre les équations diophantiennes.

David Hilbert, 1912

En 1928 il pose le *problème de la décision* ou *Entscheidungsproblem* :

- Existe-t-il un algorithme permettant de déterminer si un énoncé est démontrable, dans un système formel donné ?

Pour fonder les mathématiques, on cherche des méthodes *finitistes*.

Formalisation de la notion de calcul

Leibniz est le premier à avancer l'idée d'un langage formalisé et d'une méthode mécanique permettant de calculer la véracité de tout énoncé.

Gottfried Wilhelm Leibniz, ~1700

Plusieurs définitions formelles de la notion de calcul ont été proposées dans les années 1930 suite au programme de Hilbert, notamment :

- les fonctions récursives (Herbrand, Gödel puis Kleene)
- le λ-calcul (Church)
- les machines de Turing

Jacques Herbrand, 1931

Stephen Cole Kleene, 1978

Alonzo Church, ~1950

Les machines de Turing

Formaliser le calcul

Apprenons à calculer

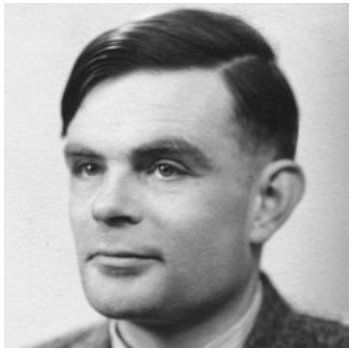
- Posons une addition.

Apprenons à calculer

- Posons une addition.
- Posons une multiplication.

Apprenons à calculer

- Posons une addition.
- Posons une multiplication.



Alan Turing, 1951

On travaille en écrivant des symboles, en se déplaçant sur un espace de travail, en se souvenant de quelques informations (comme l'étape où on en est dans le calcul, les retenues, etc.).

Une définition formelle

Définition

Une machine de Turing est définie par

- un ensemble fini Σ (l'alphabet)
  - un symbole particulier $\square$: case vide
- un ensemble fini Q (les états)
  - un état initial $q_0 \in Q$
  - un état final $q_f \in Q$
- une table de transition $\delta : Q \times \Sigma \to Q \times \Sigma \times \{-1, 0, 1\}$

$\delta(q, a) = (q', b, d)$ signifie

Si je suis dans l'état q et si je vois la lettre a, alors je passe dans l'état q', j'écris la lettre b et je me déplace dans la direction d.

Une définition formelle

Définition

Une *configuration* d'une telle machine est définie par

- un état $q \in Q$,
- un entier $p \in \mathbb{Z}$ (la position),
- une fonction $r : \mathbb{Z} \to \Sigma$ (l'état du ruban) avec $f(x) = \square$ pour presque tout x

- La configuration *initiale* utilise l'état q_0, la position 0, et porte sur le ruban un certaine *entrée* m (un suite de symboles de l'alphabet).
- La table de transition dit comment on passe d'une configuration à une autre.
- Le résultat du calcul est le contenu du ruban quand on atteint l'état final q_f (si jamais on l'atteint).

Exemple : incrémenter un entier

Écrivons une machine qui ajouter 1 à un nombre entier écrit en base 10, avec les unités à gauche.

- l'alphabet : $\Sigma = \{\square, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- les états : $Q = \{q_0, q_f\}$
- les transitions :

$$
\begin{array}{cccccc}
q_0 & 0 & \rightarrow & q_f & 1 & 0 \\
q_0 & 1 & \rightarrow & q_f & 2 & 0 \\
 & & & \dots & & \\
q_0 & 8 & \rightarrow & q_f & 9 & 0 \\
q_0 & 9 & \rightarrow & q_0 & 0 & +1 \\
q_0 & \square & \rightarrow & q_f & 1 & 0
\end{array}
$$

Faisons tourner la machine au tableau.

Exemple : incrémenter un entier, à l'endroit

La même chose, mais avec le chiffre des unités à droite !

- l'alphabet : $\Sigma = \{\$, \square, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- les états : $Q = \{q_0, I, q_f\}$
- les transitions :

q_0	i	$\rightarrow$	q_0	i	$+1$	pour $i \neq \square$
q_0	$\square$	$\rightarrow$	I	$\square$	-1	
I	i	$\rightarrow$	q_f	$i+1$	0	pour $i \in \{0, 1, ..., 8\}$
I	9	$\rightarrow$	I	0	-1	
I	$\square$	$\rightarrow$	q_f	1	0	

q_0 signifie « je vais chercher le dernier chiffre à droite »
I signifie « j'incrémente de droite à gauche »

Exercices

- Écrire une machine qui calcule l'addition :
  si en entrée il y a le texte « 1862 + 9645 », le calcul termine avec « 11507 » sur le ruban
- Écrire une machine qui calcule la multiplication :
  si en entrée il y a le texte « 1862×9645 », le calcul termine avec « 17958990 » sur le ruban
- Écrire la division euclidienne, l'algorithme d'Euclide, etc.

Faisons le point

Quelques remarques :

- Une machine de Turing manipule des symboles sans leur donner de sens, elle ne calcule des choses qu'au travers de *codages*.
- La machine elle-même est un objet *fini*, les seules sources d'infini sont l'espace du ruban et le temps de calcul.

Cette définition arbitraire est-elle pertinente ?

Thèse de Church-Turing

Les fonctions calculables par quelque moyen que ce soit (mécanique ou mental) sont exactement celles que l'on peut représenter par une machine de Turing.

Ce n'est pas assez formel pour être une théorème, néanmoins :

- le fait que ce qui est calculable par une machine de Turing soit effectivement calculable est assez consensuel,
- tous les modèles de calcul connus (notions de machine, langages de programmation, etc) peuvent être représentés avec des machines de Turing.

Et les ordinateurs dans tout ça ?

Fait

Les ordinateurs réels ne sont pas conçus comme avec les machines de Turing comme modèle.

Turing n'a pas « inventé l'ordinateur » (et n'a pas cherché à le faire) en définissant sa notion de machine.

- Les premiers ordinateurs on plutôt été une réalisation de la machine de von Neumann (notion de processeur et de mémoire contenant les données et les programmes).
- Un ordinateur ne fait pas que manipuler des symboles dans une mémoire interne : il interagit avec le monde réel.

Même dans l'activité purement calculatoire, un ordinateur réel se comporte de façon plus compliquée qu'une machine de Turing (parce que c'est plus efficace), mais c'est équivalent.

Les problèmes de décision

Calculer pour démontrer

Problèmes de décision

Ce sont les fonctions dont la réponse est « oui » ou « non ».

- Un mot donné est-il un palindrome ?
  - entrée : une suite de lettres $a_1 a_2 a_3 ... a_n$
  - sortie : 1 si pour tout i, $a_i = a_{n+1-i}$, 0 sinon
- Un nombre entier est-il premier ?
  - entrée : un nombre entier n écrit en base 10 avec les unités à droite (par exemple)
  - sortie : 1 si n est premier, 0 sinon
- Telle équation diophantienne a-t-elle une solution ?
  - entrée : un suite de symboles comme « $19 \times a\hat{}17 - 85 \times a\hat{}3 \times b\hat{}5 + 58 \times a\hat{}2 - 607 \times b + 8$ »
  - sortie : 1 s'il existe $a, b \in \mathbb{Z}$ qui annulent le polynôme, 0 sinon

  C'est le dixième problème de Hilbert.

On se demande s'il existe une machine qui calcule ladite fonction.

Décidabilité algorithmique

Définition

Un problème est (algorithmiquement) *décidable* s'il existe une machine qui calcule la fonction associée, il est *indécidable* sinon.

Pour établir qu'un problème est décidable :

- on écrit un algorithme qui calcule le résultat,
- on le programme sous forme d'une machine de Turing,
- on démontre la correction de l'algorithme et du programme !

Pour établir qu'un problème n'est pas décidable, comment fait-on ?

- soit on trouve un moyen de l'établir directement,
- soit on y ramène un problème déjà connu comme indécidable.

Intermède : les machines universelles

Regardons-nous calculer avec les machines de Turing.

- Quand on fait tourner une machine à la main, on fait un calcul.
- Si la table de transition est écrite entièrement, l'appliquer pour dérouler le calcul est une opération très simple.
- Si simple qu'une machine de Turing peut l'effectuer !

Intermède : les machines universelles

Regardons-nous calculer avec les machines de Turing.

- Quand on fait tourner une machine à la main, on fait un calcul.
- Si la table de transition est écrite entièrement, l'appliquer pour dérouler le calcul est une opération très simple.
- Si simple qu'une machine de Turing peut l'effectuer !

Définition

Une machine de Turing est *universelle* si, étant donnés

- une description d'une machine M,
- une entrée e,

elle calcule le résultat que M obtiendrait en calculant sur l'entrée e.

Le problème de l'arrêt

On considère le problème de décision suivant :

- entrée : une description d'une machine de Turing M et une entrée e pour cette machine
- sortie : 1 si le calcul de M sur e s'arrête avec un résultat, 0 si ce ce calcul ne s'arrête jamais

Ce problème est-il décidable ?

Le problème de l'arrêt

On considère le problème de décision suivant :

- entrée : une description d'une machine de Turing M et une entrée e pour cette machine
- sortie : 1 si le calcul de M sur e s'arrête avec un résultat, 0 si ce ce calcul ne s'arrête jamais

Ce problème est-il décidable ?

Il est facile d'écrire une machine qui répond 1 si M s'arrête sur l'entrée e et qui ne répond jamais si M ne s'arrête pas.
Le problème est au moins *semi-décidable*.

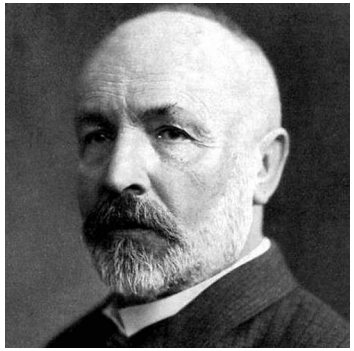
Intermède : le diabolique argument diagonal

Théorème (Cantor)

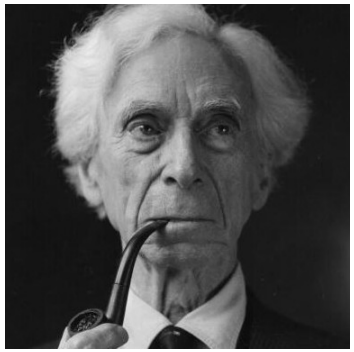
Aucun ensemble ne peut être en bijection avec l'ensemble de ses parties.

- Soit E un ensemble, supposons qu'il existe une bijection $f : E \to \mathscr{P}(E)$.
- Posons l'ensemble $A = \{x \in E \mid x \notin f(x)\}$.
- Par hypothèse, il existe $a \in E$ tel que $f(a) = A$.
- On déduit que $a \in A$ si et seulement si $a \notin f(a)$, c'est contradictoire !

On peut voir cet argument comme un exploitation du paradoxe de Russel dans le cadre de la théorie des ensembles.



Georg Cantor



Bertrand Russell

Indécidabilité du problème de l'arrêt

Théorème

Le problème de l'arrêt est indécidable.

Indécidabilité du problème de l'arrêt

Théorème

Le problème de l'arrêt est indécidable.

Supposons qu'on ait une machine H qui résout le problème de l'arrêt.

Indécidabilité du problème de l'arrêt

Théorème

Le problème de l'arrêt est indécidable.

Supposons qu'on ait une machine H qui résout le problème de l'arrêt.

Définissons le programme suivant :

- entrée : un mot m décrivant une machine de Turing
- programme :
  appeler la machine H avec la description m et l'entrée m ;
  si H a répondu 1, tourner en rond sans jamais répondre,
  si H a répondu 0, terminer le calcul (en répondant 1).

Indécidabilité du problème de l'arrêt

Théorème

Le problème de l'arrêt est indécidable.

Supposons qu'on ait une machine H qui résout le problème de l'arrêt.

Définissons le programme suivant :

- entrée : un mot m décrivant une machine de Turing
- programme :
  appeler la machine H avec la description m et l'entrée m ;
  si H a répondu 1, tourner en rond sans jamais répondre,
  si H a répondu 0, terminer le calcul (en répondant 1).

Si on suppose que H existe, il n'est pas difficile d'écrire une machine P qui réalise ce programme. Soit p une description de cette machine.

Indécidabilité du problème de l'arrêt

Théorème

Le problème de l'arrêt est indécidable.

Supposons qu'on ait une machine H qui résout le problème de l'arrêt.

Définissons le programme suivant :

- entrée : un mot m décrivant une machine de Turing
- programme :
  appeler la machine H avec la description m et l'entrée m ;
  si H a répondu 1, tourner en rond sans jamais répondre,
  si H a répondu 0, terminer le calcul (en répondant 1).

Si on suppose que H existe, il n'est pas difficile d'écrire une machine P qui réalise ce programme. Soit p une description de cette machine.

Que répondra P sur l'entrée p ?

Problèmes de décision en logique

Que peut-on dire du problème suivant ?

- entrée : une formule logique
- sortie : 1 si cette formule est démontrable, 0 sinon

Tout dépend de ce que l'on s'autorise à écrire comme formules.

- Calcul propositionnel : décidable
- Arithmétique : indécidable
- Arithmétique avec quantification bornée : décidable
- Arithmétique sans multiplication : décidable
- Théorie ensembles : indécidable
- Théorie des corps réels clos : décidable

La question de la cohérence

Entre logique et informatique fondamentale

La cohérence

En logique mathématique comme en informatique fondamentale, une question centrale est celle de la cohérence :

- Un système logique est cohérent s'il n'est pas dégénéré, c'est-à-dire qu'il ne démontre pas n'importe quoi.
- Un programme est cohérent s'il se comporte bien (pas d'erreur, pas de boucle infinie, pas de blocage).

Dans un cas comme dans l'autre, une question sous-jacente est celle de la *signification* (des démonstrations et des programmes).

Décidabilité logique

Définition

Un énoncé est *décidable* dans un système formel donné s'il peut être démontré ou réfuté dans ce système, il est *indécidable* sinon.

Des exemples classiques d'énoncés indécidables :

- Le postulat des parallèles en géométrie euclidienne
- L'hypothèse du continu en théorie des ensembles

Définition

Un système formel est décidable s'il existe un algorithme permettant de savoir si un énoncé donné est décidable ou pas.

Remarque : un système indécidable contient nécessairement des énoncés indécidables, l'inverse n'est pas vrai !

Décidabilité, entre logique et informatique

La décidabilité logique se ramène à la décidabilité algorithmique :

- Les démonstrations (formelles) sont des objets finis que l'on peut énumérer avec une machine de Turing. On peut donc énumérer tous les théorèmes d'un système formel donné.
- Un énoncé est décidable si et seulement si cette énumération finit par trouver l'énoncé ou sa négation.
- Un système formel est décidable si et seulement si le problème de la terminaison de cette recherche est décidable.

Décidabilité, entre logique et informatique

La décidabilité logique se ramène à la décidabilité algorithmique :

- Les démonstrations (formelles) sont des objets finis que l'on peut énumérer avec une machine de Turing. On peut donc énumérer tous les théorèmes d'un système formel donné.
- Un énoncé est décidable si et seulement si cette énumération finit par trouver l'énoncé ou sa négation.
- Un système formel est décidable si et seulement si le problème de la terminaison de cette recherche est décidable.

Théorème

L'arithmétique est indécidable (ou incohérente).

On peut représenter en arithmétique le calcul d'une machine de Turing !

Les théorèmes d'incomplétude

Théorème (Gödel)

Dans tout système formel cohérent contenant l'arithmétique élémentaire, il y a un énoncé indécidable.

On fixe une énumération de tous les prédicats.
L'énoncé $G(n)$ = « le n-ième prédicat appliqué à n n'est pas démontrable » s'écrit en arithmétique.
Soit g le numéro du prédicat G. Si l'arithmétique est cohérente, alors l'énoncé $G(g)$ est indécidable.

Kurt Gödel, 1925

Théorème (Gödel)

Un système formel qui permet de démontrer sa propre cohérence est nécessairement incohérent.

On représente la démonstration précédente en arithmétique !

Et si l'arithmétique était incohérente ?

Théorème (Gentzen)

L'arithmétique est cohérente.

- On pose une définition adaptée de la notion de démonstration formelle : le *calcul des séquents*.
- On définit un algorithme permettant de transformer une démonstration quelconque en démonstration directe : l'*élimination des coupures*.
- On prouve qu'aucune démonstration directe ne peut établir de contradiction.

Gerhard Gentzen, 1945

Et si l'arithmétique était incohérente ?

Théorème (Gentzen)

L'arithmétique est cohérente.

- On pose une définition adaptée de la notion de démonstration formelle : le *calcul des séquents*.
- On définit un algorithme permettant de transformer une démonstration quelconque en démonstration directe : l'*élimination des coupures*.
- On prouve qu'aucune démonstration directe ne peut établir de contradiction.

Gerhard Gentzen, 1945

Mais on ne peut pas prouver en arithmétique que l'algorithme termine !

La correspondance preuves-programmes

Correspondance de Curry-Howard

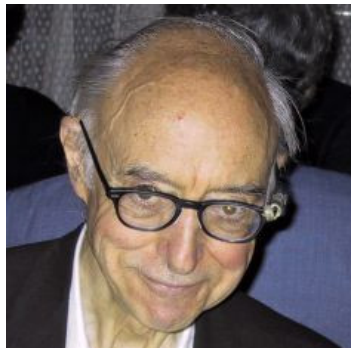
Les démonstrations formelles et les programmes sont des objets de même nature.

- La normalisation des preuves correspond à l'évaluation des programmes.
- La cohérence logique est équivalente à la question de la terminaison.

C'est le cœur de la théorie de la démonstration moderne et de l'informatique fondamentale, avec de nombreuses applications :

- démonstration assistée par ordinateur ...
- extraction et certification de programmes ...

Haskell Brooks Curry



William Alvin Howard

Questions pour l'apéritif

La thèse de Church-Turing est-elle inébranlable ?

- Peut-il exister des processus de calcul qui dépassent la capacité des machines de Turing (notamment grâce à la mécanique quantique) ?
- Que dire de la calculabilité *réelle* dans un univers borné ?

Le cerveau est-il une machine de Turing ?

- Si oui, il y a des notions intrinsèquement inaccessibles, que dire alors de la nature des objets mathématiques ?
- Si non, quel est cet *organe de l'intuition* qui nous différencierait des machines ?
- La logique *classique* rend-elle bien compte du raisonnement réel, y a-t-il une vérité mathématique indépendante de notre réflexion ?