Didactique de l'informatique

M1 MEEF NSI

Sur la calculabilité

Emmanuel Beffara

Institut national supérieur du professorat et de l'éducation
Académie de Grenoble

2022 – 2023

Au programme

Notion de programme en tant que donnée. Calculabilité, décidabilité.	Comprendre que tout programme est aussi une donnée. Comprendre que la calculabilité ne dépend pas du langage de programmation utilisé. Montrer, sans formalisme théorique, que le problème de l'arrêt est indécidable.	L'utilisation d'un interpréteur ou d'un compilateur, le téléchargement de logiciel, le fonctionnement des systèmes d'exploitation permettent de comprendre un programme comme donnée d'un autre programme.

Plan

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 $a_1 a_2 a_3 \dots a_n$
  - 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 polynomiale 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 un programme qui calcule ladite fonction.

Décidabilité algorithmique

Définition

Un problème est (algorithmiquement) *décidable* s'il existe un programme 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 dans le langage de son choix
- 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.

Formalisation de la notion de calcul

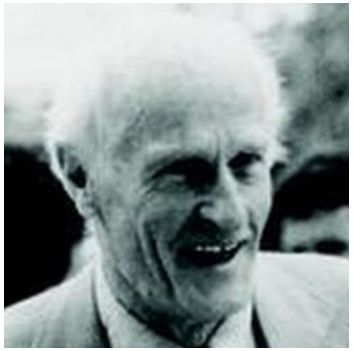
Leibniz est le premier à avancer l'idée d'un langage formalisé et d'une méthode mécanique permettant de calculer la véracité de tout énoncé.

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

Apprenons à calculer

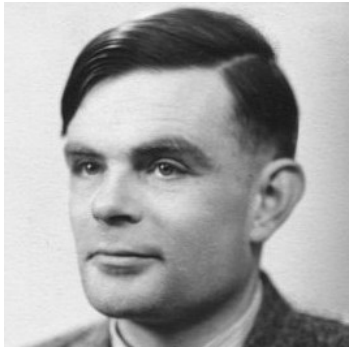
- Posons une addition.

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

Pour quoi faire ?

Définition

Une machine de Turing est définie par

- un ensemble fini Σ (l'alphabet)
  - un symbole particulier $\emptyset$: 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 $f(x) = \emptyset$ 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 : $\Sigma = \{\emptyset, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- les états : $Q = \{q_0, q_f\}$
- les transitions :

$$
\begin{array}{lcllc}
q_0 & 0 & \rightarrow & q_f & 1 \quad 0 \\
q_0 & 1 & \rightarrow & q_f & 2 \quad 0 \\
 & & \ldots & & \\
q_0 & 8 & \rightarrow & q_f & 9 \quad 0 \\
q_0 & 9 & \rightarrow & q_0 & 0 \quad +1 \\
q_0 & \emptyset & \rightarrow & q_f & 1 \quad 0
\end{array}
$$

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

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

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 !

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*.

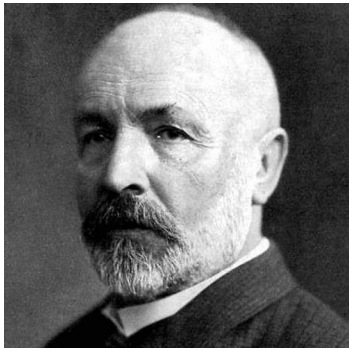
Intermède : le diabolique argument diagonal

Théorème (Cantor)

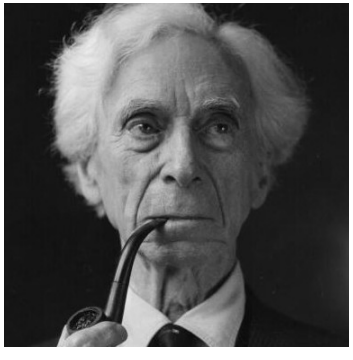
Aucun ensemble ne peut être en bijection avec l'ensemble de ses parties.

- Soit E un ensemble, supposons qu'il existe une bijection $f : E \to \mathcal{P}(E)$.
- 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 ?

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.

Plan

Défi

Écrire ensemble une explication du théorème de l'arrêt avec le vocabulaire d'une classe de terminale NSI.

Transposition en classe

- Quels thèmes fondamentaux ?
- Quoi institutionnaliser ?
- Quelle intégration avec le reste du programme ?
- Quels objectifs ?

Le code comme donnée

Une idée fondamentale à l'origine des grands résultats d'indécidabilité.

- Écrire un programme qui produit du code.
  - dérouleur de boucles en Python
  - produire de l'assembleur (contexte d'architecture)
- Interpréter du code
  - les L-systèmes
  - simuler une machine de Turing (difficile)
- Les machines virtuelles

Lien avec le débogage

Le débogage consiste à étudier un programme pour déterminer s'il fait bien ce qu'on attend de lui. C'est la version concrète de ce qu'on étudie en calculabilité :

- Déterminer s'il y a des bugs est indécidable.
  - terminaison
  - accessibilité, détection de code mort
  - respect d'une spécification
- Possibilité de traiter des cas particuliers, de faire des approximations.
- Écrire des tests permet d'identifier des bugs mais pas de prouver leur absence.
- Déterminer une propriété peut être arbitrairement compliqué.

Universalité

Les résultats de la calculabilité ne dépendent pas du langage choisi :

- chaque langage permet de simuler les autres (thèse de Church-Turing)
- on fait la théorie avec le modèle Turing parce que c'est commode

Situations susceptibles d'évoquer ces points :

- architecture : un jeu d'instructions réduit pour tout implémenter
- automates cellulaires, jeu de la vie de Conway
- les limites des ordinateurs (modèle théorique vs machines physiques)