Décidabilité

Yann Strozecki
yann.strozecki@uvsq.fr

6 mai 2021

Calculabilité

La calculabilité est le domaine de l'informatique qui s'intéresse au pouvoir de décision des différents modèles de calcul.

Conjecture (Thèse de Church)

Toute fonction calculable par un dispositif physique est calculable par une machine de Turing.

Calculabilité

La calculabilité est le domaine de l'informatique qui s'intéresse au pouvoir de décision des différents modèles de calcul.

Conjecture (Thèse de Church)

Toute fonction calculable par un dispositif physique est calculable par une machine de Turing.

Cette conjecture est renforcée par l'expérience, les modèles suivants ont la même expressivité (Turing puissant) :

- toutes les variantes des machines de Turing
- le lambda calcul
- les langage de programmation
- les machine de Von Neumann
- les ordinateurs quantiques

Calculabilité

La calculabilité est le domaine de l'informatique qui s'intéresse au pouvoir de décision des différents modèles de calcul.

Conjecture (Thèse de Church)

Toute fonction calculable par un dispositif physique est calculable par une machine de Turing.

Cette conjecture est renforcée par l'expérience, les modèles suivants ont la même expressivité (Turing puissant) :

- toutes les variantes des machines de Turing
- le lambda calcul
- les langage de programmation
- les machine de Von Neumann
- les ordinateurs quantiques

Questions fondamentales

On utilise les termes *langage* et *problème* de manière interchangeable. Le problème associé au langage est de décider si un mot appartient au langage.

Definition

Un problème est dit décidable (et une fonction calculable) s'il existe une machine de Turing qui le décide.

1. Étant donné un problème, est-il décidable ?
2. Existe-t-il un problème qui n'est pas décidable ?
3. Peut-on décider efficacement d'un problème ?

Questions fondamentales

On utilise les termes *langage* et *problème* de manière interchangeable. Le problème associé au langage est de décider si un mot appartient au langage.

Definition

Un problème est dit décidable (et une fonction calculable) s'il existe une machine de Turing qui le décide.

1. Étant donné un problème, est-il décidable ?
2. Existe-t-il un problème qui n'est pas décidable ?
3. Peut-on décider efficacement d'un problème ?

Références

Quelques références.

- Introduction to the Theory of Computation, M Sipser
- Introduction à la calculabilité - 3ème édition, P. Wolper
- Complexité algorithmique. S. Perifel.

Compilation et pseudo-code

Il existe un langage de programmation minimal qui permet de décrire les fonctions calculables (fonctions récursives) :

1. la fonction constante égale à 0
2. la fonction successeur
3. les compositions de fonctions
4. l'équivalent du while

Le passage d'un langage qui permet de décrire comment on calcule une fonction, voir qui décrit seulement la fonction, à une machine qui réalise la calcul s'appelle la compilation.

Compilation et pseudo-code

Il existe un langage de programmation minimal qui permet de décrire les fonctions calculables (fonctions récursives) :

1. la fonction constante égale à 0
2. la fonction successeur
3. les compositions de fonctions
4. l'équivalent du while

Le passage d'un langage qui permet de décrire comment on calcule une fonction, voir qui décrit seulement la fonction, à une machine qui réalise la calcul s'appelle la compilation.

On peut compiler du pseudo-code ou du C vers une machine de Turing.

Compilation et pseudo-code

Il existe un langage de programmation minimal qui permet de décrire les fonctions calculables (fonctions récursives) :

1. la fonction constante égale à 0
2. la fonction successeur
3. les compositions de fonctions
4. l'équivalent du while

Le passage d'un langage qui permet de décrire comment on calcule une fonction, voir qui décrit seulement la fonction, à une machine qui réalise la calcul s'appelle la compilation.

On peut compiler du pseudo-code ou du C vers une machine de Turing.

Existence de problèmes indéciables

Théorème

Il existe un codage des machines de Turing par des entiers (ou des suites de 0 *et de* 1*).*

En proposer un, on notera $< M >$ le code de M.

Théorème

Il existe une fonction qui n'est pas calculable par machine de Turing.

Existence de problèmes indéciables

Théorème

Il existe un codage des machines de Turing par des entiers (ou des suites de 0 et de 1).

En proposer un, on notera $< M >$ le code de M.

Théorème

Il existe une fonction qui n'est pas calculable par machine de Turing.

Pour comprendre cela, il faut savoir que les entiers sont moins nombreux que les fonctions des entiers vers les entiers.

$\aleph_0 \neq 2^{\aleph_0}$ ou encore $\mathbb{N} \neq \mathbb{R}$

Existence de problèmes indéciables

Théorème

Il existe un codage des machines de Turing par des entiers (ou des suites de 0 et de 1).

En proposer un, on notera $< M >$ le code de M.

Théorème

Il existe une fonction qui n'est pas calculable par machine de Turing.

Pour comprendre cela, il faut savoir que les entiers sont moins nombreux que les fonctions des entiers vers les entiers.
$\aleph_0 \neq 2^{\aleph_0}$ ou encore $\mathbb{N} \neq \mathbb{R}$

Diagonalisation et Kantor

Pour comprendre l'argument du transparent précédent, on doit connaître le procédé de diagonalisation de Kantor.

On veut montrer qu'il y a plus de réels que d'entiers (ou de fonctions des entiers vers les entiers), autrement dit qu'on ne peut pas numéroter les réels.

On obtient alors facilement une contradiction, au tableau.

Machine universelle

Théorème

Il existe une machine de Turing U à 3 rubans qui simule n'importe quelle machine M à un ruban sur l'alphabet $\{0, 1, \square\}$. C'est à dire que pour toute entrée x de la machine M, le calcul de U sur $\langle M \rangle, x$ est le même que le calcul de M sur x.

Cela correspond à la notion d'interpréteur en informatique.
Cela correspond également au fonctionnement d'un système d'exploitation ou d'une machine virtuelle.

Machine universelle

Théorème

Il existe une machine de Turing U *à* 3 *rubans qui simule n'importe quelle machine* M *à un ruban sur l'alphabet* $\{0, 1, \Box\}$*. C'est à dire que pour toute entrée* x *de la machine* M*, le calcul de* U *sur* $\langle M \rangle, x$ *est le même que le calcul de* M *sur* x*.*

Cela correspond à la notion d'interpréteur en informatique.
Cela correspond également au fonctionnement d'un système d'exploitation ou d'une machine virtuelle.

Description de la machine universelle

L'alphabet de la machine U est $\{0, 1, \Box, \sharp\}$ et les états de M sont représentés par des entiers dénotés par $< q >$. Les trois rubans contiennent :

- $< M >$, le code de la machine à simuler sur le premier ruban
- le codage de l'état initial $< q0 >$ sur le deuxième ruban et le symbole sous la tête de lecture du troisième ruban
- x, le mot d'entrée sur le troisième ruban

Machine présentée par Turing avec 5 rubans (un pour la sortie, un pour l'entrée et un pour gérer plusieurs rubans de la machine simulée).

Fonctionnement de la machine universelle

À chaque étape de simulation, est écrit sur le deuxième ruban l'état courant $< q >$. Le symbole s est sous la tête de lecture dans le troisième ruban. Par recherche linéaire dans $< M >$, on trouve la transition $(< q > \sharp s \sharp < q' > \sharp s' \sharp D)$ qui permet :

- d'écrire le nouvel état $< q' >$ sur le deuxième ruban
- d'écrire le nouveau symbole s' sur la troisième bande
- de déplacer la tête de lecture du troisième ruban en suivant D

Simulation d'une machine M qui fonctionne en temps t en temps $O(t|M|)$ et $O(t^2|M|)$ si on doit gérer plusieurs rubans.

Problème de l'arrêt

On voudrait détecter automatiquement les bugs des programme.

Problème de l'arrêt :
Entrée : Une machine de Turing $< M >$ et une entrée x.
Question : Est-ce que M s'arrête sur x ?

Problème de l'arrêt

On voudrait détecter automatiquement les bugs des programme.

<u>**Problème de l'arrêt :**</u>
<u>**Entrée :**</u> Une machine de Turing $< M >$ et une entrée x.
<u>**Question :**</u> Est-ce que M s'arrête sur x ?

Indécidabilité du problème de l'arrêt

On dit qu'un problème est indécidable s'il n'existe aucune machine de Turing qui le décide.

Théorème

Le problème de l'arrêt est indécidable.

On ne peut donc pas trouver les bugs d'un programme ou se protéger contre des virus de manière automatique. Ceci explique qu'on fasse de la certification ou de la preuve de programme.

Indécidabilité du problème de l'arrêt

On dit qu'un problème est indécidable s'il n'existe aucune machine de Turing qui le décide.

Théorème

Le problème de l'arrêt est indécidable.

On ne peut donc pas trouver les bugs d'un programme ou se protéger contre des virus de manière automatique. Ceci explique qu'on fasse de la certification ou de la preuve de programme.

Voir la vidéo pour expliquer la preuve :
`https://www.youtube.com/watch?v=92WHN-pAFCs`

Indécidabilité du problème de l'arrêt

On dit qu'un problème est indécidable s'il n'existe aucune machine de Turing qui le décide.

Théorème

Le problème de l'arrêt est indécidable.

On ne peut donc pas trouver les bugs d'un programme ou se protéger contre des virus de manière automatique. Ceci explique qu'on fasse de la certification ou de la preuve de programme.

Voir la vidéo pour expliquer la preuve :
`https://www.youtube.com/watch?v=92WHN-pAFCs`

Éléments de preuve

On va exhiber un problème du même type qui est indécidable.

$$L = \{\langle M \rangle | M \text{ est une MT qui rejette l'entrée } \langle M \rangle\}$$

Simple raisonnement par l'absurde appliqué à une machine qui reconnait L. Paradoxe du barbier.

Éléments de preuve

On va exhiber un problème du même type qui est indécidable.

$$L = \{\langle M\rangle | M \text{ est une MT qui rejette l'entrée } \langle M\rangle\}$$

Simple raisonnement par l'absurde appliqué à une machine qui reconnait L. Paradoxe du barbier.

Quelques exemples de problèmes indécidables

1. Problème de correspondance de Post. On se donne un jeu de dominos avec des mots sur les deux parties de chaque domino. Peut-on trouver une suite de dominos qui compose le même mot en haut et en bas ?
2. Peut-on décider si un théorème d'arithmétique est prouvable dans l'axiomatique de Peano ?

Quelques exemples de problèmes indécidables

1. Problème de correspondance de Post. On se donne un jeu de dominos avec des mots sur les deux parties de chaque domino. Peut-on trouver une suite de dominos qui compose le même mot en haut et en bas ?
2. Peut-on décider si un théorème d'arithmétique est prouvable dans l'axiomatique de Peano ?
3. Peut-on trouver une solution entière à une équation polynomiale ? Dixième problème de Hilbert.

Quelques exemples de problèmes indécidables

1. Problème de correspondance de Post. On se donne un jeu de dominos avec des mots sur les deux parties de chaque domino. Peut-on trouver une suite de dominos qui compose le même mot en haut et en bas ?
2. Peut-on décider si un théorème d'arithmétique est prouvable dans l'axiomatique de Peano ?
3. Peut-on trouver une solution entière à une équation polynomiale ? Dixième problème de Hilbert.
4. Un ensemble de machines qui vérifient une propriété non triviale, théorème de Rice.

Quelques exemples de problèmes indécidables

1. Problème de correspondance de Post. On se donne un jeu de dominos avec des mots sur les deux parties de chaque domino. Peut-on trouver une suite de dominos qui compose le même mot en haut et en bas ?
2. Peut-on décider si un théorème d'arithmétique est prouvable dans l'axiomatique de Peano ?
3. Peut-on trouver une solution entière à une équation polynomiale ? Dixième problème de Hilbert.
4. Un ensemble de machines qui vérifient une propriété non triviale, théorème de Rice.

Réduction

Pour montrer que des problèmes sont indécidables on utilise des réductions.

Definition

On dit que L_1 se réduit à L_2 s'il existe une fonction f calculable par machine de Turing telle que $x \in L_1 \Leftrightarrow f(x) \in L_2$.

Même idée que la simulation entre modèles de calcul.
Pour montrer qu'on problème est indécidable, il suffit de lui réduire le problème de l'arrêt.

Exemple de réductions

- L_1 ensemble des machines de Turing qui acceptent toutes leurs entrées.
- L_2 ensemble des machine de Turing qui calculent 0 sur toutes les entrées.
- L_3, esemble des paires de machine de machine de Turing qui calculent la même chose.

Généraliser ces résultats ?

Exemple de réductions

- L_1 ensemble des machines de Turing qui acceptent toutes leurs entrées.
- L_2 ensemble des machine de Turing qui calculent 0 sur toutes les entrées.
- L_3, esemble des paires de machine de machine de Turing qui calculent la même chose.

Généraliser ces résultats ?
C'est le théorème de Rice.

Exemple de réductions

- L_1 ensemble des machines de Turing qui acceptent toutes leurs entrées.
- L_2 ensemble des machine de Turing qui calculent 0 sur toutes les entrées.
- L_3, esemble des paires de machine de machine de Turing qui calculent la même chose.

Généraliser ces résultats ?
C'est le théorème de Rice.

Langages semi-décidable

On peut définir une notion plus large que la décidabilité, en prenant en compte les "boucles infinies".

Definition

On dit qu'un langage L est semi-décidable s'il existe une machine de Turing qui accepte tous les mots $x \in L$ et qui boucle à l'infini sur les mots $x \notin L$.

- Montrer que le problème de l'arrêt est semi-décidable.
- Y-a-t-il des problèmes qui ne sont pas semi-décidables ?