Calculabilité et Complexité (11X008) - Printemps 2024

4. Diagonalisation et indécidabilité

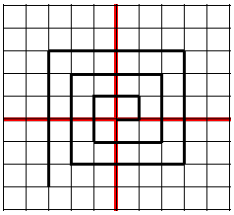
Enseignant: Arnaud Casteigts *Exercices: A. Berger, M. De Francesco, L. Heiniger*

4.1 Infini dénombrable et diagonalisation de Cantor

Lorsque l'on parle d'infini, beaucoup de choses deviennent étranges. Par exemple, y a-t-il plus d'entiers relatifs ($\mathbb{Z}$) que d'entiers naturels ($\mathbb{N}$) ? Intuitivement, il y en a le double, mais $2 \times \infty = \infty$, comment s'y retrouver ? Pour montrer que deux ensembles infinis ont la même taille (on parle de *cardinalité*), il suffit établir une *bijection* entre les deux, c'est à dire mettre en correspondance leurs éléments deux à deux. Par exemple, pour les entiers naturels et les entiers relatifs, on peut les associer comme suit :

0	1	2	3	4	5	6	$\cdots$
0	1	-1	2	-2	3	-3	$\cdots$

On voit bien ici que pour chaque entier naturel i, il existe un $i^{\text{ème}}$ entier relatif, et réciproquement, chaque entier relatif correspond à exactement un entier naturel i. Les deux ensembles ont donc bien la même cardinalité, qui est celle des entiers naturels, aussi appelée $\aleph_0$ ("aleph 0") ou **infini dénombrable** [1]. On dit aussi qu'un tel ensemble peut être *énuméré*. Qu'en est-il des nombres rationels $\mathbb{Q}$? Cet ensemble contient $\mathbb{N}$, il est donc au moins aussi grand. À première vue, il semble beaucoup plus grand car il est *dense* (il possède des élements entre chaque paire d'éléments). Mais en fait, il a la même cardinalité que $\mathbb{N}$, car on peut énumérer ses éléments. Par exemple, si l'on représente chaque rationel p/q par un point en deux dimensions (p, q) (numérateur en abscisses, dénominateur en ordonnée), on peut les énumérer en effectuant un parcourt en spirale comme celui-ci :



Clairement, tous les nombres rationels seront atteints lors de ce parcours, on a donc $|\mathbb{Q}| = |\mathbb{N}| = \aleph_0$. Bien, mais qu'en est-il des réels $\mathbb{R}$?

En 1874, le mathématicien Georg Cantor démontra que l'ensemble $\mathbb{R}$ n'est pas dénombrable, il est strictement plus grand que $\mathbb{N}$. Il s'agit d'une preuve par l'absurde. En résumé, on sup-

1. Le mot est plutôt mal choisi, car il ne s'agit pas de les compter, mais c'est comme ça.

pose que $\mathbb{R}$ est dénombrable. Il existe donc un ordre dans lequel on peut énumérer ses éléments. Mais on montre ensuite qu'en s'appuyant sur un tel ordre, il existe nécessairement des nombres qui ne seront pas atteints (contradiction). L'idée est la suivante, supposons qu'il existe un ordre pour énumérer les réels, par exemple :

Natural	Real
0	0.236436775676...
1	0.098473294543...
2	0.193214042202...
3	0.843279242093...
4	0.012934812343...
5	0.639423412934...
6	0.017773923845...
7	0.238920090909...
8	0.123984732999...
9	0.646329878122...
10	0.000123943437...
11	0.981298312892...
⋮	⋮
	0.293233992132...
	0.746894310875...

Intéressons-nous au nombre formé par la diagonale comprenant la $i^{\text{ème}}$ décimale de la $i^{\text{ème}}$ ligne (en rouge sur le dessin), par exemple ici $x = 0.293233992132\ldots$. On peut alors définir un autre nombre y dont chaque décimale est *différente* de la même décimale sur x (première décimale différente de 2, deuxième différente de 9, etc.), par exemple $y = 0.746894310875\ldots$. Et bien un tel nombre ne sera jamais atteint par l'énumération, car si c'était le cas, par exemple à la ligne j, sa $j^{\text{ème}}$ décimale serait la même que celle de la diagonale... ce qui contredit le fait qu'elle est différente. En fait, il y a plein de choix possibles pour y, et donc plein de nombres qui ne seront jamais atteints (une infinité).

Cet argument peut être appliqué quel que soit l'ordre choisi. L'ensemble $\mathbb{R}$ n'est donc pas dénombrable. On dit que sa cardinalité est $\aleph_1$ (**infini non-dénombrable**).

En quoi cela nous concerne ? La notion d'énumération et de diagonalisation sont très utiles pour étudier les limites de la calculabilité. Voyons cela.

4.2 Énumération des machines de Turing

Lors du dernier cours, nous avons vu que toute machine de Turing déterministe à une bande sur l'alphabet $\{0, 1, \triangleright, \square\}$ (ces hypothèses étant faites sans perte de généralité) peut être décrite sous forme textuelle sur l'alphabet $\{0, 1\}$ (son encodage). Outre la simulation, cet encodage permet également d'énumérer les machines de Turing.

Comment faire ? Tout d'abord, remarquons qu'on peut énumérer tous les mots binaires, par exemple dans l'ordre "shortlex" qui liste les mots par ordre de taille ("short") et par ordre lexicographique à l'intérieur d'une même taille ("lex"), autrement dit : $\mathtt{0}, \mathtt{1}, \mathtt{00}, \mathtt{01}, \mathtt{10}, \mathtt{11}, \mathtt{000}, \ldots$.

On peut coupler cette énumération à une procédure qui vérifie si un mot donné représente bien un encodage valide, à savoir (c.f. cours précédent) commence-t-il et termine-t-il par 111, chaque transition est-elle bien au format $u\texttt{1}v\texttt{1}x\texttt{1}y\texttt{1}z$, *etc.* Si c'est le cas, on peut sélectionner ce mot, et sinon l'ignorer. Un tel processus va ainsi énumérer toutes les machines de Turing possibles.

L'ensemble des machines de Turing est donc un ensemble infini certes, mais dénombrable !

4.3 Un langage non-reconnaissable (langage diagonal)

Nous allons utiliser un argument diagonal pour montrer que certains langages ne sont pas reconnaissables. Soit $M_1, M_2, \ldots$ une énumération des machines de Turing. On note L_i le langage reconnu par M_i, autrement dit, l'ensemble des mot w tels que $M_i(w) = 1$.

Faisons un tableau dont les colonnes correspondent aux machines de Turing M_i et les lignes correspondent à tous les mots w_j (par exemple, dans l'ordre shortlex). Les cases (i, j) du tableau indiquent si le mot w_j fait partie du langage L_i, on écrit donc :

- 1 si $M_i(w_j) = 1$
- 0 si $M_i(w_j) \neq 1$ ou ne termine pas.

Mots \ Machine	M_1	M_2	M_3	M_4	M_5	$\ldots$
ε	**0**	1	0	1	$\ldots$	
0	0	**1**	1	0	$\ldots$	
1	0	0	**0**	0	$\ldots$	
00	1	0	1	**0**	$\ldots$	
01	0	1	0	1	$\ldots$	
10	1	0	0	0	$\ldots$	
11	1	0	0	0	$\ldots$	
000	0	1	0	1	$\ldots$	
$\ldots$	$\ldots$					

Inspiré par l'argument de Cantor, définissons un nouveau langage L_{diag} qui diffère en chaque point de la diagonale. Autrement dit,

$L_{diag} = \{w_i \mid M_i(w_i) = 0\}$. Ici, par exemple, $L_{diag} = \{\varepsilon, 1, 00, \cdots\}$.

Aucune machine M_i ne peut reconnaître ce langage (ni à fortiori le décider). Preuve par l'absurde : soit M_i une machine qui reconnaît L_{diag}. Observons le comportement de M_i sur le mot w_i. Si $M_i(w_i) = 0$, alors ce mot est inclus dans L_{diag} (par définition), mais dans ce cas, il faut que M_i l'accepte (puisqu'elle reconnaît L_{diag}), ce qui implique que $M_i(w_i) = 1$ (contradiction).

En résumé, quel que soit l'ordre considéré pour énumérer des machines de Turing, il existe des langages qu'aucune machine ne reconnaît. Plus généralement, on peut montrer que l'ensemble de tous les langages n'est pas dénombrable, il a la cardinalité des réels $\aleph_1$, tandis que l'ensemble des machines de Turing est dénombrable (cardinalité $\aleph_0$). Il y a donc beaucoup plus de langages que de machines [2], ce qui implique que beaucoup d'entre eux ne sont pas reconnaissables (et encore moins décidables). Bien sûr, ces langages sont définis artificiellement, il ne correspondent pas forcément à un problème "naturel".

4.4 Quelques langages indécidables

4.4.1 Langage universel

On appelle langage universel le langage :

$L_U = \{\langle M\rangle w \mid M(w) = 1\}$

Autrement dit, tous les couples (machine, entrée) tels que cette machine accepte cette entrée. Il est facile de voir que L_U est *reconnaissable* : il suffit de simuler M sur l'entrée w et accepter si $M(w)$ accepte. Mais L_U est-il *décidable* ? On peut montrer que non. Pour montrer cela, on montre que si L_U était décidable, alors cela impliquerait que L_{diag} est décidable. Or, on sait que L_{diag} n'est pas décidable, donc L_U ne peut pas l'être non plus.

Voici la preuve plus en détail : Supposons que L_U est décidable. Il existe donc une machine M_U qui prend en entrée $\langle M\rangle$ et w, qui termine toujours, et renvoit 1 si $M(w) = 1$ et 0 sinon. Nous allons montrer qu'une telle machine permet de décider L_{diag}. Soit x un mot d'entrée pour L_{diag}. On commence par énumérer tous les mots $x_1, x_2, ...$ jusqu'à trouver l'indice i tel que $x_i = x$. On énumère ensuite les machines de Turing pour trouver la $i^{\text{ème}}$ machine M_i. On demande alors à M_U si $\langle M_i\rangle x_i \in L_U$. Si c'est le cas, on renvoit 0 ; sinon, on renvoit 1. Une telle construction va bien renvoyer 1 si $x \in L_{diag}$ et 0 sinon. On a donc réussi à décider L_{diag}. Cette contradiction montre que notre hypothèse était fausse : L_U n'est pas décidable.

Cette technique est très commune : on montre d'abord qu'un langage A peut **se réduire** à un langage B (autrement dit, décider B permet de décider A). Si l'on sait que A est indécidable, on en déduit donc que B est indécidable.

4.4.2 Le problème de l'arrêt (halting problem)

Étant donné la description $\langle M\rangle$ d'une machine M et une entrée w, peut-on décider si M termine ? Autrement dit, peut-on décider le langage L_H (H comme "Halt") suivant :

$$L_H = \{\langle M\rangle w \mid M \text{ termine sur l'entrée } w\}.$$

2. "Beaucoup plus de problèmes algorithmiques que d'algorithmes".

Clairement, L_H est reconnaissable : il suffit de simuler $M(w)$ et d'accepter si M accepte. En revanche, Alan Turing a montré que L_H n'est pas décidable. Nous avons déjà vu une preuve dans le Cours 12 de Langages Formels. Il est aussi possible de démontrer cela plus simplement en utilisant une réduction depuis un autre langage indécidable (vous le ferez peut-être en exercice).

4.4.3 Le problème du langage vide

Étant donné une description de machine $\langle M \rangle$, on souhaite décider si M rejette tous les mots, autrement dit, on s'intéresse au langage :

$L_\emptyset = \{\langle M \rangle \mid L(M) = \emptyset\}$.

Nous allons montrer que ce langage est indécidable. Pour faire cela, nous allons réduire L_U à $L_\emptyset$, c'est à dire montrer que si $L_\emptyset$ était décidable, on pourrait l'utiliser pour décider L_U. Cela contredit l'indécidabilité de L_U, donc $L_\emptyset$ ne peut pas être décidable.

Preuve : Supposons que $L_\emptyset$ est décidable. Il existe donc une machine $M_\emptyset$ qui décide $L_\emptyset$. Nous allons utiliser $M_\emptyset$ pour décider L_U. Une entrée pour L_U se présente comme une machine $\langle M \rangle$ et une entrée w et on veut décider si $M(w) = 1$. Une fois cette entrée reçue, on commence par créer une machine intermédiaire M' qui encapsule M de la façon suivante. Soit x un mot d'entrée pour M' :

- Si $x \neq w$, rejetter
- Sinon, renvoyer $M(x)$ (c.à.d. $M(w)$)

L'intérêt de cette machine est que si $L(M') = \emptyset$, alors $M(w) = 0$ et si $L(M') \neq \emptyset$, alors $M(w) = 1$. Il suffit donc d'utiliser $M_\emptyset$ sur l'entrée $\langle M' \rangle$ pour décider si $M(w) = 1$. Cela montre bien qu'on peut décider L_U si on a accès à $M_\emptyset$. Une telle machine ne peut donc pas exister, ce qui implique que $L_\emptyset$ est indécidable.

4.4.4 Le problème du langage non-vide

Étant donné la description de machine $\langle M \rangle$, on souhaite décider si M accepte au moins un mot, autrement dit, on s'intéresse au langage complément de $L_\emptyset$:

$\overline{L_\emptyset} = \{\langle M \rangle \mid L(M) \neq \emptyset\}$.

Le fait que $L_\emptyset$ soit indécidable implique que $\overline{L_\emptyset}$ l'est aussi. (Souvenez-vous que la famille des langages décidables est stable par complémentation.) On peut cependant observer que $\overline{L_\emptyset}$ est reconnaissable (ce qui n'était pas le cas de $L_\emptyset$). L'idée est de tester M "en parallèle" sur tous les mots possibles jusqu'à ce qu'on en trouve un qui est accepté. La stratégie la plus simple est d'utiliser du non-déterminisme pour cela. P.ex., on part du mot vide et on effectue trois branchements, selon que le symbole suivant vaut 0, 1, ou que le mot s'arrête là. Dans la

branche où il vaut 0, on explore trois nouveaux branchements selon que le deuxième symbole vaut 0, 1, ou le mot s'arrête là, etc. À chaque fois qu'un mot "s'arrête là", on simule M sur ce mot dans la branche courante et on accepte si $M(w)$ accepte. Pour rappel, une machine non-déterministe accepte si et seulement si au moins une branche accepte. Clairement, s'il existe un mot $w \in L(M)$, une des branches d'exécution le trouvera et acceptera.

4.4.5 Problème de correspondance de Post

Le problème de correspondance de Post (PCP) est un problème plus naturel que les précédents. Étant données deux listes de k mots chacune :

$$A = x_1, \ldots, x_k \qquad B = y_1, \ldots, y_k$$

on veut savoir s'il est possible de construire un mot w en concaténant les mots de A, de sorte que la concaténation des mots de B *ayant les mêmes indices* donnent aussi w. Les répétitions sont autorisées.

Par exemple, prenons $A = (\texttt{b}, \texttt{a}, \texttt{ca}, \texttt{abc})$ et $B = (\texttt{ca}, \texttt{ab}, \texttt{a}, \texttt{c})$. On peut trouver une solution qui correspond à la suite d'indices $2, 1, 3, 2, 4$. Cela donne :

`a.b.ca.a.abc` (pour A)

`ab.ca.a.ab.c` (pour B)

Autre exemple utilisant l'alphabet binaire :

$A = (1, 10111, 10)$ et $B = (111, 10, 0)$.

La séquence d'entiers $2, 1, 1, 3$ est une solution :

`10111.1.1.10` (pour A)

`10.111.111.0` (pour B)

En revanche, pour $A = (10, 011, 101)$ et $B = (101, 11, 011)$, il n'existe aucune solution.

Le problème de correspondance de Post est celui de décider s'il existe une solution, autrement dit à décider le langage :

$L_P = \{A = x_1, \ldots, x_k; B = y_1, \ldots, y_k \mid \exists i_1, i_2, ..., i_\ell,\ x_{i_1} x_{i_2} ... x_{i_\ell} = y_{i_1} y_{y_2} ... y_{i_\ell}\}$.

Ce problème est indécidable. Nous ne ferons pas la preuve, mais c'est bon à retenir, car de nombreux problèmes ont été montrés indécidable en utilisant une réduction depuis PCP.