Calculabilité et Complexité (11X008) - Printemps 2026

4. Théorème de Rice, Degrés de Turing, Diagonalisation

Enseignant: Arnaud Casteigts

Assistants: M. De Francesco, M. Marseloo
Moniteurs: E. Bussod, N. Beghdadi

Dans ce dernier cours sur la calculabilité, nous présentons le *théorème de Rice*, qui établit qu'un grand nombre de propriétés sur les programmes sont indécidables. Puis, nous évoquons la notion de *degrés d'indécidabilité de Turing*. Enfin, nous adaptons la technique de *diagonalisation* de Cantor pour montrer qu'il existe des langages non-reconnaissables.

4.1 Théorème de Rice

Le théorème de Rice a des implications très profondes en informatique. Intuitivement, ce théorème établit qu'on ne peut pas vérifier, en général, qu'un programme informatique est correct ou qu'il effectue bien le traitement que l'on souhaite qu'il fasse.

Theorème 4.1 (Rice)**.** *Toute propriété sémantique et non triviale d'un programme est indécidable.*

Décryptons. Ici, "programme" peut être compris comme "machine de Turing" (mais ce théorème reste vrai pour toute notion usuelle de programme). Ensuite, rappelons-nous que pour toute machine M, on note $L(M)$ l'ensemble des mots acceptés par M (langage reconnu par M). Les deux termes importants sont :

- Propriété sémantique : Une propriété P sur une machine M est *sémantique* si elle porte sur $L(M)$. Autrement dit, si $L(M_1) = L(M_2)$, alors soit M_1 et M_2 satisfont toutes les deux P, soit aucune des deux ne satisfait P.
- Propriété non-triviale : Il faut qu'il existe au moins une machine M qui satisfasse P et une machine M' qui ne satisfasse pas P.

Exemples : "M accepte le mot `010`" est une propriété sémantique et non-triviale. En revanche, "M a 5 états" n'est pas une propriété sémantique, et "$L(M)$ est reconnaissable" est une propriété triviale (bien que sémantique), car elle est toujours vraie.

Si ces deux conditions sont vérifiées (sémantique et non-triviale), alors le théorème de Rice nous dit que le langage $L = \{\langle M \rangle \mid M \text{ satisfait } P\}$ est indécidable !

Remarque 4.2. L'implication du théorème n'est pas bidirectionnelle. Par exemple, le fait qu'une propriété de programme ne soit pas sémantique n'implique pas qu'elle est décidable.

Remarque 4.3. Il existe un deuxième théorème de Rice qui s'intéresse à la non-reconnaissabilité, plutôt qu'à l'indécidabilité. Nous ne le verrons pas pour éviter les confusions.

4.1.1 Exemples

Le langage vide (revisité)

$L_\emptyset = \{\langle M \rangle \mid L(M) = \emptyset\}$ (M n'accepte aucun mots)

Nous savions déjà que $L_\emptyset$ est indécidable. Cependant, le théorème de Rice permet d'obtenir le même résultat encore plus facilement qu'avec une réduction. Ici, la propriété P est $L(M) = \emptyset$.

- est-ce une propriété sémantique ?
  $\rightarrow$ Clairement oui : si $L(M_1) = L(M_2)$ alors soit les deux langages sont égaux à $\emptyset$, soit aucun des deux n'est égal à $\emptyset$.
- est-ce une propriété non-triviale ?
  $\rightarrow$ Clairement oui : il existe machine M_1 telle que $L(M_1) = \emptyset$ (par exemple, une machine qui rejette dès qu'elle démarre) et une autre machine M_2 telle que $L(M_2) \neq \emptyset$ (par exemple, une machine qui accepte dès qu'elle démarre).

Le théorème de Rice nous permet déjà de conclure que $L_\emptyset$ est indécidable.

Le langage infini

Peut-on décider si une machine accepte un nombre infini de mots ?

$L_\infty = \{\langle M \rangle \mid |L(M)| = \infty\}$

La propriété P est ici $|L(M)| = \infty$.

- Propriété sémantique ? Oui, si deux machines ont le même langage, alors il est impossible que l'un des deux langages soit de taille infinie et l'autre non.
- Propriété non triviale ? Oui (même exemple que pour $L_\emptyset$).

En invoquant le théorème de Rice, L_∞ est donc indécidable.

4.1.2 Portée du théorème et intuition

Prenons un peu de recul. Imaginons que vous travaillez chez Airbus et vous avez demandé à l'équipe de développement de créer un module qui effectue un certain traitement. Par exemple, une fonction qui calcule le carré d'un nombre (oui, c'est caricaturalement simple).

Vous souhaitez vérifier que le programme développé calcule bien le carré. Et bien le théorème de Rice nous dit que c'est impossible à vérifier en général : certes, il existe de nombreux programmes pour lesquels vous pourriez vérifier cela. Mais *il existe* certains programmes calculant le carré pour lesquels vous n'arriverez pas à vous en assurer. Imaginez par exemple un programme écrit par des développeurs un peu farceurs, qui calcule le carré du nombre passé en entrée seulement après avoir simulé une machine M dont le code $\langle M \rangle$ est contenu dans le programme. Vous ne pourrez décider si ce programme calcule bien le carré du nombre passé en entrée que si vous êtes capable de décider si M termine... glups.

En fait, sans exagérer jusqu'à ce point, il existe de nombreux phénomènes de contamination entre différentes partie d'un programme qui rendent ce type de question difficile ou indécidable dans les programmes suffisamment complexes. Cela explique en partie que nos compilateurs peuvent difficilement anticiper ce qu'il va se passer à l'exécution.

4.2 Degrés de Turing

Soient deux langages L_1 et L_2. Ces langages sont **Turing-équivalents** s'il existe une réduction de L_1 à L_2 et une réduction de L_2 à L_1. Autrement dit, si $L_1 \leq_T L_2$ et $L_2 \leq_T L_1$. On note cela $L_1 \equiv_T L_2$.

Les problèmes indécidables sont-ils tous équivalents ?

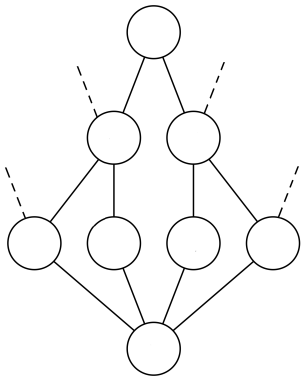
Non ! Étant donné deux langages indécidables L_1 et L_2, on peut être dans l'un des quatre cas suivants :

- $L_1 \equiv_T L_2$ (ils sont Turing-équivalents)
- $L_1 \leq_T L_2$ mais l'inverse n'est pas vrai (L_2 est strictement plus difficile)
- Idem dans l'autre sens (L_1 est strictement plus difficile)
- L_1 et L_2 sont incomparables (aucun des deux ne se réduit à l'autre)

Exemples : Soient les langages :

- $L_H = \{(\langle M \rangle, w) \mid M \text{ termine sur l'entrée } w\}$ // Problème de l'arrêt
- $L_{H^*} = \{(\langle M \rangle) \mid M \text{ termine sur toute entrée}\}$ // Problème de l'arrêt généralisé

Les deux sont indécidables, mais l'un est plus difficile que l'autre, en effet $L_H \leq_T L_{H^*}$, mais $L_{H^*} \not\leq_T L_H$. Lorsque deux langages sont équivalents, on dit qu'ils appartiennent au même **degré de Turing**. Du fait de l'incomparabilité entre certains langages, les degrés de Turing forment un **ordre partiel** comme illustré ci-dessous, où le degré le plus bas correspond aux langages décidables.



Cette classification est infinie : pour n'importe quel langage L, on peut toujours créer un langage L' tel que $L \leq_T L'$ mais $L' \not\leq_T L$. La classification de ces degrés a été un domaine de recherche très actif dans les années 1950-60, il l'est beaucoup moins aujourd'hui.

4.3 Diagonalisation et langage non-reconnaissable

4.3.1 Infini dénombrable

Nous avons déjà parlé de ce contenu au premier semestre, mais il n'était pas écrit. Le voici donc. Lorsqu'on parle d'infini, certaines choses deviennent étranges. Par exemple, il y a intuitivement deux fois plus d'entiers entiers relatifs ($\mathbb{Z}$) que d'entiers naturels ($\mathbb{N}$), 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. Les deux ensembles ont donc bien la même cardinalité, qui est celle des entiers naturels, aussi appelée $\aleph_0$ ("aleph 0") ou **infini dénombrable**. Le mot "dénombrable" n'est pas très bien choisi, on pourrait dire énumérable : on peut lister tous les entiers relatifs en utilisant l'ordre considéré. Certes, ils sont en nombre infini, mais pour *chaque entier relatif*, on a la garantie de l'atteindre un jour ou l'autre.

4.3.2 Énumération des machines de Turing

Lors du dernier cours, nous avons vu que toute machine de Turing peut être encodée sous forme textuelle sur l'alphabet $\{0, 1\}$. Cela implique que l'ensemble des machines de Turing, bien qu'infini, est dénombrable : il est possible d'énumérer les machines en étant sûr d'atteindre un jour ou l'autre n'importe quelle machine particulière.

L'idée est d'énumérer tous les mots binaires dans l'ordre "shortlex", à savoir $\texttt{0}, \texttt{1}, \texttt{00}, \texttt{01}, \texttt{10}$ $\texttt{11}, \texttt{000}, \ldots$. Pour chaque mot, on vérifie s'il s'agit d'un encodage valide dans le format souhaité (p.ex. celui vu au cours $n°2$) et si oui, on considère cette machine. On peut ainsi énumérer toutes les machines de Turing valides $M_1, M_2, \ldots$.

4.3.3 Infini non-dénombrable

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 utilisant un argument de *diagonalisation*. Supposons que $\mathbb{R}$ est dénombrable. Il existe donc un ordre dans lequel on peut énumérer ses éléments. En s'appuyant sur un tel ordre, on peut ensuite montrer qu'il existe nécessairement des nombres qui ne seront jamais atteints, ce qui est une 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 construire un nombre y dont les décimales sont toutes différentes des décimales de x au même endroit (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 est exclu par construction. En fait, il y a plein de choix possibles pour y, et donc plein de nombres qui ne seront jamais atteints.

4.3.4 Un langage diagonal (non-reconnaissable)

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 mots w que M_i accepte.

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). La case (i, j) contient 1 si M_i accepte w_j et 0 sinon (elle rejette 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	$\ldots$	
$\ldots$	$\ldots$					
	1	**0**	**1**	**1**	$\ldots$	

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

$L_{diag} = \{w_i \mid M_i \text{ n'accepte pas } w_i\}$. Ici, par exemple, $L_{diag} = \{\varepsilon, 1, 00, \cdots\}$.

Par l'absurde, s'il existe une machine M_k qui reconnaît L_{diag}, alors cette machine doit se tromper au moins sur le mot w_k, car ce mot appartient à L_{diag} si et seulement si la machine M_k ne l'accepte pas (contradiction).

Plus généralement, on peut montrer avec ce type d'arguments que l'ensemble des langages n'est pas dénombrable, il a la cardinalité des réels $2^{\aleph_0}$, tandis que l'ensemble des machines est "seulement" dénombrable (cardinalité $\aleph_0$). Il y a donc beaucoup plus de langages que de machines, ou dit autrement, beaucoup plus de problèmes que d'algorithmes.