Calculabilité et Complexité (11X008) - Printemps 2024

5. Theorème de Rice (et degrés de Turing)

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

Dans ce cours, nous revenons sur la notion de réduction entre langages. Puis, nous évoquons une hiérarchie d'indécidabilité connue sous le nom de *degrés de Turing*. Enfin, nous terminons avec le Théorème de Rice, qui est un résultat central de la théorie de la calculabilité ayant d'importantes conséquences en informatique.

5.1 Réduction entre langages

Étant donnés deux langages L_1 et L_2, on dit que L_1 est **Turing-réductible** à L_2 (ou juste que L_1 se réduit à L_2) si l'existence d'une machine décidant L_2 permet de décider L_1. On écrit cela $L_1 \leq_T L_2$.

Si l'on sait déjà que L_1 est indécidable, alors cela implique que L_2 l'est aussi. Cette technique permet souvent d'obtenir des résultats d'indécidabilité plus facilement que si l'on cherchait à les obtenir "from scratch". Attention à ne pas se tromper de sens. Il s'agit bien de réduire un langage que l'on sait indécidable *vers* un langage dont on cherche à prouver l'indécidabilité. Une réduction dans l'autre sens n'impliquerait pas grand chose.

La semaine dernière, nous avons d'abord prouvé par un argument de diagonalisation que le langage L_{diag} n'est pas reconnaissable, et à fortiori pas décidable. Puis, nous avons évoqué les réductions suivantes :

- L_{diag} se réduit à L_U, donc L_U est indécidable (c.f. notes du Cours n°4)
- L_U se réduit à $L_\emptyset$, donc $L_\emptyset$ est indécidable (c.f. notes du Cours n°4)
- L_U se réduit à L_H, donc L_H est indécidable (sera fait en exercices)

Pour rappel :

- $L_{diag} = \{w_i \mid M_i(w_i) = 0\}$ (où w_i est le $i^{\text{ème}}$ mot et M_i est la $i^{\text{ème}}$ machine valide dans une énumération shortlex sur l'alphabet $\{0, 1\}$).
- $L_U = \{\langle M \rangle w \mid M(w) = 1\}$
- $L_H = \{\langle M \rangle w \mid M(w) \text{ termine }\}$
- $L_\emptyset = \{\langle M \rangle \mid L(M) = \emptyset\}$ (M n'accepte aucun mots)

5.2 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 $L_1 \equiv_T L_2$.

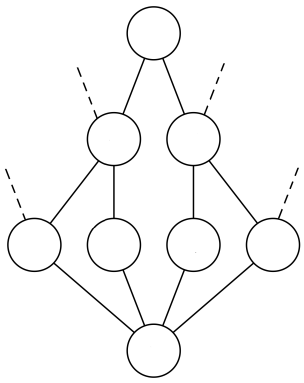
Tous les problèmes décidables sont Turing-équivalents. Cette déclaration est vraie, bien qu'elle ne donne aucune information intéressante. En effet, si L_1 et L_2 sont décidables, alors L_1 peut être décidé en ayant accès à une machine pour L_2, mais c'est tout aussi vrai sans L_2 (puisque L_1 est déjà décidable). Même remarque dans l'autre sens.

Plus intéressant : Les problèmes indécidables sont-ils tous équivalents ?

Non ! Certains langages indécidables sont utiles pour certains autres, mais pas pour tous. 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 fort)
- Idem dans l'autre sens (L_1 est strictement plus fort)
- L_1 et L_2 sont incomparables (aucun des deux ne se réduit à l'autre)

Ces possibilités permettent de classifier les langages par ordre de difficulté, chaque ordre (appelé un **degré de Turing**) regroupant des langages Turing-équivalents. Du fait de l'incomparabilité entre certains langages, il s'agit seulement d'un ordre partiel, tel qu'illustré ci-dessous, où le degré le plus bas correspond aux langages décidables (le nombre de degrés représentés et leurs relations sont complètement arbitraires).



La classification de ces degrés a été un domaine de recherche très actif à partir des années 1950 (un peu moins aujourd'hui).

5.3 Théorème de Rice

Pour terminer notre partie de cours sur la calculabilité, nous allons voir un théorème dont les implications en informatique sont très profondes. 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. En bref, encore une mauvaise nouvelle.

Que dit ce théorème, intuitivement ?

Theorème 5.1 (Rice)**.** *Toute propriété sémantique non triviale d'un programme est indécidable*

Uh ? !

Déjà, remplaçons "programme" par "machine de Turing". Que signifient ensuite "propriété sémantique" et "propriété non triviale".

- **Propriété sémantique** : Il s'agit d'une propriété P qui ne concerne pas les détails de la machine elle-même (la syntaxe), mais seulement le langage qu'elle reconnaît (la sémantique). Concrètement, si M_1 et M_2 sont deux machines telles que $L(M_1) = L(M_2)$, alors soit elles satisfont toutes deux P, soit aucune des deux ne satisfait P.
- **Propriété non triviale** : Il faut que cette propriété ne soit pas satisfaite (ou non satisfaite) par toutes les machines de Turing. Autrement dit, il faut qu'il existe au moins une machine M qui satisfait P et une machine M' qui ne satisfait pas P.

Si ces deux conditions sont vérifiées, alors le théorème de Rice nous dit que le langage $L = \{\langle M \rangle \mid M \text{ satisfait } P\}$ est indécidable.

Remarque 5.2. Il existe un autre théorème de Rice qui permet de montrer qu'un langage n'est pas reconnaissable. Ce théorème est plus compliqué et ne sera pas au programme.

5.3.1 Exemples d'utilisation

Le langage vide (revisité)

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

Nous avons déjà montré l'indécidabilité de $L_\emptyset$ en utilisant une réduction depuis le langage universel L_U. Le théorème de Rice nous permet d'obtenir le même résultat encore plus facilement (presque sans réfléchir). Ici, la propriété P est $L(M) = \emptyset$.

- P est-elle une propriété sémantique ?
  $\rightarrow$ Soient M_1 et M_2 telles que $L(M_1) = L(M_2)$. Clairement, $L(M_1) = \emptyset$ si et seulement si $L(M_2) = \emptyset$ (puisque $L(M_1) = L(M_2)$). P est donc bien une propriété de langage.

- P est-elle une propriété non triviale ?
  $\rightarrow$ On peut facilement créer une machine M telle que $\langle M \rangle \in L_\emptyset$ (par exemple, une machine qui rejette quel que soit le mot d'entrée, avec $L(M) = \emptyset$) et une autre machine telle que $\langle M \rangle \notin L_\emptyset$ (par exemple, une machine qui accepte quel que soit le mot d'entrée, $L(M) = \Sigma^* \neq \emptyset$). P est donc une propriété non triviale.

Le théorème de Rice nous permet 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 ? Soit M_1 et M_2 deux machines telles que $L(M_1) = L(M_2)$. Clairement, $|L(M_1)| = \infty$ si et seulement si $|L(M_2)| = \infty$. Donc oui, P est sémantique.
- Propriété non triviale ? Oui (même exemple que pour $L_\emptyset$).

L_∞ est donc indécidable.

5.3.2 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 bel et bien le carré, dans tous les cas et sans erreur. Et bien le théorème de Rice nous dit que ce n'est pas toujours possible : il existe certains programmes pour lesquels vous ne pouvez pas décider cela. Intuitivement, en prenant un exemple extrêmement absurde, on pourrait imaginer que le programme contient dans une variable la description d'une machine de Turing dont l'arrêt est indécidable. Le code du programme consiste alors à simuler cette machine, puis à renvoyer le carré du nombre demandé. On peut réaliser que ce programme renvoit le carré du nombre si et seulement si la machine termine. Vous ne pouvez donc pas décider s'il fonctionne correctement. OK, c'est tiré par les cheveux (et les développeurs sont de sacrés farceurs...) mais cela donne l'intuition que n'importe quelle propriété sémantique, aussi simple soit-elle, peut être "contaminée" par des traitements plus complexes cachés dans le programme, ce qui rend la vérification de la propriété elle-même indécidable. C'est l'essence du théorème de Rice, et malheureusement, ce type de problèmes peuvent se produire avec des programmes tout à fait normaux.