Langages formels (11X003) - Automne 2023

12. Universalité et indécidabilité

Enseignant: Arnaud Casteigts *Assistant: Alexandre-Quentin Berger*

Nous commençons par rappeler la thèse de Church-Turing. Puis nous mentionnons la notion de machine de Turing universelle. Enfin, nous plongeons dans la démonstration d'Alan Turing, qui a montré en 1936 que certaines questions naturelles ne sont pas décidables.

12.1 Thèse de Church-Turing

La notion d'algorithme existe depuis l'antiquité (au minimum), on peut citer par exemple l'algorithme d'Euclide pour trouver le PGCD de deux nombres. Cependant, c'est seulement au vingtième siècle que la notion d'algorithme a été définie de manière précise, par Alonzo Church et Alan Turing. Ils ont stipulé que tout algorithme concevable pouvait être spécifié de manière rigoureuse, pour le premier en utilisant le λ-calcul et pour le second les machines de Turing, les deux étant équivalents. En fait, tous les modèles de calcul imaginés depuis, du moment qu'ils peuvent accéder à une mémoire infinie et de manière libre, se sont avérés équivalents aux machines de Turing. Il est aujourd'hui admis que ces dernières capturent ce qui est calculable (y compris par un ordinateur quantique). C'est ce qu'on appelle la **thèse de Church-Turing**.

Une fois cette correspondance admise, il n'est pas nécessaire de décrire un algorithme sous forme de machine de Turing (et heureusement !). On peut souvent se contenter d'utiliser un plus haut niveau d'abstraction, comme par exemple les langages de programmation modernes. D'ailleurs, tous les langages de programmations "généralistes" (C, Python, Java, etc.) sont équivalents aux machines de Turing. On dit qu'ils sont **Turing-complets**.

12.2 Machine de Turing universelle

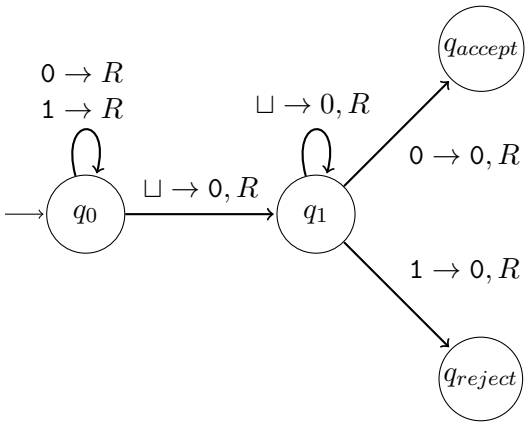
Les machines de Turing représentent-elles des programmes, ou des ordinateurs ? Nos ordinateurs semblent plus généraux, car ils ne correspondent pas à *un seul* programme, ils peuvent exécuter n'importe quel programme. C'est en fait la même chose avec les machines de Turing : on peut concevoir une machine de Turing qui prend en entrée la description d'une autre machine de Turing M et une seconde entrée E, et qui simule l'exécution de M sur l'entrée E. Finalement, un ordinateur n'est qu'un programme qui exécute un autre programme. On parle alors de **machine de Turing universelle**.

Pour la suite, retenons qu'il est possible de concevoir une machine qui prend en entrée

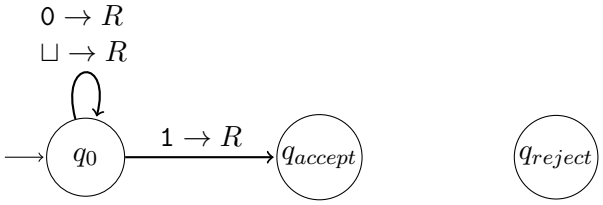
la description d'une autre machine et une entrée pour cette machine, et qui en simule l'exécution.

12.3 Le problème de l'arrêt (halting problem)

Soit la machine suivante, dont l'alphabet d'entrée est $\Sigma = \{0, 1\}$:



Que fait cette machine ? Termine-t-elle quelle que soit l'entrée ? Termine-t-elle au moins pour certaines entrées ? (Hum...) La machine suivante, quant à elle, termine parfois, mais pas toujours, cela dépend de l'entrée. Par exemple, elle termine sur l'entrée `0010`, mais boucle infiniment sur l'entrée `000`.



Soit M une machine, notons $\langle M \rangle$ sa description. Peut-on concevoir une machine de Turing qui, recevant en entrée $\langle M \rangle$ et une seconde entrée E, décide si M termine sur l'entrée E ? On peut représenter cela comme un langage L_H (H comme "Halt") :

$$L_H = \{(\langle M \rangle, E) \mid M \text{ termine sur l'entrée } E\}.$$

Une question naturelle est la suivante : le langage L_H peut-il être reconnu par une machine de Turing ? C'est la question que s'est posée Alan Turing en 1936.

12.4 Turing-reconnaissable *ou* Turing-décidable ?

Pour rappel, nous avons défini les langages **Turing-reconnaissable** comme étant les langages tels qu'il existe une machine de Turing qui accepte un mot w si et seulement si $w \in L$. Le langage L_H est-il Turing-reconnaissable ?

Oui ! Étant donné $\langle M \rangle$ et E, on peut utiliser une machine de Turing universelle M_U qui simule l'exécution de M sur l'entrée E et *accepte* quand cette exécution termine. Ce langage est bien Turing-reconnaissable : si $(\langle M \rangle, E) \in L_H$, alors M_U finira par accepter. Et si $(\langle M \rangle, E) \notin L_H$, alors elle n'acceptera pas ! La subtilité est qu'il y a deux façon possibles de ne pas accepter : soit rejeter, soit boucler à l'infini, et tant que M_U s'exécute, on ne reconnaît pas la réponse (qui pourrait ne jamais arriver).

Pour clarifier cela, on a besoin d'une nouvelle notion, plus forte que la précédente. Un langage L est **Turing-décidable** s'il existe une machine qui accepte tous les mots de L *et rejette* tous les mots de $\overline{L}$.

Le langage L_H est-il Turing-décidable ? Hélas non...

12.5 Indécidabilité du problème de l'arrêt

Pour démontrer cela, Turing effectue un raisonnement par l'absurde. Il suppose d'abord qu'il existe une machine M_H capable de décider le langage L_H, puis il en déduit que cette machine se trompe (une contradiction). Regardons cela plus en détails !

Supposons que M_H existe.

Intéressons-nous d'abord à une question plus spécifique : est-ce qu'une machine termine lorsqu'elle prend sa propre représentation en entrée ? C'est un cas particulier du problème de l'arrêt correspondant au langage L_S suivant (S comme "Self") :

$$L_S = \{\langle M \rangle \mid M \text{ termine sur l'entrée } \langle M \rangle\}$$

Ce langage est facile à reconnaître si l'on dispose de M_H, il suffit de créer une machine M_S qui l'utilise comme suit :

$M_S(\langle M \rangle)$:
    Si $M_H(\langle M \rangle, \langle M \rangle)$ accepte, alors :
        Accepter
    Sinon :
        Rejeter

Puisqu'on a l'esprit contradictoire, on voudrait maintenant créer une dernière machine M_C (C comme "Contradictoire") qui se comporte de manière opposée à la machine M testée : si M termine sur $\langle M \rangle$, M_C entre dans une boucle infinie ; sinon, M_C termine. Il suffit pour cela de modifier légèrement M_S comme suit :

$M_C(\langle M \rangle)$:
    Si $M_H(\langle M \rangle, \langle M \rangle)$ accepte, alors :
        Boucler à l'infini
    Sinon :
        Terminer

Jusqu'ici, tout va bien. Maintenant, que se passe-t-il si l'on donne à M_C sa propre description ? Autrement dit, quel est le comportement de $M_C(\langle M_C \rangle)$. Tout d'abord, cela va causer l'appel $M_H(\langle M_C \rangle, \langle M_C \rangle)$. Deux possibilités :

- Si M_H accepte, cela implique que $M_C(\langle M_C \rangle)$ est censée terminer, mais dans ce cas elle boucle à l'infini.
- Si M_H rejette, cela implique que $M_C(\langle M_C \rangle)$ est censée boucler à l'infini, mais dans ce cas elle termine.

Huh ?

Bref, si M_H existe, alors M_H doit se tromper, donc M_H ne peut pas exister. □

12.6 Autres problèmes non décidables / non reconnaissables

Il existe de nombreux problèmes "naturels" qui ne sont pas Turing-décidables. Par exemple, certaines équations diophantiennes [1] sont indécidables. Un autre exemple célèbre est celui du problème de correspondance de Post (PCP). Plus proche de ce cours : savoir si un automate à pile (dont la description est donnée) reconnaît le langage Σ^* est indécidable. Et plus généralement, le théorème de Rice nous dit que toute question "non-triviale" (dans un sens précis) sur un programme informatique est indécidable.

Bon, mais comme discuté plus haut, la décidabilité est plus exigente que la reconnaissabilité (par exemple, le problème de l'arrêt est Turing-reconnaissable). La situation est-elle meilleure si on se contente de reconnaître plutôt que de décider ? Un peu, certes, mais pas tant que ça. En fait, le nombre de langages possibles est non-dénombrable (cardinalité de l'infini des nombres réels), alors que le nombre de machines de Turing est dénombrable (cardinalité de l'infini des nombres entiers). Chaque machine reconnaissant un seul langage, cela implique qu'une infinité de langages ne sont même pas reconnaissables. Rassurez-vous, beaucoup d'entre eux ne sont pas intéressants non plus.

1. Équations polynomiales à une ou plusieurs inconnues dont les solutions et les coefficients sont des nombres entiers.