Langages formels (11X003) - Automne 2024

12. Universalité et indécidabilité

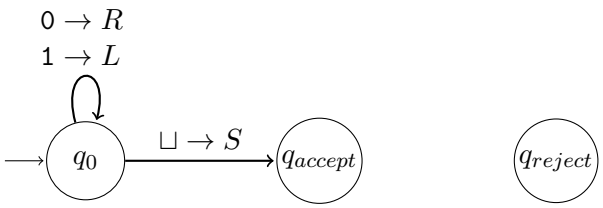
Enseignant: Arnaud Casteigts

Assistants: A.-Q. Berger & M. De Francesco
Monitrices: L. Heiniger & A. Tekkoyun

Dans le cours précédent, nous avons évoqué la thèse de Church-Turing, qui stipule que les machines de Turing peuvent modéliser n'importe quel traitement physiquement réalisable. Il est alors très pertinent de s'interroger sur les capacités de ces machines. Peuvent-elles tout calculer ? Peuvent-elles reconnaître n'importe quel langage ? Nous allons aujourd'hui présenter la notion de machine de Turing *universelle*, qui joue un rôle clé dans ces réflexions. Nous ferons également la distinction entre les langages Turing-*reconnaissables* et les langages Turing-*décidables*. Enfin, nous plongerons dans la démonstration d'Alan Turing, qui a montré en 1936 que certaines questions naturelles ne sont pas décidables par ces machines (et accessoirement, par nos ordinateurs actuels).

12.1 Reconnaissable *versus* décidable

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



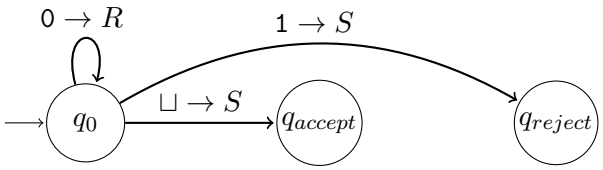
Que fait cette machine ? Quel langage reconnaît-elle ?

Pour rappel, une machine de Turing **reconnaît** un langage L si elle accepte tous les mots $w \in L$, et seulement ceux-là. La machine ci-dessus reconnaît le langage $L = \{0^i \mid i \geq 0\}$. En effet, tous les mots de L seront acceptés, et aucun autre mot ne sera accepté. Mais que se passe-t-il pour ces autres mots ? La définition de "reconnaître" laisse deux options ouvertes : soit la machine les rejette, soit elle boucle à l'infini. Ces deux comportements sont tout à fait valides pour cette définition. Si l'on veut une machine qui termine à tous les coups, on a besoin d'une notion plus forte.

Une machine de Turing **décide** un langage L si :

1. Elle accepte tous les mots de L.
2. Elle rejette tous les mots de $\overline{L}$ (le complément de L).

Ainsi, la machine précédente *reconnaît* L, mais elle ne *décide* pas L, car elle ne rejette pas les mots de $\overline{L}$. La machine suivante décide bien L :



Bien sûr, une machine qui décide un langage le reconnaît aussi (à fortiori).

12.1.1 Familles de langages correspondantes

La distinction entre *reconnaître* et *décider* n'existait pas pour les modèles de machines plus simples, comme les automates finis ou les automates à piles. Mais pour les machines de Turing, on a donc deux grandes familles de langages :

- Un langage L est **Turing-reconnaissable** (aussi appelé *récursivement énumérable*) s'il existe une machine de Turing qui reconnaît L.
- Un langage L est **Turing-décidable** (aussi appelé *récursif*) s'il existe une machine de Turing qui décide L.

On enlève souvent le préfixe "Turing-" en parlant simplement de langages reconnaissables et de langages décidables. Tous les langages décidables sont reconnaissables, puisqu'une machine qui décide L reconnaît aussi L. Mais l'inverse n'est pas vrai, il existe des langages qui sont reconnaissables mais pas décidables. Au final, nous avons la hiérarchie suivante :

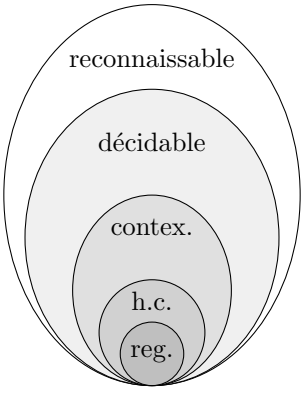


FIGURE 1 – Hiérarchie de familles de langages.

12.2 Machine de Turing universelle

Les machines de Turing représentent-elles des programmes, ou des ordinateurs ? Autrement dit, sont-elles des modèles de machines ou d'algorithme ? Nous avons maintenu cette ambiguité jusqu'à présent, c'était volontaire.

À première vue, les ordinateurs sont plus généraux, car ils ne correspondent pas à *un seul* programme, ils peuvent exécuter n'importe quel programme qu'on leur donne à exécuter, tandis qu'une machine de Turing correspond à un seul programme. Euh ? Et bien non, c'est en fait la même chose avec les machines de Turing : on peut concevoir une **machine de Turing universelle** M_U, 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. Les machines de Turing correspondent donc à la fois à nos ordinateurs et aux programmes qu'ils exécutent (et toc !).

Pour éviter les confusions, on notera $\langle M \rangle$ la description d'une machine M.

12.3 Problème de l'arrêt

Étant donné qu'on peut fournir à une machine la description d'une autre machine, il devient pertinent de s'intéresser à des *langages de machines*, par exemple $L = \{\langle M \rangle \mid M$ fait ceci$\}$, $L = \{\langle M \rangle \mid M$ fait cela$\}$, ou encore $L = \{\langle M \rangle \mid M$ a 5 états et 10 transitions$\}$. On peut alors se demander s'il existe une machine capable de reconnaître ou décider de tels langages. C'est ce que Turing a fait.

En particulier, Turing s'est intéressé au langage suivant :

$$L_H = \{(\langle M \rangle, E) \mid M \text{ termine sur l'entrée } E\} \qquad \text{(H pour "Halt")}$$

Autrement dit, étant donné la description d'une machine $\langle M \rangle$ et une seconde entrée E, on souhaite déterminer si M termine sur l'entrée E. Ce langage est-il décidable ?

Commençons doucement, et demandons-nous d'abord si L_H est *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 (nous verrons comment faire cela une autre fois). Il suffit alors que notre machine M_U attende que la simulation se termine, puis qu'elle *accepte*. Clairement, une telle machine accepte $(\langle M \rangle, E)$ si et seulement si $(\langle M \rangle, E) \in L_H$. Très bien, L_H est donc reconnaissable. Mais que se passe-t-il si $(\langle M \rangle, E) \notin L_H$. Dans ce cas, M_U bouclera elle-même à l'infini. C'est quand même embêtant, car vous ne pouvez pas distinguer les entrées dont l'exécution est longue de celles qui ne s'arrêtent pas. En fait, reconnaître seulement ce langage n'est pas satisfaisant, on aimerait plutôt le décider, en rejetant également $(\langle M \rangle, E)$ lorsque l'exécution de M ne termine pas sur l'entrée E.

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

12.4 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 une contradiction. Cela implique qu'une telle machine ne peut pas exister (ou que si elle existe, elle doit parfois se tromper, ce qui revient au même). Regardons cela plus en détails !

Supposons que M_H existe.

Intéressons-nous d'abord à un cas particulier : est-ce qu'une machine donnée termine lorsqu'elle prend sa propre description comme entrée ? Cela correspond au langage L_S suivant :

$$L_S = \{\langle M \rangle \mid M \text{ termine sur l'entrée } \langle M \rangle\} \qquad (S \text{ comme "Self"})$$

Ce problème est bien un cas particulier du problème de l'arrêt, on devrait donc pouvoir le décider facilement si l'on dispose de la machine M_H.

En effet, il suffit de créer une machine M_S qui utilise M_H comme suit :

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

On récapitule : si M_H existe, alors M_S existe. Très bien.

On voudrait maintenant créer une troisième machine M_C (C comme "Contradictoire"), qui se comporte un peu comme M_S, c'est à dire qu'elle prend en entrée la description d'une machine $\langle M \rangle$ et s'intéresse au comportement de cette machine lorsqu'elle prend sa propre description en entrée. Mais au lieu de décider si $M(\langle M \rangle)$ termine, elle va plutôt faire l'inverse de $M(\langle M \rangle)$: elle va terminer si $M(\langle M \rangle)$ boucle à l'infini, et boucler à l'infini si $M(\langle M \rangle)$ termine.

Là encore, si l'on dispose de M_H, c'est facile, il suffit de l'utiliser 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. Quoique ?

L'estocade

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 ? Mais c'est une contradiction ! Nos hypothèses de départ étaient donc mauvaises : M_H ne peut pas exister (ou si elle existe, elle se trompe forcément).

Le langage L_H n'est donc pas décidable. □

12.5 Autres problèmes non décidables / non reconnaissables

Il existe de nombreux problèmes "naturels" qui ne sont pas 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 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 pas non plus 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.