Langages formels (11X003) - Automne 2025

12. Universalité et indécidabilité

Enseignant: Arnaud Casteigts

Assistants: A.-Q. Berger & M. Marseloo
Moniteurs: N. Beghdadi & E. Bussod

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 Machine de Turing universelle

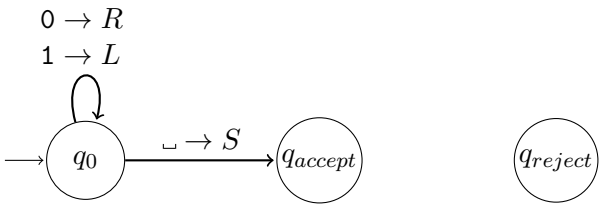
Les machines de Turing représentent-elles des programmes ou des ordinateurs ? Nous avons maintenu cette ambiguité jusqu'à présent, c'était volontaire.

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, non ? En fait non, c'est la même chose : on peut concevoir une **machine de Turing universelle** M_U, qui prend en entrée la *description* $\langle M \rangle$ d'une autre machine de Turing M la description d'une entrée E pour M, et qui simule l'exécution de $M(E)$. Les machines de Turing correspondent donc à la fois à nos ordinateurs et aux programmes qu'ils exécutent !

Cela signifie aussi, et c'est bien pratique, que l'on peut concevoir des machines qui utilisent d'autres machines, de la même manière qu'un programme utilise des fonctions.

12.2 Reconnaissable ou décidable ?

La semaine dernière, nous avons vu un exemple de machine (sur l'alphabet $\Sigma = \{0, 1\}$) qui ne termine pas systématiquement, cela dépend du mot d'entrée :



Quel langage cette machine reconnaît-elle ? En fait, elle reconnaît exactement le langage $L = \{0^i \mid i \geq 0\}$. En effet, tout mot de L sera accepté, et aucun mot de $\overline{L}$ ne sera accepté. La nouveauté est qu'il y a deux manière possibles de ne pas accepter un mot : soit on le rejette, soit on boucle à l'infini (comme avec le mot `01` pour cette machine). Cela donne lieu à deux notions différentes :

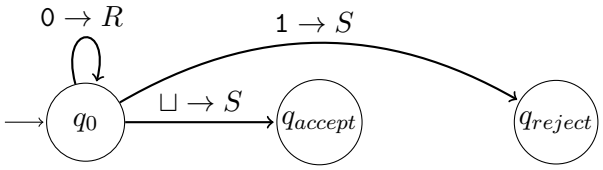
Une machine M **reconnaît** un langage L ssi :

- Pour tout mot $w \in L$, M accepte w.
- Pour tout mot $w \in \overline{L}$, soit M rejette w, soit elle boucle à l'infini.

Une machine M **décide** un langage L ssi :

- Pour tout mot $w \in L$, M accepte w.
- Pour tout mot $w \in \overline{L}$, M rejette w.

Décider un langage est donc une notion plus forte que reconnaître un langage. La machine ci-dessus reconnaît bien $L = \{0^i \mid i \geq 0\}$, mais elle ne le décide pas. En revanche, la machine suivante le décide :



Cette distinction entre *reconnaître* et *décider* n'existait pas pour les AF et les AP, c'est une nouveauté. Cela donne lieu à deux familles de langages distinctes :

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

Un langage est **Turing-décidable** (aussi appelé *récursif*) s'il existe une machine de Turing qui le décide.

On enlève souvent le préfixe "Turing-". Par définition, tous les langages décidables sont reconnaissables. Existe-t-il des langages non-décidables et/ou non-reconnaissables ?

12.3 Problème de l'arrêt

Comme évoqué plus haut pour les machines universelles, on peut fournir à une machine la description d'une autre machine. Il devient alors pertinent de s'intéresser à des *langages de machines*, par exemple $L = \{\langle M \rangle \mid M \text{ écrit sur la bande}\}$ ou $L = \{\langle M \rangle \mid M \text{ a 5 états}\}$, en se demandant si ces langages peuvent être reconnus ou décidés par une machine.

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

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

Turing s'est demandé s'il existe une machine capable de décider L_H.

Commençons doucement, et demandons-nous d'abord si L_H est *reconnaissable*.

Oui ! Étant donné $\langle M \rangle$ et E, on peut utiliser la machine universelle M_U qui simule $M(E)$ (M sur l'entrée E), puis accepte quand $M(E)$ termine :

$M_H(\langle M \rangle, E)$:
    $M_U(\langle M \rangle, E)$
    Accepter

Clairement, une telle machine accepte $(\langle M \rangle, E)$ si et seulement si $(\langle M \rangle, E) \in L_H$. Le langage 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. 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.

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

12.3.1 Indécidabilité

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. Regardons cela plus en détails !

Supposons que M_H existe.

Intéressons-nous 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 :

$M_S(\langle M \rangle)$:

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 ressemble un peu à M_S, c'est à dire qu'elle prend $\langle M\rangle$ en entrée et s'intéresse au comportement de $M(\langle M\rangle)$. Mais au lieu de décider si $M(\langle M\rangle)$ termine, on veut que M_C fasse l'inverse de $M(\langle M\rangle)$: Si $M(\langle M\rangle)$ boucle à l'infini, alors M_C doit terminer, et si $M(\langle M\rangle)$ termine, M_C doit boucler.

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. Euh, tout va bien ?

Que se passe-t-il maintenant si l'on exécute $M_C(\langle M_C\rangle)$? 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.

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).

Le langage L_H n'est donc pas décidable. □

12.4 Autres problèmes non décidables / non reconnaissables

Il existe de nombreux problèmes "naturels" qui ne sont pas décidables. Par exemple, déterminer si une équation diophantienne admet des solutions entières est indécidable. 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) accepte tous les mots 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. Bien

sûr, cela ne veut pas dire qu'on y arrive jamais, seulement qu'il n'existe pas d'algorithme qui fonctionne *pour toutes les entrées possibles*.

Au final, nous avons la hiérarchie suivante :

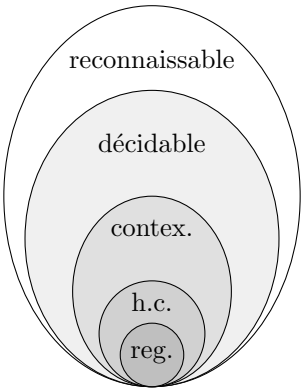


FIGURE 1 – Hiérarchie de familles de langages.

Tous les langages sont-ils reconnaissables ?

Hum... 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 que la majorité des langages ne sont pas reconnaissables. Fort heureusement, beaucoup d'entre eux ne sont pas intéressants non plus.

FIN