CALCULABILITÉ ET COMPLEXITÉ

Auteur
Yago Iglesias

25 décembre 2024

Table des matières

Ce document est un recueil de notes du cours de Calculabilité et Complexité de niveau M1, portant exclusivement sur la partie consacrée à la Calculabilité. Il est basé sur les cours de M. Hugo Férée à l'Université Paris Cité. Cependant, toute erreur ou inexactitude est de ma responsabilité.

Ce document a été rédigé principalement par Yago Iglesias, mais tout contributeur peut être retrouvé dans la section contributeurs du répertoire GitHub. Un remerciement particulier est adressé à Erin Le Boulc'h pour sa participation active à la rédaction et correction de ce document.

1 Introduction

Dans ce cours, on s'intéresse à la notion de calculabilité, mais pour cela, il faut comprendre ce qu'est un problème. D'une manière informelle, on peut voir un problème comme une fonction qui prend en entrée des données et retourne une réponse binaire. On fait la distinction entre un problème et une fonction qui, au lieu de retourner une réponse binaire, renvoie un nouvel ensemble de données.

Définition 1.1. Un **problème** est une fonction $f : \Sigma^* \to \{0, 1\}$, où Σ est un alphabet fini.

Remarque 1.0.1. *L'ensemble des problèmes est infini non dénombrable et il est en bijection avec* $\mathcal{P}(\Sigma^*)$.

Remarque 1.0.2. *L'ensemble des fonctions (en utilisant la définition de fonction donnée) est l'ensemble des fonctions* $f : \Sigma^* \to \Sigma^*$ *qui est infini non dénombrable. De plus, l'ensemble des problèmes est un sous-ensemble de l'ensemble des fonctions à isomorphisme près.*

Il nous manque maintenant la notion de programme pour pouvoir parler de calculabilité.

En 1900, David Hilbert a posé 23 problèmes mathématiques qu'il considérait comme les plus importants de son époque. L'un de ces problèmes, le 10e, consistait à trouver une méthode (un nombre fini d'étapes) pour décider si une équation diophantienne a une solution entière. Cependant, aucune méthode n'a été trouvée, car il n'en existe pas. Pouvoir faire ce genre de démonstration relève de la calculabilité. Une synthèse de cette preuve peut être retrouvée dans [Matiyasevich(1993)].

Regardons quelques exemples de constructions qui sont normalement associées à la notion de programme :

— Les langages de programmation comme Python, Java, C, etc. (3)
— Le λ-calcul (3)
— Les machines de Turing (3)
— Les automates $\iff$ les expressions régulières (1)
— Les automates à pile $\iff$ les grammaires hors-contexte (2)
— Les automates à 2 ou plusieurs piles (3)
— Les transducteurs

Ces constructions sont classées selon la hiérarchie de Chomsky 1.

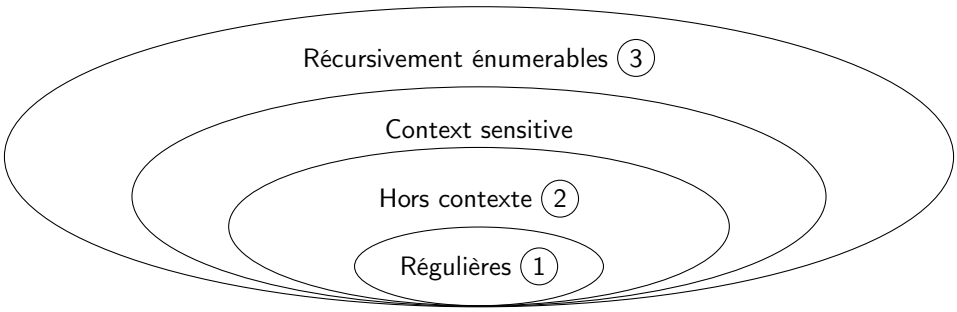


FIGURE 1 – Hiérarchie de Chomsky

La thèse de Church-Turing est une hypothèse qui postule que la seule notion de problème décidables est (3).

2 Machines de Turing

2.1 Définition

Définition 2.1 (Machine de Turing)**.** Étant donné un alphabet Σ, une **machine de Turing** est un 6-uplet $M = (Q, \Gamma, \delta, q_0, q_a, q_r)$ où :

— Q est un ensemble fini d'états
— Γ est un alphabet fini de symboles de ruban, et $\Sigma \subseteq \Gamma$
— δ est la fonction de transition

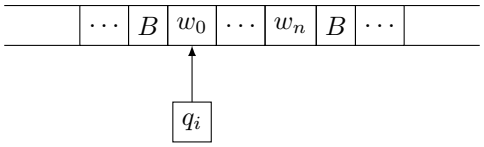
$$\delta : \underbrace{Q}_{\text{État courant}} \times \underbrace{\Gamma}_{\text{Lettre lue}} \to \underbrace{Q}_{\text{Nouvel état}} \times \underbrace{\Gamma}_{\text{Lettre écrite}} \times \underbrace{\{R, L, N\}}_{\text{Direction}}$$

— $q_0 \in Q$ est l'état initial
— $q_a \in Q$ est l'état d'acceptation
— $q_r \in Q$ est l'état de rejet

Définition 2.2 (Configuration)**.** Une **configuration** d'une machine de Turing est un triplet (q, c, pos) où :
— $q \in Q$ est l'état courant
— c est le contenu du ruban
— pos est la position de la tête de lecture

Comment se déroule une exécution d'une machine de Turing pour un mot $w \in \Sigma^*$?

1. On initialise le ruban



2. Si on est dans l'état q, que la lettre sous la tête de lecture est a et que $\delta(q, a) = (q', b, d)$, alors on fait :
   — On écrit b à la place de a
   — On se déplace dans l'état q'
   — On déplace la tête de lecture dans la direction d
3. Si on arrive dans l'état q_a ou q_r, alors on arrête l'exécution. Si on arrive dans l'état q_a, alors on accepte le mot w et si on arrive dans l'état q_r, alors on rejette le mot w.

Notation 2.3. Soit M une machine de Turing, $w \in \Sigma^*$ un mot, alors on note $M(w)$ l'exécution de M sur w. Cette exécution peut être :
— Acceptée : $M(w) = 1$
— Rejetée : $M(w) = 0$
— Bouclée : $M(w) = \perp$

Remarque 2.1.1. *Les automates s'injectent dans les machines de Turing, un automate est une machine de Turing qui se déplace tout le temps vers la droite.*

2.2 Machines de Turing non déterministes

Définition 2.4 (Machine de Turing non déterministe)**.** De manière analogue aux automates la notion de **machine de Turing non déterministe** étend la définition d'une machine de Turing

en permettent d'avoir plusieurs transitions pour un état donné. La différence se trouve donc dans le type de la fonction de transition :

$$\Delta : Q \times \Gamma \to \mathcal{P}\left(Q \times \Gamma \times \{R, L, N\}\right)$$

Proposition 2.5. *Les machines de Turing déterministes et non-déterministes reconnaissent les mêmes langages.*

Démonstration. L'idée derrière la preuve est d'explorer les branches de l'évaluation non déterministe de la machine. Cette exploration se fait en largeur, afin d'éviter de suivre une branche qui boucle alors qu'une autre branche acceptante aurait pu être explorée.

La description de la machine a été étudiée en TD, et elle est également disponible dans la preuve de [Sipser(1996), Theorem 3.16]. □

3 Notion de calculabilité

3.1 Langages décidables et semi-décidables

Définition 3.1 (Langage semi-décidable)**.** Un langage $L \subseteq \Sigma^*$ est **semi-décidable** s'il existe une machine de Turing, M_L, telle que

$$\forall w \in \Sigma^*, w \in L \iff M_L(w) = 1$$

Définition 3.2 (Langage décidable)**.** Un langage $L \subseteq \Sigma^*$ est **décidable** s'il existe une machine de Turing, M_L, telle que

$$\forall w \in \Sigma^*, w \in L \implies M_L(w) = 1 \quad \text{et} \quad w \notin L \implies M_L(w) = 0$$

et donc M_L s'arrête pour tout w.

Proposition 3.3. *Tout langage décidable est semi-décidable.*

Démonstration. Il suffit de monter que si M est une machine de Turing qui décide L, alors $M(w) = 1 \iff w \in L$.

- $w \in L \implies M(w) = 1$ est vrai par définition.
- $M(w) = 1 \implies w \in L$ peut être montrée par contraposée. Si $w \notin L$, alors $M(w) = 0$ car M décide L et donc $M(w) \neq 1$.

□

Définition 3.4 (eval)**.** On note $eval(\langle M \rangle, w, n)$ la machine que simule l'exécution de M sur w en au plus n étapes.

Proposition 3.5 (Admis)**.** *eval est décidable.*

Proposition 3.6. *L est décidable $\iff$ L est semi-décidable* et *L^c est semi-décidable*

Démonstration.

- $\boxed{\Rightarrow}$ est triviale.

— $\boxed{\Leftarrow}$
On peut exécuter les deux machines "en parallèle", c'est-à-dire simuler une étape de calcul de M_L puis une étape de M_{L^c}, en alternant ainsi jusqu'à ce que l'une des deux machines s'arrête. Cette machine est décidable, car elle s'arrête nécessairement : pour tout w, soit $w \in L$, soit $w \notin L$.

Ce théorème et sa preuve détaillée correspondent à ceux présentés dans [Sipser(1996), Theorem 4.22]. □

3.2 Fonctions calculables

Définition 3.7 (Fonction calculable)**.** $f : \Sigma^* \to \Sigma^*$ est calculable si $\exists M$ tel que $\forall w \in \Sigma^*, M$ s'arrête sur w avec $f(w)$ sur le ruban.

Lemme 3.8. *La composition de deux fonctions calculables est calculable*

Lemme 3.9. *La fonction succ* $: \Sigma^* \to \Sigma^*$ *est calculable.*

Proposition 3.10. *L est décidable $\iff$ sa fonction caractéristique est calculable.*

Définition 3.11. $f : \Sigma^* \to \Sigma^*$ énumère $L \subseteq \Sigma^*$ si $\mathrm{Im}(f) = L$ *i.e.* $\forall w \in \Sigma^*, w \in L \iff \exists w' \in \Sigma^*, f(w') = w$.

Définition 3.12 (Récursivement énumerable)**.** Un langage est récursivement énumerable s'il existe une fonction qui l'énumère.

Proposition 3.13. *L est récursivement énumerable $\iff$ L est semi-décidable.*

Démonstration.

— $\boxed{\Leftarrow}$
Soit M_L qui semi-decide L. On pose M_f la machine qui pour un mot $u \in \Sigma^*$, avec $n = |u|$, fait :

1. Énumere toutes les paires $(w, k) \in \Sigma^* \times \mathbb{N}$.
2. Simule l'exécution de M_L sur l'entrée w en au plus k étapes.
3. Retourne le n-ième mot accepté.

Montrons que cette machine énumere L. Soit $w \in \Sigma^*$:

— $\boxed{\Rightarrow}$
Si $w \in L$, alors M_L accepte w un en nombre k d'étapes de calcul. Donc il existe $u \in \Sigma^*$ tel que (w, k) soit la $|u|$-iéme paire accepté. Et donc $M_f(u) = w$.

— $\boxed{\Leftarrow}$
Si w est énumeré par M_f, alors $w \in L$ car $\exists k$ tel que M_L l'accepte en k étapes.

— $\boxed{\Rightarrow}$
Si $\exists M_f$ qui énumere L, alors on a que $L = \{f(w) \mid w \in \Sigma^*\}$. Alors on pose M_L la machine qui pour w, énumere les $w' \in \Sigma^*$ jusqu'à trouver $M_f(w') = w$ et qui accepte.

□

3.3 Quelques propriétés des langages semi-décidables

Proposition 3.14. *Tout langage semi-décidable infini contient un langage décidable infini.*

Démonstration. Soit L un langage semi-décidable infini et soit M_L la machine qui le semi-décide. Alors on pose $M_{L'}$ la machine suivante qui pour un mot $w \in \Sigma^*$:

1. Énumere les paires (w_i, k_i).
2. Pour chaque paire évalue M_L sur w_i en moins de k_i étapes et :
   Si M_L accepte alors :
   — Si $w = w_i$ alors on accepte le mot
   — Si $|w| \leq |w_i|$ alors on rejette le mot.
   Dans tout les autres cas on évalue la paire suivante.

On a bien que L', le langage reconnu par $M_{L'}$, est un sous ensemble de L.

Le langage est bien décidable car $M_{L'}$ s'arrête toujours : Comme le langage L est infini, il existe m tel que $|w| < |w_m|$ et $\exists k$ tel que M_L reconnait w_m. Ainsi, si le mot w n'est pas reconnu, la machine, une fois arrivé à la m-ieme paire, rejette w et donc s'arrête.

Il faut montrer maintenant que L' est infini.

Supposons par l'absurde L' fini. On a donc que $\exists n, \forall w, |w| \geq n, w \notin L'$. Or on a aussi que $\exists w, |w| > n$ et $w \in L$, car L infini. Mais alors $\exists (w_i, k_i)$ tel que $w = w_i$ et qui est accepté en mois de k_i étapes par M_L, et donc $w \in L'$ ↯. □

Proposition 3.15. *L semi-décidable $\iff \exists L_d$ décidable, $L = \{w \in \Sigma^* \mid \exists w' \in \Sigma^*, \langle w, w' \rangle \in L_d\}$*

Démonstration. On commence par remarque que : $w \in L \iff \exists t, eval(\langle M_L \rangle, w, t) = 1$. On pose $L_d = \{\langle w, k \rangle \mid eval(\langle M_L \rangle, w, k) = 1\}$.

On a bien que L_d est décidable et le sens $\boxed{\Rightarrow}$ découle par construction.

L'autre sense, *i.e.* , montrer que $L = \{w \in \Sigma^* \mid \exists w' \in \Sigma^*, \langle w, w' \rangle \in L_d\}$ est semi-décidable, peut se faire par énumération sur les w', mais elle est laissée en exercice au lecteur. □

3.4 Énumeration des fonctions calculables

Définition 3.16 (Nombre/Codage de Gödel)**.** On peut encoder toute machine de Turing par un nombre, qui est appété le nombre Gödel de et noté : $\langle M \rangle \in \Sigma^*$.

Définition 3.17 (Énumération des fonctions calculables)**.** $\forall n \in \mathbb{N}$, φ_n est la fonction calculée par la n-ième machine de Turing.

$$\varphi_{\langle M \rangle}(w) = M(w)$$

Lemme 3.18 (Machines universelles)**.** *Il existe une machine universelle $\mathcal{U}$, i.e. ,*

$$\forall M, w, \ \mathcal{U}(\langle M, w \rangle) = M(w)$$

ou de manière equivalente

$$\exists u, \forall n, w, \ \varphi_n(w) = \varphi_u(\langle n, w \rangle)$$

3.5 Le problème de l'arrêt et autres problèmes

Les problèmes suivant en rapport aux automates sont décidables :
— ACCEPT$_A$ = $\{< A, w >|$ A est un automate qui accepte $w\}$. Dire que c'est problème est décidable revient a dire que $\exists$ un interpréteur d'automates décidable en Machine de Turing.
— EQUIV$_A$ = $\{< A, A' >|$ A et A' acceptent le meme langage$\}$.
— EXISTS$_A$ = Il existe un mot reconnu.
— INFINITE$_A$ = le langage reconnu par A est infini.

Cependant, ces problèmes étendus aux automates a piles, ne restent pas tous décidable. ACCEPT$_{A_p}$ et INFINITE$_{A_p}$ restent décidables mais pas EQUIV$_{A_p}$.

Définition 3.19 (Problème de l'arrêt)**.** Le problème de l'arrêt est définit comme suit $HALT = \{\langle M, w\rangle \mid M \text{ s'arrête sur } w\}$.

Proposition 3.20. *$HALT$ est semi-décidable.*

Démonstration. Il suffit d'écrire un programme "impératif" :

function $M'(\langle M, w\rangle)$
  $\mathcal{U}(\langle M, w\rangle)$;
  return 1
end function

□

Proposition 3.21. *$HALT$ est indécidable.*

Démonstration. Supposons par l'absurde que HALT est décidable. Alors il existe un entier n tel que φ_n décide HALT, c'est-à-dire

$$\varphi_n(\langle M, w\rangle) = \begin{cases} 1 & \text{si } \varphi_{\langle M\rangle}(w) \neq \perp \\ 0 & \text{sinon} \end{cases}$$

Soit n' le code de la fonction qui sur w vaut 1 si $\varphi_n(w, w) = 0$ et n'est pas définie sinon. Alors

$$\begin{aligned} \varphi_{n'}(n') = 1 \quad & \text{si} \quad \varphi_n(n', n') = 0 \\ & \textit{i.e.} \quad \varphi_{n'}(n') \text{ ne s'arrête pas} \\ & \textit{i.e.} \quad \varphi_{n'}(n') = \perp \end{aligned}$$

$$\begin{aligned} \varphi_{n'}(n') = \perp \quad & \text{si} \quad \varphi_n(n', n') = 1 \\ & \textit{i.e.} \quad \varphi_{n'}(n') \neq \perp \quad \lightning \end{aligned}$$

□

Exemple 3.5.1. Le Problème de Correspondance de Post (PCP) est indécidable. Le détail peut être consulté dans [Sipser(1996), Chapter 5.2]

4 Réductions

4.1 Réduction many-one

Définition 4.1 (Réduction many-one)**.** $A \leq_m B$ si et seulement si $\exists f : \Sigma^* \to \Sigma^*$ calculable et totale telle que

$$\forall w \in \Sigma, w \in A \iff f(w) \in B$$

Théorème 4.2. *$A \leq_m B$ et B est décidable, alors A est décidable*

Démonstration. Soit M_B la machine qui décide B et M_f celle qui calcule f. On pose $M_A(w) = M_B(M_f(w))$.

Montrons que M_A décide A. On a les équivalences suivantes :

$$\begin{aligned} \forall w \in \Sigma^*, w \in A &\iff f(w) \in B \\ &\iff M_B(M_f(w)) = 1 \\ &\iff M_A \text{ accepte } w \end{aligned}$$

et de manière analogue

$$\begin{aligned} \forall w \notin \Sigma^*, w \notin A &\iff f(w) \notin B \\ &\iff M_B(M_f(w)) = 0 \\ &\iff M_A \text{ rejette } w \end{aligned}$$

Et donc M_A décide A.

□

Corollaire 4.3. *$A \leq_m B$ et A n'est pas décidable, alors B n'est l'est pas non plus.*

Démonstration. C'est la contraposée du théorème précédent. □

Théorème 4.4. *$A \leq_m B$ et B est semi-décidable, alors A est semi-décidable*

Démonstration. Voir 4.2 □

Théorème 4.5. *$A \leq_m B$ et B est co-semi-décidable, alors A est co-semi-décidable.*

Démonstration. Si $A \leq_m B$ alors $\overline{A} \leq_m \overline{B}$ car

$$\begin{aligned} (w \notin A &\iff f(w) \notin B) \\ &\iff A \leq_m B \quad (\textit{par contraposée}) \end{aligned}$$

□

Définition 4.6. $A \equiv_m B \iff A \leq_m B$ et $B \leq_m A$

4.2 Classes d'équivalence sous many-one

Définition 4.7 (Many-one complet)**.** L est many-one complet pour une classe de langages $\mathcal{C}$ si $\forall L' \in \mathcal{C}, L' \leq_m L$ et $L \in \mathcal{C}$

Proposition 4.8 (Problème de Post)**.** *HALT est RE-complet.*

Démonstration. Soit $L \in$ RE, montrons que $L \leq_m$ HALT . Soit M_L la machine qui semi-décide L.
Alors on construit M'_L la machine suivante :

function $M'_L(w)$
  if $M_L(w) = 1$ **then** 1
  else $\perp$
  end if
end function

$$\begin{aligned} w \in L &\iff M_L = 1 \\ &\iff M'_L \neq \perp \\ &\iff M'_L \text{ s'arrête sur } w \\ &\iff \langle M'_L, w\rangle \in \text{ HALT} \end{aligned}$$

Et donc, on pose $f(w) = \langle M'_L, w\rangle$ et on a bien que $w \in L \iff f(w) \in$ HALT et donc HALT est RE-complet. □

Exercice 4.2.1. Construire, à partir de HALT, un langage qui n'est ni RE ni co-RE.

Démonstration. Considérons

$$L = \{\langle M, M', w\rangle \mid M(w) = \perp \text{ et } M'(w) \neq \perp\}$$

Comme HALT et $\overline{\text{HALT}}$ sont RE-complet et co-RE-complet respectivement, il suffit de montrer HALT $\leq_m L$ et $\overline{\text{HALT}} \leq_m L$.

— $\langle M, w\rangle \in$ HALT $\iff \langle M_{loop}, M, w\rangle \in L$. Donc HALT $\leq_m L$ (où M_{loop} est une machine qui boucle toujours).

— $\langle M, w\rangle \in \overline{\text{HALT}} \iff \langle M, M_1, w\rangle \in L$. Donc $\overline{\text{HALT}} \leq_m L$ (où M_1 est une machine qui s'arrête toujours).

On a donc que L n'est ni RE ni co-RE. □

Dans les deux exercices suivants, on pourra utiliser (implicitement ou explicitement) le fait que la composition de machines de Turing est *uniformément calculable.*

Lemme 4.9. *Il existe une fonction calculable* comp *telle que :*

$$\forall cc'w, \varphi_{\text{c, c'}}(w) = \varphi_c(\varphi_{c'}(w))$$

Exercice 4.2.2. Montrer que l'ensemble des programmes reconnaissant le langage vide est indécidable.

Démonstration. Il suffit de montrer que $\overline{\text{HALT}} \leq_m L_\emptyset = \{\langle M\rangle \mid \forall w \in \Sigma^*, M(w) \neq 1\}$.

Notons f_1 la fonction calculable qui sur $\langle M\rangle$ retourne le code de l'une machine M' définie par :

function $M'(w)$
  $M(w)$;
  return 1
end function

Pour une preuve plus formelle de la calculabilité de cette fonction, on pourra utiliser le lemme 4.9 ainsi que le théorème Smn (i.e. théorème 5.1).

Alors on a que

$$\begin{aligned} \langle M, w\rangle \in \overline{\text{HALT}} &\iff M(w) = \bot \\ &\iff \forall w, \varphi_{f_1(\langle M\rangle)}(w) \neq 1 \\ &\iff f_1(\langle M\rangle) \in L_\emptyset \end{aligned}$$

et donc $\overline{\text{HALT}} \leq_m L_\emptyset$.

□

Exercice 4.2.3. Montrer que l'équivalence de machines de Turing n'est ni RE ni co-RE.

Démonstration. Il faut montrer que le langage $EQUIV = \{\langle M, M'\rangle \mid \forall w \in \Sigma^*, M(w) = M'(w)\}$ vérifie $\text{HALT} \leq_m EQUIV$ et $\overline{\text{HALT}} \leq_m EQUIV$.

Soit M une machine de Turing et w un mot :

— On pose $M' =$ **fun** $\_ \to M(w)$; return 1.
  $\langle M, w\rangle \in \text{HALT} \iff \langle M_1, M', w\rangle \in EQUIV$.
  La transformation $\langle M\rangle \mapsto \langle M_1, M', w\rangle$ est bien calculable, donc $\text{HALT} \leq_m EQUIV$.

— On pose $M' =$ **fun** $\_ \to M(w)$.
  $\langle M, w\rangle \in \overline{\text{HALT}} \iff \langle M_{loop}, M', w\rangle \in L$.
  La transformation $\langle M\rangle \mapsto \langle M_{loop}, M', w\rangle$ est bien calculable, donc $\overline{\text{HALT}} \leq_m EQUIV$.

On a donc, d'après le théorème 4.4 que $EQUIV$ n'est ni RE ni co-RE. □

5 Théorèmes de récursion

L'application partielle d'une fonction calculable (ou plutôt de sa currification) est une fonction calculable. Elle est en fait *calculable uniformément*, i.e. on peut calculer le code de l'application partielle en fonction du code de la fonction.

Théorème 5.1 (d'itération / Smn / d'application partielle)**.** *Il existe une fonction calculable et totale s telle que*

$$\forall n, m, w, \varphi_{s(m,n)}(w) = \varphi_m(\langle n, w\rangle)$$

Si m est le code d'un programme et n un mot, alors $s(m, n)$ est le code de l'application partielle de φ_m à n.

Théorème 5.2 (de point fixe)**.** *Si $f : \Sigma^* \to \Sigma^*$* calculable *et* totale*. Alors il existe $e \in \Sigma^*$ tel que $\varphi_e = \varphi_{f(e)}$.*

Démonstration. Soit G la machine : $(x, y) \to$ **let** $e = \mathcal{U}(\langle x, x\rangle)$ **in** $\mathcal{U}(\langle e, y\rangle)$.

On pose $h(x) = s(\langle G\rangle, x)$ (le s du théorème precedent). On a que $f \circ h$ est calculable et totale et notons son code c. Alors

$$\begin{aligned}
\varphi_{h(c)}(w) &= \varphi_{s(\langle G\rangle,c)}(w) \quad (\textit{par définition de } h)\\
&= \varphi_{\langle G\rangle}(\langle c,w\rangle) \quad (\textit{par } 5.1)\\
&= G(\langle c,w\rangle) \quad (\textit{correspondence énumeration machine })\\
&= \textbf{let } e = \mathcal{U}(\langle c,c\rangle) \textbf{ in } \mathcal{U}(\langle e,w\rangle) \quad (\textit{par définition de } G)\\
&= \varphi_{f\circ h(c)}(w) \quad (\textit{car } \mathcal{U}(\langle c,c\rangle) =_{3.18} \varphi_c(c) =_{3.17} f\circ h(c))
\end{aligned}$$

Donc $\varphi_{h(c)} = \varphi_{f\circ h(c)}$ et donc $h(c)$ est notre point fixe. □

Théorème 5.3 (de récursion)**.** *Si $f : \Sigma^* \times \Sigma^* \to \Sigma^*$ est une fonction partielle et calculable. Alors il existe une machine R qui calcule $r : \Sigma^* \to \Sigma^*$ tel que*

$$\forall w, r(w) = f(\langle R\rangle, w)$$

Autrement dit, $\exists e, \varphi_e(w) = f(e,w)$

Démonstration. Soit M_f la machine qui calcule f et $g : \Sigma^* \to \Sigma^*$, $g(p) = s(\langle M_f\rangle, p)$ Alors on applique le théorème de point fixe à g et on a qu'il existe r tel que $\varphi_r(w) = \varphi_{s(\langle M_f\rangle,r)}(w) = \varphi_{\langle M_f\rangle}(r,w) = f(r,w)$ □

Théorème 5.4 (de Rice)**.** *Toute propriété non triviale relative au langage reconnu par une machine de Turing est indécidable.*

Autrement dit, soit $L = \{\langle M\rangle \mid P(L_M)\}$, avec P une propriété non triviale, i.e. $\exists M_1$ tel que $\langle M_1\rangle \in L$ et $\exists M_2$ tel que $\langle M_2\rangle \notin L$. Alors L n'est pas décidable.

Démonstration. Soit $L = \{\langle M\rangle \mid P(L_M)\}$. Sans perte de généralité, supposons $P(\emptyset)$, *i.e.* $\langle M_{loop}\rangle \in L$. On peut faire ceci car, dans le cas ou $\neg P(\emptyset)$, alors on considère $\bar{L}$.

Alors $\langle M_{loop}\rangle \in L$ et $\exists \langle M_2\rangle \notin L$, car P est non triviale.

Montrons que $\overline{\text{HALT}} \leq_m L$. Soit M une machine de Turing, pour tout w on pose

$$M' = \textbf{fun } u \to M(w); M_2(u)$$

On commence par remarque que, comme $M_{loop} \in L$ et $M_2 \notin L$ alors $\mathcal{L}(M_2) \neq \emptyset$. Ainsi, $\exists w', M_2(w') \neq \bot$. Alors

$$\begin{aligned}
\langle M,w\rangle \in \overline{\text{HALT}} &\iff M(w) = \bot\\
&\iff \mathcal{L}(M') = \emptyset\\
&\iff M_L(M') = 1\\
&\iff \langle M'\rangle \in L
\end{aligned}$$

Donc $\overline{\text{HALT}} \leq_m L$ et ainsi L n'est pas décidable. □

Proposition 5.5. *Il existe un Quine, i.e. une machine M tel que :*

$$\forall w, M(w) = \langle M\rangle$$

Démonstration. On applique le théorème de récursion avec la fonction $f(e, \_) = e$. Donc $\exists R$, $R(w) = f(\langle R\rangle, w) = \langle R\rangle$. □

6 Machines de Turing à oracle

Nous allons présenter une extension des Machines de Turing qui nous sera utile pour la section suivante.

Définition 6.1 (Machine de Turing à oracle)**.** Une machine à oracle est une machine de Turing qui a acces à une fonction $\mathscr{O} : \Sigma^* \to \Sigma^*$ pendant son exécution. Elle a donc, en plus de la machine initiale, un ruban d'appel (pour l'oracle) et un état spécial d'appel.

Si la machine rentre dans l'état d'appel avec $u \in \Sigma^*$ sur le ruban, alors $\mathscr{O}(u) \in \Sigma^*$ est écrit sur le ruban d'appel.

Définition 6.2 (Reconnaissance)**.** On dit que M reconnait L relativement à un oracle A si

$$\forall w, M^A(w) = 1 \iff w \in L$$

où M^A est l'exécution de M avec l'oracle A.

Définition 6.3 (Reduction de Turing)**.** On dit que $A \leq_T B \iff A$ est décidable relativement à B.

Remarque 6.0.1. *On a que $A \leq_m B \implies A \leq_T B$. En effet, il suffit de construire la machine à oracle qui sur w appelle l'oracle sur $f(w)$, où f est la fonction de reduction.*

Remarque 6.0.2. *Pour tout langage L, on a que $L \leq_T \bar{L}$ et $\bar{L} \leq_T L$ et donc $L \equiv_T \bar{L}$.*

Lemme 6.4. *Si A est C-complet et $Y \in C$, alors :*

$$X \text{ est } re(Y) \implies X \text{ est } re(A)$$

Démonstration. Par définition, X est $re(Y)$ s'il existe une machine à oracle M telle que X est reconnu par M^Y. Comme A est C-complet et $Y \in C$, il existe une réduction calculable f de Y vers A. On pose N la machine à oracle construite à partir de M et qui applique f au ruban d'appel, avan chaque appel à l'oracle. Autrement dit, pour tout oracle $\mathscr{O}$ et toute entrée w, $N^{\mathscr{O}}(w) = M^{\mathscr{O} \circ f}$. On a donc que N^A reconnaît X. □

Définition 6.5 (eval avec oracle)**.** $eval(\langle M \rangle, \sigma, w, t)$ évalue $M^\sigma(w)$ en un nombre d'étapes inférieur à t.

Notation 6.6. $\varphi_{c \in \Sigma^*}$: énumeration des fonctions calculables.

$\varphi^X_{c \in \Sigma^*}$: énumeration des fonctions calculables relativement à X. Et donc $\varphi^X_{\langle M \rangle}(w) = M^X(w)$

7 Hiérarchie arithmétique

7.1 Les ensembles Σ_n, Π_n et Δ_n

Dans cette section, nous allons étudier les $\underbrace{\text{formules}}_{\wedge,\vee,\neg}$ $\underbrace{\text{arithmétiques}}_{\mathbb{N},+,*,\leq,\mathrm{S}}$ du $\underbrace{\text{premier ordre}}_{\exists x, \forall x}$ sous forme prénexe et voir comment les classifier.

Définition 7.1 (Forme prénexe)**.** Une formule est sous forme prénexe si tous ses quantificateurs non-bornés sont en tête (i.e. "à gauche").

$(\forall x(x \leq 2)) \wedge (\forall y(2 \leq y))$ n'est pas sous forme prénexe mais $(\forall x \forall y((x \leq 2) \wedge (2 \leq y))$ l'est.

Définition 7.2. On définit les ensembles :

$$\begin{aligned} \Sigma_{n+1} &= \{\exists x \psi \mid \psi \in \Pi_n\} \\ \Pi_{n+1} &= \{\forall x \psi \mid \psi \in \Sigma_n\} \\ \Delta_{n+1} &= \Sigma_{n+1} \cap \Pi_{n+1} \\ \Delta_0 &= \Pi_0 = \Sigma_0 \quad (\textit{formules sans quantificateurs non-bornés}) \end{aligned}$$

Autrement dit, Σ_n est l'ensemble des formules avec n quantificateurs alternés, qui commencent par un quantificateur existentiel.

Remarque 7.1.1. *Dans la définition précédente, les formules de la forme $\forall x \forall y \ldots$ ne sont pas prises en compte. Ceci est dû au fait que, grâce au lemme suivant, les quantificateurs égaux qui se suivent peuvent être regroupés en un seul.*

Lemme 7.3. *(Admis) Pour tout φ il existe φ_1 et φ_2 tels que :*

$$\forall x_1, \forall x_2, \varphi(x_1, x_2) \iff \forall x, \varphi(\varphi_1(x), \varphi_2(x))$$

Le même résultat peut être obtenu pour le quantificateur existentiel. Cette propriété découle du fait que $\mathbb{N} \cong \mathbb{N} \times \mathbb{N}$ et qu'il existe un tel encodage des paires d'entiers qui peut être décodé via une formule Δ_0. On pourra trouver quelques exemples de fonctions de couplage comme celle de Cantor ici : https://en.wikipedia.org/wiki/Pairing_function .

Proposition 7.4. *Les langages de Δ_0 sont décidables.*

Démonstration. Soit $\varphi \in \Delta_0$, alors, comme elle ne contient pas de quantificateurs non-bornés, on peut essayer toutes les valeurs possibles, car il y en a un nombre fini. □

7.1.1 Quelques exemples

Exercice 7.1.1. Montrer que $ACCEPT = \{\langle M, w \rangle \mid M(w) = 1\} \in \Sigma_1$

Démonstration.

$$\varphi(M, w) = \exists n, \underbrace{eval(\langle M \rangle, w, n) = 1}_{\in \Delta_0} \in \Sigma_1$$

Ainsi, φ décrit $ACCEPT$ et $\varphi \in \Sigma_1$ et donc $ACCEPT \in \Sigma_1$. □

Exercice 7.1.2. Montrer que $ACCEPT$ est Σ_1-complet, *i.e.*

$$\forall L \in \Sigma_1, L \leq_m ACCEPT$$

Démonstration. Soit $L \in \Sigma_1$. Alors il existe $\varphi \in \Delta_0$ tel que

$$L = \{n \mid \exists x \varphi(x, n)\}$$

Soit $M(n)$ la machine qui énumère tous les x et accepte si $\varphi(x, n)$ est satisfaite. Alors M reconnait L. □

Proposition 7.5. $RE = \Sigma_1$

Démonstration. D'après la proposition 3.15 :

$$L \text{ est r.e.} \Rightarrow \exists L_d \text{ décidable }, L = \left\{ w \mid \overbrace{\exists w', \underbrace{\langle w, w' \rangle \in L_d}_{\in \Delta_0}}^{\in \Sigma_1} \right\}$$

Inversement,

$$L \in \Sigma_1 \iff L = \{w \mid \exists w', \varphi(w, w')\} \text{ avec } \varphi \in \Delta_0$$

Semi-décider $w \in L$ revient alors à énumérer les w' jusqu'à trouver en trouver un tel que $\varphi(w, w')$, que l'on peut décider d'après la proposition 7.4. Et donc $RE = \Sigma_1$ □

7.2 Hiérarchie arithmétique

Dans cette section, nous examinons les relations entre ces ensembles afin de les classifier, ce qui conduit à la hiérarchie illustrée dans la Figure 2. On démontrera que ces ensembles forment une suite croissante (7.6), que chaque ensemble contient un élément complet pour cette classe (7.2.2)

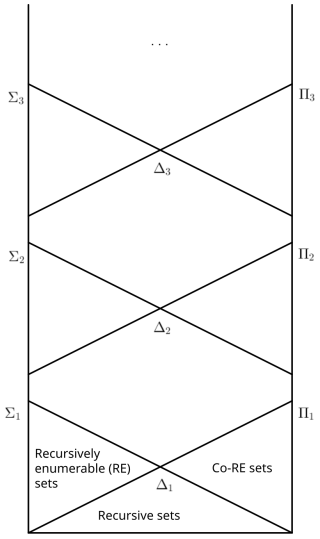


FIGURE 2 – Hiérarchie arithmétique

Théorème 7.6 (de Post)**.** *Nous avons les résultats suivants :*

1. (a)

$$\begin{aligned} L \in \Sigma_{n+1} &\iff L \textit{ est r.e. relativement à un langage } \Pi_n \\ &\iff L \textit{ est r.e. relativement à un langage } \Sigma_n \end{aligned}$$

(b)

$$\begin{aligned} L \in \Pi_{n+1} &\iff L \textit{ est co-r.e. relativement à un langage } \Sigma_n \\ &\iff L \textit{ est co-r.e. relativement à un langage } \Pi_n \end{aligned}$$

2. Il existe un langage Σ_n-complet, noté $\emptyset^{(n)}$.

Démonstration du point 1a $\boxed{\Leftarrow}$.
L est calculable relativement à $A \in \Pi_n$ via la machine M_L.

$$\begin{aligned} w \in L &\iff \exists t, M^A \text{ accepte } w \text{ en un temps inférieur à } t \\ &\iff \exists t, \exists \sigma, \underbrace{\sigma \subseteq A}_{\forall u,v,(u,v)\in\sigma \to (v=1 \iff \varphi_A(u))} \wedge M^\sigma(w) \text{ accepte en temps inférieur à } t \end{aligned}$$

Ainsi, si $A \in \Pi_n$ alors $L \in \Sigma_{n+1}$. Sinon il suffit d'écrire $v = 0 \iff \underbrace{\neg\varphi_A(u)}_{\in\Pi_n}$ et on a aussi que $L \in \Sigma_{n+1}$.

Les autres cas du point 1 se démontrent de manière similaire. □

7.2.1 Σ_n complétude

Définition 7.7 (Saut de Turing)**.** Soit X un langage,
$X' = \{c \mid \varphi_c^X(c) \text{ est défini}\} = \{\langle M\rangle \mid M^X \text{ s'arrête sur } \langle M\rangle\}$

Définition 7.8.

$$\emptyset' = \left\{\langle M\rangle \mid M^\emptyset \text{ s'arrête sur } \langle M\rangle\right\}$$

$$\emptyset^{(n+1)} = \left\{\langle M\rangle \mid M^{\emptyset^{(n)}} \text{ s'arrête sur } \langle M\rangle\right\}$$

Remarque 7.2.1.

$$\emptyset' \equiv_m \mathit{HALT}$$

Démonstration.

—

$$\begin{aligned} \langle M\rangle \in \emptyset' &\iff M^\emptyset(\langle M\rangle) \neq \bot \\ &\iff \langle M, \langle M\rangle\rangle \in \text{ HALT} \end{aligned}$$

Et donc $\emptyset' \leq_m$ HALT .

—

$$\begin{aligned} \langle M, w\rangle \in \text{ HALT} &\iff M(w) \neq \bot \\ &\iff \langle \textbf{fun } \_ \to M(w)\rangle \in \emptyset' \end{aligned}$$

Et donc HALT $\leq_m \emptyset'$.

□

Exercice 7.2.1. Montrer que $\emptyset'$ est Σ_1-complet :

1. $\emptyset' \in \Sigma_1$
2. $\forall L \in \Sigma_1, L \leq_m \emptyset'$

Démonstration.

1. $\langle M \rangle \in \emptyset' \iff \exists t, eval(\langle M \rangle, ((\_,0),\ldots,(\_,0)), \langle M \rangle, t) \neq \perp$
2. On a que $\emptyset' \equiv_m$ HALT qui est Σ_1-complet, donc $\emptyset'$ l'est aussi.

□

Lemme 7.9. *$\forall n$, $\emptyset^{(n)}$ est Σ_n-complet.*

Remarque 7.2.2. *Ce lemme montre en particulier le point 2 du théorème de Post.*

Démonstration. La démonstration est faite par induction. Les cas $n = 0$ et $n = 1$ on été traités précédemment, il suffit alors de montrer le cas d'induction.

Supposons $\emptyset^{(n)}$ Σ_n-complet.

$$\begin{aligned} L \in \Sigma_{n+1} &\iff \exists A \in \Sigma_n, L \text{ est r.e. relativement à } A \\ &\iff L \text{ est re}(\emptyset^{(n)}) \\ &\iff \exists M_L, M_L^{\emptyset^{(n)}} \text{ reconnait } L \\ &\overset{?}{\iff} L \leq_m \emptyset^{(n+1)} \end{aligned}$$

— $\boxed{\Rightarrow}$

$$\begin{aligned} w \in L &\iff M_L^{\emptyset^{(n)}}(w) \text{ s'arrête et accepte} \\ &\iff M_{L,w}^{\emptyset^{(n)}}(\cdot) \text{ s'arrête} \\ &\iff \langle M_{L,w} \rangle \in \emptyset^{(n+1)} \\ &\iff L \leq_m \emptyset^{(n+1)} \end{aligned}$$

— $\boxed{\Leftarrow}$

On a que $L \leq_m \emptyset^{(n+1)}$. Soit f la fonction de réduction, alors on pose $M_L = M_{\emptyset^{(n+1)} \circ f}$. Cette machine vérifie bien que $M_L^{\emptyset^{(n)}}$ reconnait L.

□

7.2.2 La hiérarchie ne s'effondre pas

On s'intéresse maintenant à savoir si la hiérarchie est stricte, *i.e.* , $\exists \varphi \in \Sigma_{n+1}$, tel que , $\varphi \notin \Sigma_n$ et $\varphi \notin \Pi_n$.

Proposition 7.10.

$$\forall n \in \mathbb{N}^*, \emptyset^{(n)} \notin \Delta_n$$

Remarque 7.2.3. *Pour $n > 0$*

$$\begin{aligned} L \in \Delta_n &\iff L \in \Sigma_n \wedge L \in \Pi_n \\ &\iff L \text{ est } re(\Sigma_{n-1}) \wedge L \text{ est } co-re(\Sigma_{n-1}) \\ &\iff L \text{ est re relativement à } \emptyset^{(n-1)} \wedge L \text{ est co}-re \text{ relativement à } \emptyset^{(n-1)} \\ &\iff L \text{ est décidable relativement à } \emptyset^{(n-1)} \end{aligned}$$

Démonstration. On peut utiliser la remarque précédente. Ainsi, il suffit de montrer que $\emptyset^{(n)}$ n'est pas décidable relativement à $\emptyset^{(n-1)}$.

Supposons par l'absurde que $\emptyset^{(n)}$ est décidable relativement à $\emptyset^{(n-1)}$, *i.e.* , il existe M_n telle que $M_n^{\emptyset^{(n-1)}}$ décide $\emptyset^{(n)}$.

Alors on construit

$$M_0(\langle M\rangle) = \begin{cases} \text{Si } M_n^A(\langle M\rangle) \text{ accepte} & \to \text{ boucle} \\ \text{Si } M_n^A(\langle M\rangle) \text{ rejecte} & \to \text{ accepte} \end{cases}$$

Alors on a que

$$\begin{aligned} M_0^{\emptyset^{(n-1)}}(\langle M_0\rangle) \text{ boucle} &\iff M_n^{\emptyset^{(n-1)}}(\langle M_0\rangle) \text{ accepte} \\ &\iff M_0^{\emptyset^{(n-1)}}(\langle M_0\rangle) \text{ s'arrête } \lightning \end{aligned}$$

□

Corollaire 7.11. *La hiérarchie ne s'effondre pas.*

Références

[Matiyasevich(1993)] Yuri V. Matiyasevich. Hilbert's tenth problem. *American Mathematical Monthly*, 102 :366, 1993. URL https://api.semanticscholar.org/CorpusID:116671943.

[Sipser(1996)] Michael Sipser. *Introduction to the Theory of Computation*. International Thomson Publishing, 3rd edition, 1996. ISBN 053494728X.