Calculabilité

II

Kévin Perrot – Aix Marseille Université – printemps 2016

Table des matières

3 Les limites du calcul

3.1 Enumération des machines de Turing

Remarque 14.
Enumérer *un ensemble* S $=$ *donner (au moins) un numéro à chaque élément*
$=$ *donner une fonction surjective de* $\mathbb{N}$ *dans* S.
Dans ce cas S *ne peut pas être plus grand que* $\mathbb{N}$.
Une énumération[1] *de* S *sans répétition (= injective) est une bijection de* $\mathbb{N}$ *dans* S.

Remarque 15. *Il y a*

$$((nm2)+1)^{(n-1)m}$$

machines de Turing à $|Q| = n$ *états et* $|\Gamma| = m$ *symboles. Il faut au moins 2 états (*q_0 *et* q_F*) et au moins 2 symboles de ruban (*B *et un autre) pour définir une MT.*

Remarque 16. *On peut écrire la définition d'une machine de Turing sur une feuille découpée en cases (ça ressemble beaucoup à un ruban). En prenant un ordre sur les symboles utilisés (ça ressemble beaucoup à un ordre sur* Γ*), on peut définir un ordre lexicographique*[2] *sur l'ensemble des machines de Turing. Donc étant donné un ensemble fini quelconque de machines de Turing, on peut les énumérer.*

Lemme 17. *Il existe une bijection entre l'ensemble des machines de Turing et* $\mathbb{N}$.

Donc il existe une **énumération des machines de Turing** (et même plusieurs).

1. Il faut que l'énumération soit totale, c'est-à-dire que tout élément ait une image.
2. Comme dans un dictionnaire.

Démonstration. Nous allons énumérer sans répétition l'ensemble des machines de Turing. L'idée est de numéroter de 0 à 10 les 11 MT à 2 états et 2 symboles, puis de 11 à 24 les 14 MT à 2 états et 3 symboles, puis de 25 à 33 les 9 MT à 3 états et 2 symboles, puis de 34 à 54 les 21 MT à 3 états et 3 symboles, etc.

Plus formellement, soit f une bijection de $\mathbb{N}$ dans $(\mathbb{N} \setminus \{0,1,2\})^2$. Nous allons commencer par énumérer les MT avec $(|Q|,|\Gamma|) = f(0)$, puis les MT avec $(|Q|,|\Gamma|) = f(1)$, etc. Pour un $(|Q|,|\Gamma|)$ donné il y a un nombre fini de machines de Turing (remarque 15), nous pouvons donc les énumérer sans problème (remarque 16). □

3.2 Simplifications

Lorsque cela nous arrangera, nous pourrons nous ramener à ne considérer
— que **les langages sur** $\Sigma = \{0,1\}$ (au lieu de Σ fini quelconque),
— que **les fonctions de** $\mathbb{N}$ **dans** $\mathbb{N}$ (au lieu des fonctions de Σ^* dans Γ^*).

Remarque 18. *Quand on dit qu'une chose A « peut se ramener à » une chose B, on dit en fait plus formellement que l'ensemble des choses A peut être mis en* **bijection** *avec l'ensemble des choses B.*

Pour un Σ donné, avec une bijection de Σ^* dans $\mathbb{N}$ (par exemple $f(w_0w_1\dots w_k) = \sum_{i=0}^{k} w_i m^i + m^{k+1} - 1$ avec $m = |\Sigma|$ et la convention $f(\epsilon) = 0$) on peut traduire un mot en un nombre (et réciproquement car c'est une bijection). En prenant la représentation binaire de ce nombre, nous obtenons une bijection de Σ^* dans $\{0,1\}^*$.

Avec une bijection de Σ^* dans $\mathbb{N}$ et une bijection de Γ^* dans $\mathbb{N}$, toute fonction de Σ^* dans Γ^* calculée par une MT peut se ramener à une fonction de $\mathbb{N}$ dans $\mathbb{N}$.

3.3 Dénombrement

Notation 19. *L'***ensemble des parties de** $\mathbb{N}$ *(ensemble des sous-ensembles de* $\mathbb{N}$*) est dénoté* $\mathcal{P}(\mathbb{N})$ *ou* $2^{\mathbb{N}}$*, et est en bijection avec* $\mathbb{R}$*, qui est en bijection avec* $[0,1[$.

Notation 20. $|\mathbb{N}| = |\mathbb{Q}| = |\mathbb{Z}| = |\mathbb{N}\times\mathbb{N}| = \aleph_0$ *et* $|\mathbb{R}| = |\mathbb{R}\times\mathbb{R}| = |[0,1[| = 2^{\aleph_0}$.

Lemme 21. *Il existe une bijection entre l'ensemble des langages sur* $\Sigma = \{0,1\}$ *et* $[0,1[$.

Démonstration. L'ensemble des mots sur $\Sigma = \{0,1\}$ est en bijection avec $\mathbb{N}$ (représentation binaire). Un langage peut donc se ramener à un sous-ensemble de $\mathbb{N}$ (un ensemble de nombres). Par conséquent l'ensemble des langages sur $\{0,1\}$ est en bijection avec $\mathcal{P}(\mathbb{N})$, qui est en bijection avec $[0,1[$ (notation 19). □

Lemme 22. *Il existe une bijection entre l'ensemble des fonctions de* $\mathbb{N}$ *dans* $\mathbb{N}$*, et* $[0,1[$.

Démonstration. Il y a autant de fonctions de A dans B que d'éléments dans $B^{|A|}$. Dans notre cas il y a $\aleph_0^{\aleph_0}$ fonctions de $\mathbb{N}$ dans $\mathbb{N}$. Montrons que ce nombre n'est pas plus grand que $2^{\aleph_0}$, ce que nous admettrons comme suffisant pour conclure.

Pour cela nous allons construire une fonction injective de l'ensemble des fonctions de $\mathbb{N}$ dans $\mathbb{N}$, dans $[0,1[$. Une fonction f de $\mathbb{N}$ dans $\mathbb{N}$ est une suite infinie de nombres : $f(0), f(1), f(2)$, etc (attention : chaque $f(i)$ est un nombre fini car $+\infty \notin \mathbb{N}$). Nous pouvons donc faire correspondre à chaque fonction un nombre réel $0.f(0)f(1)f(2)\dots$. Cette fonction n'est cependant pas injective (ni surjective). Pour la rendre injective nous pouvons utiliser le codage suivant :

— les $f(i)$ sont codés en binaire dédoublé (chaque bit est écrit deux fois, par exemple 9 en décimal devient 11000011),
— les $f(i)$ et $f(i+1)$ sont séparés par la séquence 01.

□

Théorème 23. $\aleph_0 < 2^{\aleph_0}$.

Démonstration. Nous allons démontrer ce résultat par l'absurde. Supposons qu'il existe une bijection entre $\mathbb{N}$ et $[0,1[$ (auquel cas $\aleph_0 = 2^{\aleph_0}$), alors il est possible d'énumérer tous les nombres réels entre 0 (inclus) et 1 (exclu) sans en oublier aucun :

$$\begin{aligned} r_1 &= 0.\boxed{1}234567890\ldots \\ r_2 &= 0.5\boxed{3}49236423\ldots \\ r_3 &= 0.72\boxed{9}1655000\ldots \\ r_4 &= 0.239\boxed{3}218693\ldots \\ &\ldots \end{aligned}$$

Nous allons montrer qu'il est impossible d'avoir énuméré tous les éléments de $[0,1[$. En effet, nous avons forcément oublié le nombre suivant :

$$r_+ = 0.2404\ldots$$

construit en prenant pour première décimale la première décimale de r_1 plus 1 modulo 10, en seconde décimale la seconde décimale de r_2 plus 1 modulo 10, en troisième décimale la troisième décimale de r_3 plus 1 modulo 10, etc, à l'infini (les nombres réels peuvent avoir une infinité de décimales). On a bien $r_+ \in [0,1[$, et pour tout $i \in \mathbb{N} : r_i \neq r_+$ car ils diffèrent par la $i^{\text{ème}}$ décimale. (Diagonale de Cantor, 1891) □

Corollaire 24.
Il existe des langages non récursifs et des fonctions non calculables.

Démonstration. Application du théorème 23 d'après les lemmes 17 et 21 pour les langages, et lemmes 17 et 22 pour les fonctions. □

Remarque 25. *Il existe donc des infinis de tailles différentes. On dira*
— **infini dénombrable** *s'ils sont en bijection avec* $\mathbb{N}$ *(donc de taille* $\aleph_0$*),*
— **infini indénombrable** *sinon.*

Le théorème 24 (corollaire du théorème 23) ne nous donne pas d'exemple de langage non récursif / fonction non calculable, mais nous dit qu'ils / elles sont très nombreux / nombreuses (infiniment plus que les récursifs / calculables).

Le théorème 23 amène une question naturelle : existe-t-il des infinis strictement plus grands que $\aleph_0$, mais strictement plus petits que $2^{\aleph_0}$? En d'autres termes, si l'on dénote $\aleph_1$ le second plus petit infini après $\aleph_0$, est-ce que

$$2^{\aleph_0} = \aleph_1 \,?$$

Cette question fameuse est appelée **hypothèse du continu** (HC), et fut posée par Cantor autour de 1878. Il fallut attendre l'axiomatisation de la théorie des ensembles [3]

3. C'est-à-dire la définition précise d'axiomes à partir desquels on dérive des théorèmes (vérités mathématiques). Avant cela (et pour Cantor notamment), on utilisait une définition intuitive (« naïve ») des ensembles. Par exemple, rien n'interdisait de considérer l'ensemble de tous les ensembles, $\mathcal{S}$. Russell souleva a ce propos un paradoxe divertissant : soit $X = \{A \in \mathcal{S} \mid A \notin A\}$, est-ce que $X \in X$?

par Zermelo et Fraenkel (ZF) au début du XX^{e} siècle, une preuve par Gödel en 1938 que HC ne peut pas être réfutée dans ZF, et une preuve par Cohen en 1963 que HC ne peut pas être prouvée dans ZF, pour arriver à la conclusion suivante : HC est indépendante de ZF. Reformulé, la théorie des ensembles qui fait consensus en mathématique ne permet pas de dire si l'hypothèse du continu est vraie ou fausse, les deux éventualités sont consistantes.

3.4 Code d'une machine de Turing

Les résultats fondamentaux sur les limites du calcul sont liés à des problèmes dans lesquels une machine de Turing doit répondre à une question sur les machines de Turing. Pour cela, il faut pouvoir donner en entrée à une machine de Turing la définition (le code, le programme) d'une autre machine de Turing. Deux possibilités :

— en donnant le numéro de la machine dans une énumération des machines de Turing,

— en écrivant le code de la machine sur la ruban.

Nous avons vu dans la section 3.1 comment énumérer les machines de Turing, voyons maintenant comment les encoder sur le ruban.

Notation 26. *Nous noterons* $\langle M \rangle$ *le* **code d'une machine de Turing**.

Il y a de nombreuses façons d'encoder les machines de Turing sur le ruban. Par exemple, en numérotant de q_1 à q_n les états et de a_1 à a_m les symboles de ruban utilisés par une machine M, et en fixant $D = 0$ et $L = 00$, il est possible d'encoder chaque transition $\delta(q_i, a_j) = (q_k, a_l, D)$ de M par la séquence

$$\text{transition} = \underbrace{0\ldots0}_{i}1\underbrace{0\ldots0}_{j}1\underbrace{0\ldots0}_{k}1\underbrace{0\ldots0}_{l}10$$

On peut alors encoder une machine complète en commençant par dire combien elle a d'états, combien elle a de symboles de ruban, puis en listant les x transitions une à une :

$$\langle M \rangle = 111\underbrace{0\ldots0}_{n}11\underbrace{0\ldots0}_{m}11\text{transition}_1 11\text{transition}_2 11\ldots 11\text{transition}_x 111.$$

Par convention, nous pouvons énumérer les transitions dans l'ordre lexicographique selon l'état et le symbole.

On se convaincra que le résultat suivant est vrai.

Lemme 27. *Le langage* $L_{enc} = \{w \in \{0,1\} \mid w = \langle M \rangle$ *pour une MT* $M\}$ *est récursif.*

3.5 Théorème de l'arrêt

Théorème 28. *La fonction* **halt** : $(\langle M \rangle, w) \mapsto \begin{cases} 0 \text{ si } M(w) \uparrow \\ 1 \text{ sinon} \end{cases}$ *n'est pas calculable.*

Démonstration. Par l'absurde, supposons qu'il existe une machine de Turing M_{halt} qui calcule la fonction *halt*. Nous pouvons alors sans difficulté construire la machine M_{diag} suivante :

$$M_{diag}(i) = \begin{cases} 1 \text{ si } M_{halt}(\langle i \rangle, \langle i \rangle) = 0 \\ \uparrow \text{ si } M_{halt}(\langle i \rangle, \langle i \rangle) = 1 \end{cases}$$

où $\uparrow$ signifie que M_{diag} entre dans une boucle infinie (et ne termine donc pas). Considérons à présent l'entrée $\langle diag \rangle$ donnée à la machine M_{diag}. Deux cas sont possibles.

— Si $M_{diag}(\langle diag \rangle) = 1$ alors, par définition de M_{diag}, nous avons $M_{halt}(\langle diag \rangle, \langle diag \rangle) = 0$ ce qui signifie, par définition de M_{halt}, que $M_{diag}(\langle diag \rangle) \uparrow$, une contradiction.
— Si $M_{diag}(\langle diag \rangle) \uparrow$ alors, par définition de M_{diag}, nous avons $M_{halt}(\langle diag \rangle, \langle diag \rangle) = 1$ ce qui signifie, par définition de M_{halt}, que $M_{diag}(\langle diag \rangle)$ s'arrête, une contradiction.

Dans les deux cas nous arrivons à une contradiction. □

3.6 Raisonnement par l'absurde et diagonalisation

N'est-il pas surprenant que des résultats si révolutionnaires admettent des preuves si courtes et simples ? On peut noter le caractère diagonal (auto-référant) des preuves de Cantor (1891) et Turing (1936). C'est également sur un argument diagonal qu'est basé le premier théorème d'incomplétude de Gödel (1931) !

3.7 Réductions

Une des formulations les plus populaires du résultat de Turing en 1936 est donnée par le théorème 28. Voyons maintenant des développements mathématiquement plus « épurés » qui mènent au même résultat, mais nous permettront d'aller plus loin de façon élégante [4].

Théorème 29. *Le langage $L_d = \{\langle M \rangle \mid M$ n'accepte pas le mot $\langle M \rangle\}$ n'est pas re.*

Démonstration. Par l'absurde, supposons qu'une machine M_d reconnaisse L_d. Considérons alors l'entrée $\langle M_d \rangle$ donnée à la machine M_d. Deux cas sont possibles.
— Si M_d n'accepte pas $\langle M_d \rangle$ alors, par définition du langage L_d, le mot $\langle M_d \rangle$ est dans L_d. Or M_d reconnait L_d, donc M_d accepte $\langle M_d \rangle$, une contradiction.
— Si M_d accepte $\langle M_d \rangle$ alors, par définition du langage L_d, le mot $\langle M_d \rangle$ n'est pas dans L_d. Or M_d reconnait L_d, donc M_d n'accepte pas $\langle M_d \rangle$, une contradiction.

Dans les deux cas nous arrivons à une contradiction. □

La preuve est cette fois encore plus simple ! Ce résultat nous dit qu'il n'existe pas de MT M_d pour décider si une machine M reconnait le mot $\langle M \rangle$ (même si la machine M_d a le droit de ne pas s'arrêter si M ne reconnait pas $\langle M \rangle$). Ce problème peut sembler artificiel, mais il sert de *graine* pour dériver la non récursivité d'autres problèmes (plus naturels). Cette méthode s'appelle une **réduction**.

Corollaire 30. *Le langage $L_u = \{\langle M \rangle \# w \mid M$ accepte le mot $w\}$ n'est pas récursif. Plus précisément, son complément n'est pas re.*

Démonstration. Supposons le contraire : il existe une machine de Turing $M_{\bar{u}}$ qui reconnait le complémentaire de L_u. En faisant un appel à $M_{\bar{u}}$, nous pouvons alors construire une MT M_d qui reconnait L_d, ce qui contredit le théorème 29.

On construit M_d comme suit. Sur une entrée w,

1. la machine vérifie $w \in L_{enc}$ (lemme 27), si w n'est pas un encodage valide alors M_d s'arrête dans un état non final ;
2. si $w = \langle M \rangle$ est un encodage valide, alors M_d écrit $w \# w = \langle M \rangle \# \langle M \rangle$ sur le ruban et exécute $M_{\bar{u}}$ sur cette entrée.

4. Je vous souhaite d'être de cet avis un jour.

Si $M_{\bar{u}}$ existe il est clair que la machine M_d que nous venons de définir existe également. Cependant, le mot w est accepté par M_d si et seulement si w est un encodage valide d'une MT qui n'accepte pas son propre encodage. En d'autres termes, M_d reconnait L_d, une contradiction. Donc $M_{\bar{u}}$ ne peut pas exister. □

Remarque 31. *Dans les preuves d'indécidabilité nous utilisons souvent un mécanisme appelé* **réduction**. *Pour démontrer qu'un problème P est indécidable, nous supposons par l'absurde qu'il existe un algorithme A qui résout P. Alors nous décrivons un algorithme A' qui résout un problème P' connu pour être indécidable, en faisant appel à A.*

Puisqu'un tel algorithme A' ne peut pas exister (c'est la contradiction), notre hypothèse concernant l'existence d'un algorithme A pour résoudre P est fausse : P est indécidable.

Définition 32. *On dit qu'un problème A se réduit à un problème B si, connaissant une réponse à B, on peut trouver une réponse à A. Plus formellement, on dira qu'un langage A est* **Turing-réductible** *à un langage B si il existe une machine de Turing avec* **oracle** *B, qui résout le problème A. Une machine de Turing avec oracle B est une machine qui peut, au cours de son calcul, obtenir autant de réponses qu'elle souhaite sur des questions d'appartenance au langage B : est-ce que tel $w \in B$? est-ce que tel autre $w' \in B$? Et l'oracle pour le langage B lui donne des réponses oui/non instantanément.*

Dans l'application des réductions Turing aux preuves d'indécidabilité (remarque 31), le point important est qu'une machine de Turing avec un oracle dont le langage est calculable, est elle-même calculable.

Corollaire 33. *Le langage $L_{haltb} = \{\langle M \rangle \mid M$ s'arrête quand on la lance sur l'entrée vide$\}$ n'est pas récursif.*

Démonstration. Par l'absurde, supposons qu'il existe un algorithme A pour décider L_{haltb}. Alors l'algorithme A_u suivant reconnait L_u, ce qui contredit le théorème 30. Sur une entrée M et w,

1. A_u construit le code d'une nouvelle machine de Turing M', telle que M' s'arrête sur l'entrée vide si et seulement M accepte w. Cette machine M' commence par écrire w sur le ruban (cela est possible en utilisant $|w|$ états), puis entre dans l'état initial de M (M' va alors se comporter comme M). Cependant A_u rajoute également des transitions, depuis tous les états non finaux où M s'arrête, vers un état qui boucle à l'infini.
2. A_u utilise l'algorithme A pour savoir si $\langle M' \rangle \in L_{haltb}$, et donne la même réponse que A.

Bilan : A_u va s'arrêter dans un état final si et seulement si $\langle M' \rangle \in L_{haltb}$, c'est-à-dire si et seulement M accepte w, c'est-à-dire décide L_u, une contradiction. □

Voyez-vous la réduction utilisant le théorème 33 pour démontrer le théorème 28 ?

3.8 Théorème de Rice

Nous avons vu que de nombreuses questions sur les MT sont indécidables. Certaines questions sont clairement décidables, comme par exemple : est-ce qu'une MT donnée a 5 états ? Il s'avère cependant que **toute question non triviale qui concerne uniquement le langage reconnu par une MT (plutôt que la machine elle-même) est indécidable**. Une question non triviale étant une question qui n'est pas toujours vraie ou toujours fausse.

Définition 34. *Soit P une famille de langages. On appelle P une* **propriété non triviale** *si il existe deux machines de Turing M_1 et M_2 telles que $L(M_1) \in P$ et $L(M_2) \notin P$.*

Théorème 35. *Soit P une propriété non triviale. Il n'existe aucun algorithme pour décider si une MT M vérifie $L(M) \in P$.*

Démonstration. Nous utilisons une réduction depuis L_u. Sans perte de généralité, nous pouvons supposer $\emptyset \notin P$ (sinon on considère le complément de P au lieu de P). Puisque P est non triviale, il existe une MT M_P telle que $L(M_P) \in P$.

Par l'absurde, supposons qu'il existe un algorithme A pour décider si une machine de Turing M vérifie $L(M) \in P$. Alors l'algorithme A' suivant permet de décider si une machine de Turing M accepte un mot w, contredisant le théorème 30. L'entrée de A' est un mot $\langle M \rangle \# w$.

1. A' commence par construire une nouvelle machine M' qui
   — copie son entrée u sur un ruban séparé pour l'utiliser plus tard ;
   — écrit w sur le ruban et place la tête sur la première lettre de w ;
   — entre dans l'état initial de M. A partir de là A' simule M, en ignorant u, jusqu'à ce que M entre dans son état final ;
   — si M entre dans son état final, alors le mot u est recopié sur un ruban blanc (que des symboles B) et la machine M_P est simulée sur u. On entre dans un état final si M_P accepte u.
2. Ensuite A' donne la machine ainsi construite M' en entrée à l'algorithme A (dont nous supposons l'existence), et retourne la réponse produite par A.

Etant donné n'importe quels $\langle M \rangle$ et w, la machine M' peut effectivement être construite, donc l'algorithme A' décrit ci-dessus existe si l'on suppose que A existe.

La correction de A' (le fait que A' reconnait L_u) est assurée par les observations :
— si M accepte w, alors M' acceptera exactement les mots u que M_P accepte, donc dans ce cas $L(M') = L(M_P) \in P$;
— si M n'accepte pas w, alors M' n'accepte aucun mot u, donc dans ce cas $L(M') = \emptyset \notin P$.

Cet algorithme A' ne peut pas exister car il contredit le théorème 30, donc l'algorithme dont nous avons supposé l'existence, A, n'existe pas. □

Remarque 36. M_1 **simule** M_2 = *M_1 se comporte comme M_2* = *M_1 suit la table de transition de M_2.*

Remarque 37. *Appliquons le théorème de Rice. Les problèmes suivants sont indécidables :*
— est-ce qu'une MT donnée en entrée accepte tous les mots ?
— est-ce que $L(M)$ est un langage régulier pour une MT M ?
— est-ce qu'une MT donnée accepte tous les palindromes ?