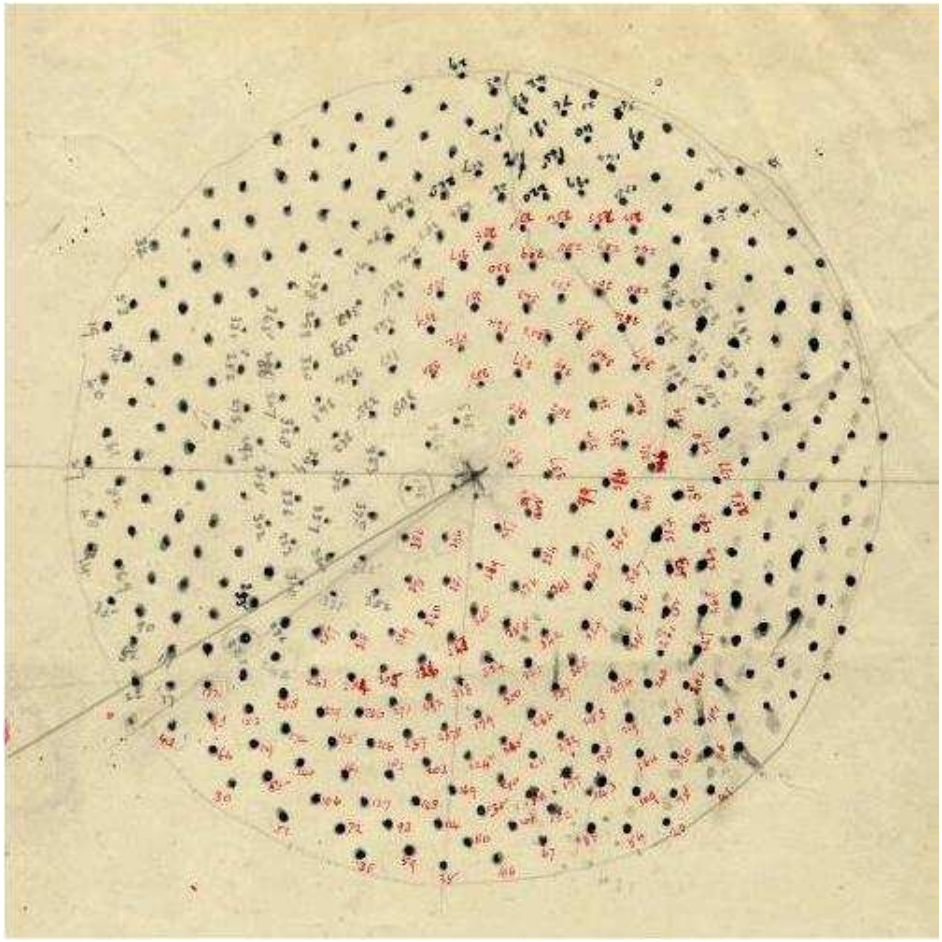
Éléments de Théorie de la Calculabilité



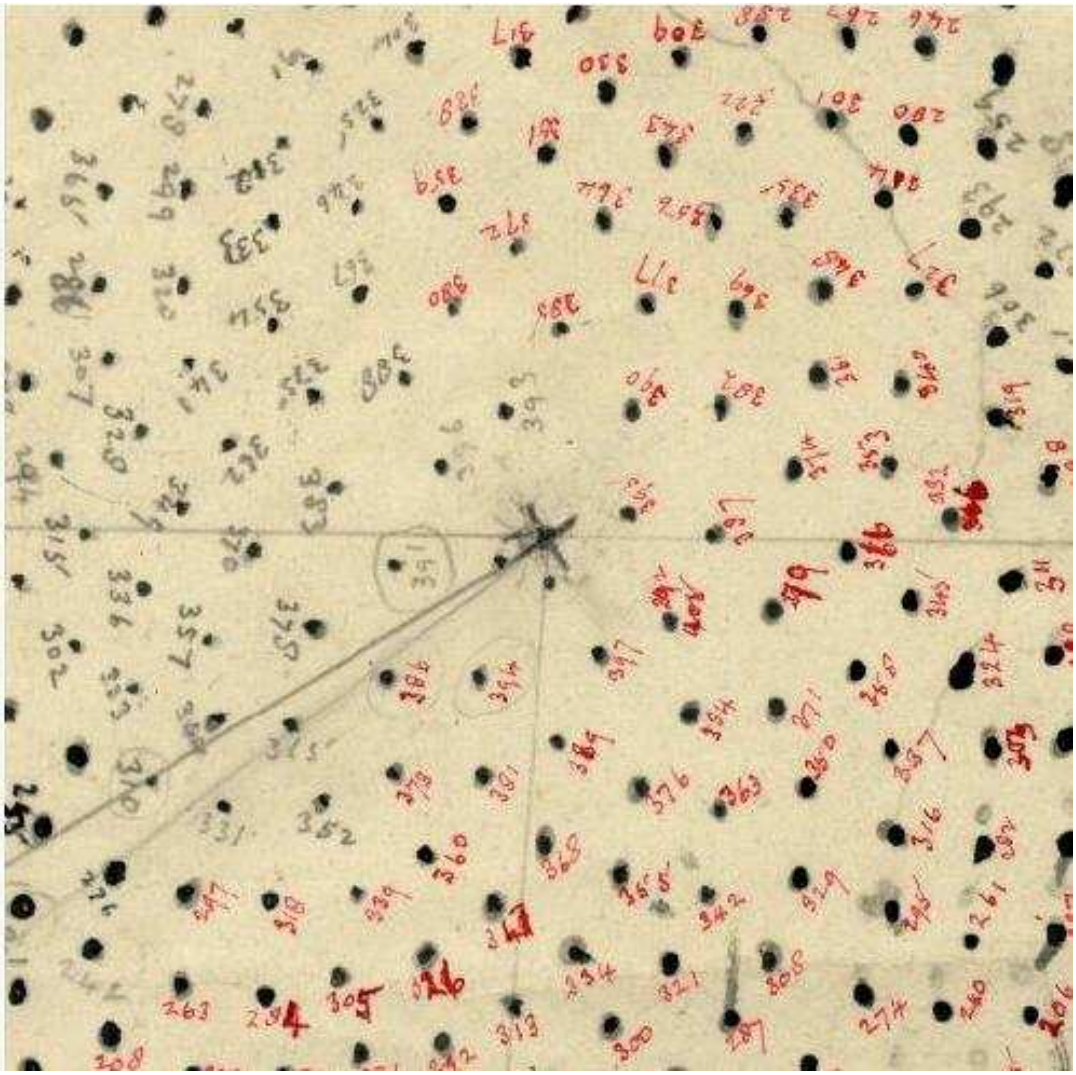
Version préliminaire du 16 octobre 2008
Merci de faire parvenir les corrections à michael@cadilhac.name

Sebastiaan A. Terwijn

Traduit de l'anglais par Michaël Cadilhac

Sebastiaan A. Terwijn
Institute for Discrete Mathematics and Geometry
Technical University of Vienna
Wiedner Hauptstrasse 8–10/E104
A-1040 Vienna, Austria
terwijn@logic.at

Traduction française par Michaël Cadilhac
michael@cadilhac.name



Emil Post
(1897–1954)

Alonzo Church
(1903–1995)

Kurt Gödel
(1906–1978)

Stephen Cole Kleene
(1909–1994)

Alan Turing
(1912–1954)

Table des matières

Chapitre 1

Introduction

Nous exposons dans ce cours les concepts primordiaux de la théorie de la calculabilité, aussi connue sous le nom de théorie de la récursivité. Il existe de nombreux points de vue sur ce que la théorie de la calculabilité *est* exactement. Odifreddi [17, 18] la définit de manière très large comme l'étude des fonctions d'entiers ; une autre école classique la voit comme l'étude de la *définissabilité* (Slaman), accentuant les liens avec la théorie des ensembles. Cela étant, la vision la plus communément admise est celle de l'étude de la calculabilité des fonctions de nombres naturels. La restriction à ces fonctions n'est en rien contraignante ; en effet, comme nous le verrons, les nombres naturels possèdent un incroyable pouvoir de codage, de telle sorte qu'ils peuvent représenter énormément d'objets d'une manière ou d'une autre. Il est aussi possible d'entreprendre l'étude de la calculabilité dans des domaines plus généraux, introduisant entre autre la théorie de la calculabilité d'ordre supérieur (Sacks [24]). Pour autant, ce sujet ne sera pas abordé dans ce document.

Ce cours ne se focalise sur aucun sujet véritablement original ; au contraire, nous avons essayé de présenter synthétiquement un petit ensemble de notions standards et importantes, ainsi que des résultats essentiels de ce domaine désormais classique. Le caractère succinct du document trahit le fait qu'il n'a pas pour vocation de contenir *toutes* les bases, et nous renvoyons le lecteur à la section « Approfondissements » pour une vision plus large. L'avantage de notre approche restreinte est que nous pouvons de manière assez sûre affirmer que tout ce que nous faisons ici est important.

Les seuls prérequis pour la lecture de ce cours sont une certaine familiarité avec le langage des mathématiques et les formalismes connexes. En particulier, notre lecteur potentiel devra avoir quelques connaissances de la logique du premier ordre. Il sera aussi fait certaines références aux cardinaux et à la théorie des ensembles, mais le lecteur qui ne serait pas familier avec ces notions pourra tout à fait passer ces points.

Le présent document se découpe de la manière suivante. Dans le chapitre 2 nous introduisons les notions élémentaires de la théorie de la calculabilité, telles que le concept d'algorithme, les fonctions récursives et les machines de Turing. Nous verrons que les diverses formalisations de la notion informelle d'algorithme conduisent toutes à la *même* notion de fonction calculable. Ce fait remarquable constitue une des pierres angulaires de la théorie de la calculabilité. Au chapitre 3 nous exposons les propriétés standards des ensembles récursifs et récursivement énumérables[1] (r.é.) et faisons une courte incursion dans l'incalculable. Aux chapitres 4 et 5 nous introduisons

[1] Les conventions anglo-saxonnes, que suivait ce document dans sa forme originelle, utilisent plus volontiers « *computable* » et « *computably enumerable (c.e.)* ». Face au manque d'esthétisme de la traduction de ces termes, nous suivrons la tradition française d'utiliser « récursifs » et « récursivement énumérables (r.é.) ».

des outils pour mesurer l'insolvabilité : dans le chapitre 4 nous étudions la hiérarchie arithmétique et les m-degrés, et dans le chapitre 5 nous étudions la calculabilité relative et les degrés de Turing. Dans le chapitre 6 nous introduisons la méthode des priorités et l'utilisons pour répondre à une importante question sur les degrés de Turing des ensembles r.é. Finalement, au chapitre 7 nous donnons quelques applications de la théorie de la calculabilité. En particulier, nous donnons une preuve du premier théorème d'incomplétude de Gödel.

1.1 Préliminaires

Nos notations sont tout ce qu'il y a de plus courantes et suivent celles des ouvrages [17, 27]. ω est l'ensemble des entiers naturels $\{0, 1, 2, 3 \ldots\}$. Par le terme *ensemble*, nous parlerons en règle générale d'un sous-ensemble de ω. L'espace de Cantor 2^ω est l'ensemble des sous-ensembles de ω. Nous identifierons fréquemment un ensemble avec sa fonction caractéristique : pour $A \subseteq \omega$, nous identifierons A avec la fonction $\chi_A : \omega \to \{0, 1\}$ définie par

$$\chi_A(n) = \begin{cases} 1 & \text{si } n \in A, \\ 0 & \text{sinon.} \end{cases}$$

Nous écrirons souvent $A(n)$ à la place de $\chi_A(n)$. Le complément de A, $\omega - A$, sera dénoté par $\overline{A}$. Quand nous parlerons de *fonction (totale)*, nous ferons toujours référence à une fonction de ω^n dans ω, pour un n donné. Nous utiliserons les lettres f, g, h pour dénoter les fonctions totales, et la composition de deux fonctions f et g sera notée $f \circ g$. Une *fonction partielle* est une fonction de ω dans ω qui peut ne pas être définie pour certains arguments. Écrire $\varphi(n)\downarrow$ signifie que φ est définie en n, et écrire $\varphi(n)\uparrow$ qu'elle ne l'est pas. Le *domaine* d'une fonction partielle φ est l'ensemble $\mathrm{Dom}(\varphi) = \{x : \varphi(x)\downarrow\}$, et l'*image* de φ est l'ensemble $\mathrm{Img}(\varphi) = \{y : (\exists x)[\varphi(x) = y]\}$. Nous utiliserons les lettres φ et ψ pour dénoter les fonctions partielles. Pour de telles fonctions, $\varphi(x) = \psi(x)$ signifiera que ou bien les deux valeurs sont non définies, ou bien qu'elles sont définies et égales. Pour faciliter les notations, nous remplacerons souvent un nombre fini d'arguments $x_1, \ldots, x_n$ par $\vec{x}$. Nous utiliserons les notations du λ-calcul pour les fonctions, ainsi $\lambda x_1 \ldots x_n . f(x_1, \ldots, x_n)$ est associé à la fonction faisant correspondre $\vec{x}$ avec $f(\vec{x})$. L'ensemble des chaînes binaires finies sera noté $2^{<\omega}$. Nous utiliserons les lettres σ et τ pour dénoter les chaînes finies. La longueur de σ sera notée $|\sigma|$ et la concaténation de σ et τ sera notée $\sigma^\frown\tau$. Le fait que τ soit une sous-séquence (on dit aussi un *segment initial*) de σ sera noté $\tau \sqsubseteq \sigma$. Pour un ensemble A, $A{\upharpoonright}n$ dénotera la chaîne finie $A(0)^\frown A(1)^\frown \ldots ^\frown A(n-1)$ constituée par les n premiers bits de A.

À la fin de chaque chapitre se trouvent des exercices, dont les plus difficiles sont marqués d'une *.

Chapitre 2

Notions élémentaires

2.1 Algorithmes

Un algorithme est une procédure finie ou un ensemble fini de règles destiné à résoudre un problème étape par étape. Le nom commun *algorithme* est dérivé du nom du mathématicien du 9^{e} siècle al-Khwarizmi et de son ouvrage « Le livre de l'addition et de la soustraction d'après le calcul hindou » qui fut publié en latin sous le titre « Algoritmi de numero Indorum ». Par exemple, la recette pour cuisiner un magret de canard au miel est un algorithme, où le « problème » serait la tâche de cuisiner un magret. Cela étant, le mot *algorithme* est le plus souvent utilisé dans des contextes plus formels. La manière de faire une division longue est un exemple d'algorithme que tout le monde apprend à l'école. Notons aussi qu'en principe n'importe quel programme informatique constitue un algorithme, en tant qu'ensemble fini de règles déterminant à chaque étape l'action à effectuer.

Des siècles durant, aucun besoin ne se faisait sentir de définir formellement ce qu'était un algorithme, et l'adage « vous le reconnaîtrez quand vous en verrez un » était de mise. Les choses changèrent au début du 20^{e} siècle lorsque des problèmes comme les suivants commencèrent à être étudiés :

- (Problème de la décision, dit *Entscheidungsproblem*) Déterminer si une formule logique du calcul du prédicat du premier ordre est une tautologie (c'est-à-dire valide pour toute interprétation). Ce problème remonte jusqu'à Leibniz [13].
- (Dixième problème de Hilbert) Déterminer si une *équation diophantienne*, un polynôme de plusieurs variables à coefficients entiers, admet une solution entière. Ce problème provient d'une liste bien connue donnée par Hilbert à la fin du 19^{e} siècle [8].

Une réponse positive à l'un de ces problèmes pourrait être d'exhiber un algorithme qui les résout. Si quelqu'un veut prouver que les problèmes ci-dessus n'ont pas de solution, il doit prouver qu'il n'existe pas d'algorithme pour les résoudre ; il faut donc précisément savoir *ce qu'est un algorithme* : nous avons besoin d'une définition formelle de la notion informelle d'algorithme. C'est le but de ce premier chapitre.

Il y a beaucoup de manières de formaliser la notion d'algorithme. Nous verrons la propriété remarquable selon laquelle *toutes mènent à la même notion formelle*. Ceci nous donne une solide base pour une théorie mathématique du calcul : la théorie de la calculabilité. Nous donnons deux formalisations parmi les plus connues : les fonctions récursives et les fonctions calculables sur machine de Turing. Nous montrons ensuite que ces deux classes de fonctions coïncident. Le lecteur soucieux d'entrevoir d'autres exemples de formalisation se référera avantageusement à Odifreddi [17].

Pour revenir aux deux problèmes précédents : avec l'aide de la théorie de la cal-

culabilité, il a été prouvé que les deux problèmes étaient insolvables, c'est-à-dire qu'il n'existait pas d'algorithme pour leurs solutions. Pour le premier problème, cela a été prouvé indépendamment par Church [2] et Turing [29] (voir Théorème 7.1.6), et pour le dixième problème de Hilbert, par Matijasevich [15] en s'appuyant sur les travaux de Davis, Putnam et Robinson.

2.2 Récursion

Nous allons dans un premier temps approcher la calculabilité en utilisant la notion de *récursion*. Cette approche, standard s'il en est, est la raison pour laquelle la théorie de la calculabilité est aussi connue sous le nom de *théorie de la récursivité*.

Historiquement, cette approche est la première à avoir vu le jour, et elle est particulièrement connue pour son application dans les théorèmes d'incomplétude de Gödel [7], voir section 7.1. En fait, bien que ces théorèmes ne mentionnent pas la création d'une nouvelle branche théorique, Gödel a inventé quelques notions de base de calculabilité pour pouvoir prouver ses résultats.

2.2.1 Fonctions récursives primitives

La récursion est une méthode pour définir de nouvelles valeurs d'une fonction à partir de valeurs précédemment calculées ou définies. Considérons par exemple la suite suivante :

$$1,\ 1,\ 2,\ 3,\ 5,\ 8,\ 13,\ 21,\ 34\ldots$$

découverte par Fibonacci en 1202. Chaque terme de la suite est obtenue en prenant la somme des deux précédents termes : $F(n+2) = F(n+1) + F(n)$. Pour initialiser la suite, on définit $F(0) = 1$ et $F(1) = 1$. Cette suite apparaît sous diverses formes dans la nature, par exemple dans la phyllotaxie des graines de tournesol (voir le dessin de la couverture par Turing) : en comptant dans le sens des aiguilles d'une montre, on dénombre 21 spirales ; dans le sens inverse, 34 spirales. Dans cet exemple et dans beaucoup d'autres cas (par exemple avec l'ananas ou les pâquerettes) les nombres de ces deux formes de spirales sont toujours deux nombres de Fibonacci consécutifs (voir aussi les exercices 2.5.1 et 2.5.2).

Dans le cas général, on dit qu'une fonction f est définie par récursion à partir de deux fonctions g et h quand sa valeur initiale $f(0)$ est donnée en utilisant g et que pour tout n, $f(n+1)$ est calculée en utilisant h et des valeurs précédentes de f. L'idée principale de la récursion est que si l'on peut calculer g et h, alors on peut calculer f. La récursion est donc un moyen de définir de nouvelles fonctions calculables à partir d'autres fonctions.

Définition 2.2.1 (Récursion primitive, Dedekind [3]) Une fonction f est définie à partir des fonctions g et h par *récursion primitive* si

$$\begin{aligned} f(\vec{x}, 0) &= g(\vec{x}) \\ f(\vec{x}, n+1) &= h(f(\vec{x}, n), \vec{x}, n). \end{aligned}$$

Dans cette définition, $f(n+1)$ est définie en fonction de la valeur précédente $f(n)$. Une forme plus générale de récursion, où $f(n+1)$ est définie en fonction de $f(0), \ldots, f(n)$, est traitée dans l'exercice 2.5.9.

Définition 2.2.2 (Fonctions récursives primitives, Gödel [7]) La classe des *fonctions récursives primitives* est la plus petite classe de fonctions qui

1. contient les fonctions initiales

$$\begin{aligned} O &= \lambda x.0 && \text{(fonction constante zéro)} \\ S &= \lambda x.x+1 && \text{(fonction successeur)} \\ \pi_n^i &= \lambda x_1,\ldots,x_n.x_i && 1 \leq i \leq n, \text{ (fonctions de projection)} \end{aligned}$$

2. est close par composition, c'est-à-dire le schéma défini par

$$f(\vec{x}) = h(g_1(\vec{x}),\ldots,g_n(\vec{x}))$$

   pour des fonctions récursives primitives h et g_i

3. est close par récursion primitive.

Par exemple, Grassmann a donné en 1861 des définitions récursives pour l'addition usuelle $+$ et la multiplication $\cdot$:

$$\begin{aligned} x+0 &= x \\ x+S(y) &= S(x+y), \end{aligned}$$

$$\begin{aligned} x\cdot 0 &= 0 \\ x\cdot S(y) &= (x\cdot y)+x. \end{aligned}$$

On remarquera que la première récursion réduit la définition de $+$ à la seule utilisation de la fonction initiale S et que la seconde réduit la multiplication à l'addition. Pour une preuve formelle que $+$ et $\cdot$ sont récursives primitives, voir l'exercice 2.5.3.

2.2.2 Fonctions récursives

La construction de la définition 2.2.2 montre bien que toute fonction récursive primitive est totale. Ce qui suit est un argument informel montrant qu'il est essentiel de considérer aussi les fonctions partielles.

Dans toutes les formalisations de la notion d'algorithme, un algorithme sera synthétiquement défini par un ensemble fini de symboles. De ce point de vue, toute fonction calculable sera un objet fini. La théorie nous amènera à reconnaître et effectivement manipuler ces objets, il est donc naturel de se donner un moyen de *lister* toutes les fonctions calculables en énumérant leurs algorithmes. Il doit exister une fonction calculable f telle que $f(n)$ est un code pour le n-ième algorithme, qui calcule la n-ième fonction calculable f_n.

Supposons maintenant que nous ne considérions que les fonctions totales. La fonction $F = \lambda n.f_n(n)+1$ serait donc une fonction calculable, mettons la m-ième. On a alors

$$F(m) = f_m(m) \neq f_m(m)+1 = F(m),$$

ce qui constitue une contradiction. Cette contradiction est due au fait que nous avons supposé que la valeur $f_m(m)$ était toujours définie. Cette supposition n'est plus obligatoire dès lors que nous incluons les fonctions partielles dans notre théorie.

Cet argument suggère que la classe des fonctions récursives primitives n'est pas assez grande pour contenir toutes les fonctions intuitivement calculables. Nous ajoutons donc un nouvel opérateur à notre ensemble d'outil : l'opérateur μ.
Pour un prédicat R,

$$(\mu n)\big[R(\vec{x},n)\big]$$

correspond au plus petit nombre n tel que $R(\vec{x}, n)$ soit vrai. Si un tel n n'existe pas, alors $(\mu n)[R(\vec{x}, n)]$ n'est pas défini. En fait, l'opérateur μ nous permet de *chercher* certaines valeurs et comme cette recherche peut être non bornée, l'opérateur μ nous fait sortir du domaine des fonctions totales.

Définition 2.2.3 (Fonctions récursives, Kleene [10]) La classe des *fonctions partielles récursives* est la plus petite classe de fonctions qui

1. contient les fonctions initiales,
2. est close par composition et récursion primitive,
3. est close par le schéma de μ-récursion, c'est-à-dire le schéma qui défini la fonction partielle
$$\varphi(\vec{x}) = (\mu y)\big[(\forall z \leq y)[\psi(\vec{x}, z)\downarrow] \wedge \psi(\vec{x}, y) = 0\big]$$
à partir d'une fonction partielle récursive ψ.

Une *fonction récursive (générale)* est une fonction partielle récursive qui s'avère être totale.

Remarquons que dans le schéma de μ-récursion, l'opérateur μ recherche le plus petit y tel que $\psi(\vec{x}, y) = 0$, mais il ne peut pas « sauter » au dessus de valeurs non définies de ψ à cause de la première clause. Cela correspond à l'intuition du fait que la recherche d'un tel y est effectuée en rejetant les valeurs de z pour lesquelles $\psi(\vec{x}, z) \neq 0$. De plus, comme nous le verrons plus tard, il est indécidable de savoir si une fonction converge, donc supprimer la première clause créerait une classe de fonctions trop importante (voir aussi l'exercice 2.5.17). Dans la suite, cette clause sera implicite et l'emploi de l'opérateur μ la sous-entendra.

On peut aisément voir que la classe des fonctions partielles récursives étend strictement celle des fonctions récursives primitives, puisqu'elle contient des fonctions partielles. Cela étant, l'ajout de l'opérateur μ permet aussi la définition de nouvelles fonctions totales. Des exemples seront simples à construire (voir exercice 2.5.12) une fois que nous aurons abordé les méthodes d'arithmétisation. Cette méthode nous permettra aussi de formaliser la remarque faite en début de section en donnant une preuve que les fonctions partielles récursives peuvent être effectivement énumérées, au contraire des fonctions récursives totales.

2.3 Machines de Turing

Dans cette section nous considérons une formalisation complètement différente de la notion d'algorithme : les machines de Turing [29]. Au lieu de construire des fonctions de manière inductive à partir d'autres fonctions, comme dans la définition des fonctions récursives, cette approche s'intéresse à savoir ce que « calculer une fonction » signifie, en utilisant une machine idéalisée. Intuitivement, on imagine un ruban de longueur non bornée que l'on considérera arbitrairement infini à droite. Ce ruban est composé de cases en quantité dénombrable qui contiennent ou bien un '1' (« bâton ») ou bien un blanc, c'est-à-dire rien. On imagine ensuite que la manipulation de ces symboles est entièrement dictée par un ensemble de règles ou *programme*. L'opération élémentaire que l'on peut effectuer après la lecture d'une case est de changer son contenu et de se déplacer d'une case à gauche ou à droite. Cette opération dépend seulement du contenu de la case lue et de *l'état courant* du programme. Cette suite d'actions que l'on effectue peut aussi bien se terminer ou non. Si l'exécution se termine, on dit que l'on a terminé un calcul, et l'on peut considérer le contenu du ruban comme le

résultat. Pour représenter l'intuition que ces calculs doivent être de complexité limitée, on contraint que l'ensemble des états doit être fini. Bien entendu, plutôt que de faire avancer les étapes de calcul nous-mêmes, on peut imaginer un appareil qui le fait de manière automatique. On arrive ainsi à l'image d'une machine de Turing de la figure 2.1.

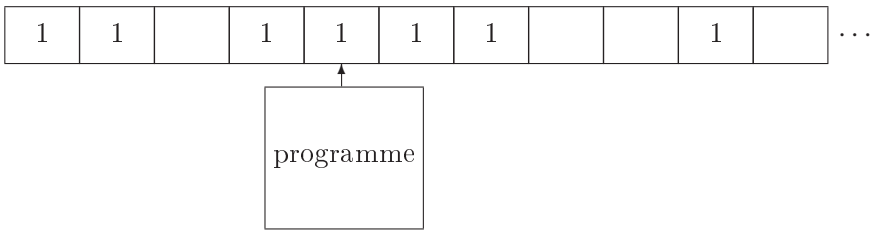


FIG. 2.1 – Une machine de Turing

De manière plus formelle, les instructions d'un programme de machine de Turing sont des quintuples

$$(q_i,\, x,\, q_j,\, y,\, X),$$

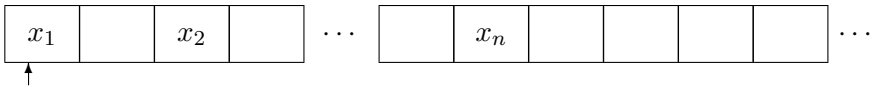
où q_i et q_j sont pris dans un ensemble fini Q *d'états*, x et y sont ou 1 ou B (pour « blanc »), et $X \in \{\lhd, \rhd\}$. Cette instruction signifie que si la machine est dans l'état q_i et qu'elle lit une case dont le contenu est x, ce contenu est changé en y ; la machine passe alors dans l'état q_j et la tête de lecture est déplacée d'une case dans la direction X. Une machine de Turing est un ensemble d'instructions décidant que faire dans toutes les situations possibles dans lesquelles la machine peut être. Elle peut donc être définie comme une fonction

$$M : Q \times \{1, B\} \to Q \times \{1, B\} \times \{\lhd, \rhd\}.$$

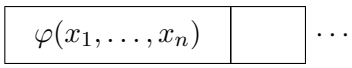
Par convention, si la tête de lecture de la machine est sur la case la plus à gauche et reçoit l'instruction de se déplacer à gauche, le calcul est indéfini. De plus, on considère que l'ensemble Q contient deux états distingués : un *état initial* q_0 et un *état final* q_f. La machine démarre en étant dans l'état q_0 et elle s'arrête dès qu'elle atteint l'état q_f.

Les machines de Turing peuvent être vues comme des ordinateurs idéalisés, avec des ressources en temps et en espace illimitées. Pour continuer le parallèle avec l'informatique, ce que nous appelons ici une « machine » est un programme, et le ruban et la tête de lecture sont le matériel (*hardware*). De plus, une machine de Turing étant un ensemble fini d'instructions, elle peut être représentée par un diagramme d'états fini. Enfin, le ruban peut être vu comme une mémoire externe illimitée. Pour modéliser les ordinateurs d'une façon plus réaliste, il est possible d'ajouter des contraintes en temps et espace : cela conduit à une branche plus jeune de la théorie de la calculabilité appelée la *théorie de la complexité*. Nous n'abordons pas les nombreux résultats et questions de ce domaine, mais nous renvoyons le lecteur à Odifreddi [18] pour plus d'informations et de références.

Définition 2.3.1 Une fonction partielle n-aire φ est *partielle calculable* s'il existe une machine de Turing M qui la calcule. Autrement dit, supposons que pour $x_1, \ldots, x_n$ on ait la configuration initiale suivante :



où une case contenant x_i signifie en fait $x_i + 1$ cases consécutives contenant des 1 (c'est la représentation dite *pointée* de x_i), où les autres cases sont blanches et où la tête de lecture est sur la case la plus à gauche. La fonction φ est partielle calculable si l'on obtient par exécution de la machine (donc par applications successives de sa fonction de transition en partant de l'état q_0) après un nombre fini d'étapes, l'arrêt dans l'état final q_f avec la configuration suivante :



La tête de lecture est alors positionnée n'importe où, $\varphi(x_1, \ldots, x_n)$ est en représentation pointée, et l'on ne considère pas ce qu'il y a après la case blanche suivant $\varphi(x_1, \ldots, x_n)$, ces cases pouvant contenir des résidus du calcul. De plus, pour tout $x_1, \ldots, x_n$ tels que $\varphi(x_1, \ldots, x_n)\uparrow$, la machine M ne s'arrête pas.

Une fonction est *calculable* si elle est partielle calculable et totale. Un ensemble A est *calculable* si sa fonction caractéristique χ_A est calculable.

Il doit être remarqué que cette définition est très solide et permet beaucoup de variations ne changeant pas son sens profond. Par exemple, le ruban est souvent vu comme étant infini à gauche et à droite, la tête de lecture comme pouvant rester au même endroit après une instruction, ou la machine comme disposant de plusieurs rubans, etc. Il est aussi tout à fait possible de considérer des alphabets plus grand que $\{1, B\}$. Il sera d'ailleurs évident après avoir vu les techniques de codage de la section 2.4 que n'importe quel alphabet fini peut être codé avec deux symboles, ce n'est donc pas une limitation.

Le lecteur devra se familiariser avec les machines de Turing en les manipulant quelque peu. Une approche pas à pas montrera que les fonctions simples sont calculables, et qu'à partir de celles-ci, on peut construire des fonctions plus compliquées, comme cela avait été fait avec les fonctions récursives.

Exemple 2.3.2 Le programme suivant calcule la fonction constante 0. Il suffit, quelque soit l'entrée sur le ruban, d'écrire 1 (la représentation pointée de 0) dans la case la plus à gauche et un blanc sur la seconde case, puis s'arrêter. Comme la machine démarre par convention sur la cellule la plus à gauche, le programme est le suivant :

q_0	1	q_1	1	$\triangleright$	se déplacer d'une case à droite
q_1	1	q_f	B	$\triangleleft$	s'il y a un 1, le remplacer par un blanc et s'arrêter
q_1	B	q_f	B	$\triangleleft$	s'il y a un blanc, s'arrêter.

Il faut remarquer que cette description n'est que partielle : seules les instructions réellement utilisées sont listées. Formellement, le programme devrait aussi donner des instructions pour les autres combinaisons d'états et de valeurs de case, mais comme elles n'interviendrons pas, on ne les écrit pas. □

Exemple 2.3.3 Le programme suivant calcule la fonction successeur. Il lit simplement l'entrée jusqu'à arriver à un blanc, le remplace par un 1 et s'arrête :

q_0	1	q_0	1	$\triangleright$	se déplacer à droite tant qu'il y a des 1
q_0	B	q_f	1	$\triangleleft$	s'il y a un blanc, le remplacer par un 1 et s'arrêter.

□

Exemple 2.3.4 Le programme suivant recopie son entrée. Ce genre de routine est utile pour construire des programmes plus importants.

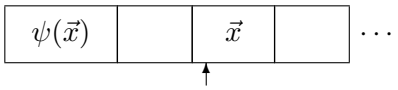
q_0	1	q_1	B	$\triangleright$	changer le 1 initial en un B, se déplacer à droite
q_1	1	q_1	1	$\triangleright$	continuer à droite
q_1	B	q_2	B	$\triangleright$	jusqu'au prochain blanc,
q_2	1	q_2	1	$\triangleright$	continuer à droite
q_2	B	q_3	1	$\triangleleft$	jusqu'au deuxième blanc, le changer en 1 et aller à gauche
q_3	1	q_3	1	$\triangleleft$	revenir
q_3	B	q_4	B	$\triangleleft$	au blanc précédent
q_4	1	q_4	1	$\triangleleft$	continuer à gauche
q_4	B	q_5	1	$\triangleright$	jusqu'au premier blanc et le changer en 1
q_5	B	q_f	B	$\triangleright$	si on était à côté du second blanc, s'arrêter
q_5	1	q_1	B	$\triangleright$	sinon, changer le 1 en B et répéter.

À la première instruction, le 1 initial (qui existe toujours) est effacé pour permettre un retour au début de la bande et nous éviter d'en sortir (voir la convention prise page 7). □

Le résultat suivant donne la mesure du pouvoir expressif des machines de Turing :

Théorème 2.3.5 (Turing [29]) *Toute fonction partielle récursive est partielle calculable.*

Preuve. La preuve est faite par induction sur la définition des fonctions récursives. Il suffit de montrer que les fonctions initiales sont calculables et que la composition, la récursion primitive et le schéma de μ-récursion peuvent tous être simulés sur une machine de Turing. Pour faciliter l'induction, on va montrer une version légèrement plus forte du théorème : toute fonction partielle récursive ψ peut être calculée d'une façon dite *soigneuse*, c'est-à-dire que pour chaque calcul tel que $\psi(\vec{x})\downarrow$, la machine s'arrête dans la configuration



avec $\psi(\vec{x})$ et $\vec{x}$ en notation pointée et la tête de lecture sur la case la plus à gauche de $\vec{x}$. L'avantage des calculs soigneux est qu'ils préservent l'entrée et laissent la tête de lecture à une place connue, ce qui facilite la fusion de programmes.

Nous laissons au lecteur (exercice 2.5.7) l'écriture de programmes soigneux pour les fonctions initiales (ce seront des programmes bien plus imposants que ceux des exemples 2.3.2 et 2.3.3). Le programme de l'exemple 2.3.4 est une machine soigneuse pour la fonction projection $\pi^1_1 = \lambda x.x$.

Nous donnons ci-dessous une description informelle des cas de la composition, de la récursion primitive et du schéma de μ-récursion. Le lecteur est invité à compléter la preuve.

Composition. Supposons que φ est définie par $\varphi(\vec{x}) = \chi(\psi_1(\vec{x}), \ldots, \psi_n(\vec{x}))$. Par hypothèse d'induction, χ et les ψ_i peuvent être calculées par des machines soigneuses. Sur l'entrée $\vec{x}$, on applique une machine soigneuse pour ψ_1. Cela nous donne une sortie qui nous permet d'appliquer immédiatement une machine soigneuse pour ψ_2, et ainsi de suite jusqu'à ψ_n. La sortie finale est alors $z_1, \ldots, z_n, \vec{x}$, où $z_i = \psi_i(\vec{x})$. La conservation finale de $\vec{x}$ est faite en l'échangeant avec les $z_1, \ldots, z_n$ et en déplaçant

la tête de lecture sur la case de z_1 la plus à gauche. On applique alors une machine soigneuse pour χ. Le ruban contient alors $\vec{x}, \chi(z_1, \ldots, z_n), z_1, \ldots, z_n$. Finalement, il suffit de supprimer les z_i et de replacer $\vec{x}$ pour s'arrêter avec $\chi(z_1, \ldots, z_n), \vec{x}$.

Récursion primitive. Supposons que φ soit définie par

$$\begin{aligned} \varphi(\vec{x}, 0) &= \psi(\vec{x}) \\ \varphi(\vec{x}, n+1) &= \chi(\varphi(\vec{x}, n), \vec{x}, n). \end{aligned}$$

En se donnant $\vec{x}, n$, on utilise n comme compteur décroissant vers 0. Sur l'entrée $\vec{x}, n$, on modifie le ruban pour obtenir $n, \vec{x}, m$ où m est initialement à 0 (toujours en notation pointée). On applique alors une machine soigneuse pour ψ afin d'obtenir la configuration $n, \psi(\vec{x}), \vec{x}, 0$. Ensuite, si $n = 0$, on agence le ruban pour qu'il contienne $\psi(\vec{x}), \vec{x}, 0$ et l'on s'arrête. Si $n > 0$, alors on décrémente n et incrémente m de un. On déplace ensuite la tête de lecture sur la bonne position pour exécuter une machine soigneuse pour χ et obtenir la sortie $n, \chi(\varphi(\vec{x}, m), \vec{x}, m), \varphi(\vec{x}, m), \vec{x}, m$. Selon la nouvelle valeur de n, si $n = 0$ alors m est égal à la valeur initiale de n, on peut donc effacer n et $\varphi(\vec{x}, m)$ et déplacer le reste vers la gauche pour s'arrêter avec $\chi(\varphi(\vec{x}, m), \vec{x}, m), \vec{x}, m$. Si $n > 0$, on efface $\varphi(\vec{x}, m)$ et l'on répète la procédure.

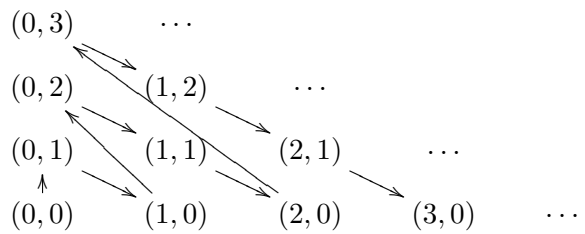
Schéma de μ-récursion. Supposons que $\varphi(\vec{x}) = \mu y(\psi(\vec{x}, y) = 0)$. Sur l'entrée $\vec{x}$, on écrit sur le ruban $\vec{x}, n$, où n vaut initialement 0. On applique ensuite une machine soigneuse pour ψ pour obtenir une configuration $\psi(\vec{x}, n), \vec{x}, n$. On vérifie si $\psi(\vec{x}, n) = 0$. Si c'est le cas, on peut s'arrêter avec la valeur $\vec{x}, n$. sinon, on efface $\psi(\vec{x}, n)$, déplace $\vec{x}, n$ à l'extrême gauche, incrémente n et l'on répète la procédure. □

2.4 Arithmétisation

L'arithmétisation est l'établissement d'une correspondance entre un langage donné et le langage de l'arithmétique, de manière à ce que les propositions faites dans le langage initial soient transformées en propositions sur les nombres naturels. Cette méthode était déjà envisagée par Leibniz [13], mais, encore une fois, elle fut mise en œuvre par Gödel [7]. Dans cette section, nous utilisons l'arithmétisation pour prouver un théorème de forme normale pour les fonctions partielles récursives et prouver la réciproque du théorème 2.3.5.

2.4.1 Fonctions de codage

Nous commençons par donner un codage du plan ω^2. La méthode employée est un simple parcours en zigzag de l'ensemble :



Ce comptage des paires est exprimé par la *fonction de jumelage* $\lambda x, y.\langle x, y\rangle$ de ω^2 dans ω définie par

$$\langle x, y\rangle = \frac{1}{2}\Big((x+y+1)^2 - (x+y+1)\Big) + x = \frac{1}{2}\Big((x+y)^2 + 3x + y\Big).$$

Nous désirons maintenant un mécanisme de codage pour des suites de tailles arbitraires $x_1, \ldots, x_n$. Il est tout à fait possible d'utiliser la fonction de jumelage plusieurs fois, mais nous prendrons plutôt un codage utilisant les nombres premiers $p_0, p_1, \ldots$ (voir exercice 2.5.4). On définit le codage pour la suite $x_1, \ldots, x_n$ comme étant

$$\langle x_0, \ldots, x_n \rangle = p_0^{x_0+1} \cdot p_1^{x_1+1} \cdots p_n^{x_n+1}.$$

Le code ainsi défini n'est pas surjectif, mais cela n'a pas d'importance. Pour faciliter la manipulation de ces codes, on définit quelques fonctions et relations auxiliaires :

- La fonction $exp(x, p_i)$ donne l'exposant de p_i dans la décomposition en facteur premier de x (voir exercice 2.5.5).
- $(x)_i = exp(x, p_i) \dot{-} 1$, qui donne le $i+1$-ième élément x_i de la suite dont le code est x, avec $\dot{-}$ la soustraction tronquée définie à l'exercice 2.5.4. Cette fonction est définie même si x n'est pas le code d'une suite.
- La longueur d'une suite x est définie par

$$lng(x) = \max\big\{n \leq x : \forall m \leq n.\ exp(x, p_m) > 0\big\}.$$

- $Seq(x) \iff (\forall i \leq x)\big[exp(x, p_i) > 0 \to i \leq lng(x)\big]$. Tout x tel que $Seq(x)$ est un code de suite.

2.4.2 Le théorème de forme normale

Nous donnons ici un premier exemple d'arithmétisation : le codage des fonctions partielles récursives dans l'arithmétique. Cela conduira à plusieurs théorèmes très utiles sur les fonctions récursives.

Théorème 2.4.1 (Théorème de forme normale, Kleene [10]) *Il existe une fonction récursive primitive U et des relations récursives primitives T_n, $n \geq 1$, telles que pour toute fonction partielle récursive φ, il existe un nombre e (appelé* code *de φ) tel que*

$$\varphi(x_1, \ldots, x_n) = U\big(\mu y T_n(e, x_1, \ldots, x_n, y)\big).$$

Preuve. L'idée principale est de coder les calculs par des nombres en utilisant le codage des suites vu à la section précédente. Le prédicat $T_n(e, x_1, \ldots, x_n, y)$ signifiera alors que y est un calcul de la fonction dont le code est e sur les arguments $x_1, \ldots, x_n$. La recherche $\mu y T_n(e, x_1, \ldots, x_n, y)$ retournera ensuite un de ces calculs valides et U extraira les composantes du résultat réel. La conception du codage n'est en rien difficile, mais c'est une quantité de travail conséquente. Cela étant, c'est un exercice qui doit être fait au moins une fois.

Codage des fonctions partielles récursives. On associe de manière inductive des nombres aux fonctions partielles récursives, en reprenant leur définition :

- $\langle 0 \rangle$ correspond à O.
- $\langle 1 \rangle$ à S.
- $\langle 2, n, i \rangle$ à π_n^i.
- $\langle 3, a_1, \ldots, a_n, b \rangle$ à $\varphi(\vec{x}) = \chi(\psi_1(\vec{x}), \ldots, \psi_n(\vec{x}))$, où $a_1, \ldots, a_n$ et b correspondent respectivement à $\psi_1, \ldots, \psi_n$ et χ.
- $\langle 4, a, b \rangle$ à $\varphi(\vec{x}, y)$ définie par récursion primitive à partir de ψ et χ, où a et b correspondent respectivement à ψ et χ.
- $\langle 5, a \rangle$ à $\varphi(\vec{x}) = \mu y(\psi(\vec{x}, y) = 0)$, où a correspond à ψ.

Arbres de calcul. Dans un tel arbre, chaque nœud représente une étape de calcul de la manière suivante :

- Aux feuilles sont écrits les calculs effectués par les fonctions initiales. Ces étapes sont donc de la forme $\varphi(x) = 0$, $\varphi(x) = x + 1$ ou $\varphi(x_1, \ldots, x_n) = x_i$.
- Composition : si $\varphi(\vec{x}) = \chi(\psi_1(\vec{x}), \ldots, \psi_n(\vec{x}))$ alors le nœud $\varphi(\vec{x}) = z$ a les $n+1$ prédécesseurs

$$\psi_1(\vec{x}) = z_1, \ldots, \psi_n(\vec{x}) = z_n,\ \chi(z_1, \ldots, z_n) = z.$$

- Récursion primitive : si $\varphi(\vec{x}, y)$ est définie par récursion primitive à partir de ψ et χ, alors le nœud $\varphi(\vec{x}, 0) = z$ a pour prédécesseur $\psi(\vec{x}) = z$, et le nœud $\varphi(\vec{x}, y+1) = z$ a pour prédécesseurs $\varphi(\vec{x}, y) = z_1$ et $\chi(z_1, \vec{x}, y) = z$.
- Schéma de μ-récursion : si $\varphi(\vec{x}) = \mu y(\psi(\vec{x}, y) = 0)$ alors le nœud $\varphi(\vec{x}) = z$ a pour prédécesseurs

$$\psi(\vec{x}, 0) = t_0, \ldots, \psi(\vec{x}, z-1) = t_{z-1},\ \psi(\vec{x}, z) = 0$$

  où les t_i sont tous différents de 0.

Codage des arbres de calcul. Les nœuds de l'arbre de calcul sont donc tous de la forme $\varphi(\vec{x}) = z$. On les code avec des nombres $\langle e, \langle x_1, \ldots, x_n\rangle, z\rangle$ où e est un code pour φ. On assigne alors des codes aux arbres de manière inductive. Chaque arbre T se compose d'un nœud v avec un nombre fini (possiblement nul) d'arbres prédécesseurs T_i, $i = 1 \ldots m$. Si les sous-arbres T_i ont reçu les codes $\widehat{T}_i$, on donne à T le code $\widehat{T} = \langle v, \widehat{T}_1, \ldots, \widehat{T}_m\rangle$.

La propriété d'être un arbre de calcul est récursive primitive. On montre qu'il existe un prédicat récursif primitif $T(y)$ qui exprime le fait que y est un code d'arbre de calcul. On utilise pour cela les fonctions de décodage $(x)_i$ de la section 2.4.1. Pour simplifier les notations, on écrira $(x)_{i,j,k}$ au lieu de $(((x)_i)_j)_k$. Soit y un code d'arbre de calcul, on a $y = \langle v, \widehat{T}_1, \ldots, \widehat{T}_m\rangle$, et donc

$$\begin{aligned}
(y)_1 &= \langle e, \langle x_1, \ldots, x_n\rangle, z\rangle \\
(y)_{1,1} &= e \\
(y)_{1,2} &= \langle x_1, \ldots, x_n\rangle \\
(y)_{1,3} &= z \\
(y)_{i+1} &= \widehat{T}_i \\
(y)_{i+1,1} &= \text{code d'un nœud de } \widehat{T}_i.
\end{aligned}$$

On exprime $T(y)$ en fonction d'un nombre de plus petits blocs. Tout d'abord, soit

$$A(y) \iff \mathit{Seq}(y) \wedge \mathit{Seq}((y)_1) \wedge \mathit{lng}((y)_1) = 3 \wedge \mathit{Seq}((y)_{1,1}) \wedge \mathit{Seq}((y)_{1,2}).$$

Ensuite, on définit les prédicats récursifs primitifs B, C, D et E correspondant respectivement aux cas des fonctions initiales, de la composition, de la récursion primitive et du schéma de μ-récursion. On n'écrira que le prédicat B, les autres étant à traiter dans l'exercice 2.5.8. Pour les fonctions initiales, il y a trois possibilités pour $v = (y)_1$:

$$\begin{gathered}
\langle\langle 0\rangle, \langle x\rangle, 0\rangle \\
\langle\langle 1\rangle, \langle x\rangle, x+1\rangle \\
\langle\langle 2, n, i\rangle, \langle x_1, \ldots, x_n\rangle, x_i\rangle.
\end{gathered}$$

On définit donc

$$\begin{aligned} B(y) \iff\ & lng(y) = 1 \wedge \\ & \Big[\big[(y)_{1,1} = \langle 0\rangle \wedge lng((y)_{1,2}) = 1 \wedge (y)_{1,3} = 0\big] \vee \\ & \big[(y)_{1,1} = \langle 1\rangle \wedge lng((y)_{1,2}) = 1 \wedge (y)_{1,3} = (y)_{1,2,1} + 1\big] \vee \\ & \big[lng((y)_{1,1}) = 3 \wedge (y)_{1,1,1} = 2 \wedge (y)_{1,1,2} = lng((y)_{1,2}) \wedge \\ & 1 \leq (y)_{1,1,3} \leq (y)_{1,1,2} \wedge (y)_{1,3} = ((y)_{1,2})_{(y)_{1,1,3}}\big]\Big]. \end{aligned}$$

On peut définir de manière équivalente $C(y)$ comme étant le prédicat récursif primitif exprimant que y est un code d'un arbre dont le nœud v est un calcul défini par composition des calculs des arbres prédécesseurs. Donc $lng((y)_{1,1}) \geq 3$, $(y)_{1,1,1} = 3$, et ainsi de suite. De même, on peut définir $D(y)$ exprimant la même chose pour le cas de la récursion primitive et $E(y)$ pour le cas du schéma de μ-récursion (voir exercice 2.5.8).

Ayant traité l'ensemble des cas, on peut définir de manière inductive

$$\begin{aligned} T(y) \iff\ & A(y) \wedge \big[B(y) \vee C(y) \vee D(y) \vee E(y)\big] \wedge \\ & \big[lng(y) > 1 \to (\forall i)_{2 \leq i \leq lng(y)} T((y)_i)\big]. \end{aligned}$$

Le prédicat T est récursif primitif car il est défini en n'utilisant que des fonctions récursives primitives appliquées à des valeurs précédemment calculées (et on a toujours $(y)_i < y$). Seul le schéma d'induction complète de l'exercice 2.5.9 a donc été utilisé.

Définition de T_n et de U. On définit finalement pour chaque $n \geq 1$,

$$T_n(e, x_1, \ldots, x_n, y) \iff T(y) \wedge (y)_{1,1} = e \wedge (y)_{1,2} = \langle x_1, \ldots, x_n\rangle$$

et

$$U(y) = (y)_{1,3}.$$

Il est immédiat de voir que T_n et U sont récursifs primitifs. Vérifions la validité de ces fonctions : soit φ une fonction partielle récursive n-aire. On peut trouver un code de φ, mettons e. Pour tout $x_1, \ldots, x_n$, $\varphi(x_1, \ldots, x_n)\downarrow$ si et seulement s'il existe un arbre de calcul pour $\varphi(x_1, \ldots, x_n)$. La valeur de $\varphi(x_1, \ldots, x_n)$ est extraite par U, on a donc bien

$$\varphi(x_1, \ldots, x_n) = U\big(\mu y T_n(e, x_1, \ldots, x_n, y)\big).$$

pour tout $x_1, \ldots, x_n$. □

Le théorème 2.4.1 montre en particulier que toute fonction partielle récursive peut être définie avec *une seule* utilisation de l'opérateur μ.

2.4.3 L'équivalence fondamentale et la thèse de Church

Théorème 2.4.2 (Turing) *Toute fonction partielle calculable est partielle récursive.*

Preuve. Une fois de plus, ceci est prouvé par arithmétisation. Au lieu de coder les calculs des fonctions partielles récursives comme dans la preuve du théorème 2.4.1, on doit maintenant coder les calculs d'une machine de Turing. Ici aussi, cela peut être fait en utilisant des fonctions de codage récursives primitives. Après avoir vu la preuve du théorème 2.4.1, le lecteur devrait avoir une idée assez précise du mode opératoire.

Comme précédemment, un prédicat récursif primitif T_n et une fonction d'extraction U peuvent être définis avec un sens similaire. On définit alors $T_n(e, x_1, \ldots, x_n, y)$ qui signifie que y correspond à un calcul de machine de Turing de code e sur l'entrée $x_1, \ldots, x_n$. Nous laissons au lecteur le soin de mener cette preuve. □

En combinant les théorèmes 2.3.5 et 2.4.2 on obtient l'équivalence fondamentale suivante :

Théorème 2.4.3 *Une fonction est partielle calculable si et seulement si elle est partielle récursive.*

Après avoir vu cette équivalence et insisté sur le fait qu'il existe beaucoup d'autres formalisations équivalentes, on peut être plus informel dans les descriptions des algorithmes, puisque l'écriture exacte n'est généralement pas utile. Parmi les autres formalisations de la notion d'algorithme, on peut citer *la définissabilité finie*, *la représentabilité dans certains systèmes formels* (voir théorème 7.1.4), *être calculable par un programme « tant que »*, *la λ-définissabilité*, *la calculabilité sur machines à registres* et *les grammaires formelles* (voir Odifreddi [17] pour plus de détails). On peut montrer que toutes ces formalisations sont équivalentes en utilisant une méthode d'arithmétisation. Ces équivalences sont habituellement perçues comme des indications que la notion informelle d'algorithme est bien formalisée par n'importe laquelle de ces méthodes.

Thèse de Church *Toute fonction algorithmiquement calculable (au sens intuitif) est calculable (au sens de Turing).*

Cette thèse n'est en aucun cas un théorème, puisqu'elle n'a pas d'écriture formelle et ne peut donc pas être prouvée. Il s'agit plutôt d'une proposition exprimant une certaine confiance dans la solidité de la théorie.

Le théorème 2.4.3 nous permet d'utiliser indifféremment les termes « récursif » et « calculable » (sur machine de Turing). L'usage moderne anglais est de préférer le terme « *computable* », au contraire de l'usage français, que nous suivrons, qui est de parler de fonctions récursives.

Définition 2.4.4 Soient U et T_n comme dans le théorème de forme normale 2.4.1.

1. φ_e^n, aussi notée $\{e\}^n$, est la *e-ième fonction partielle récursive n-aire* :

$$\varphi_e^n = U\big(\mu y\, T_n(e, x_1, \ldots, x_n, y)\big).$$

   Pour l'arité $n = 1$ on supprime généralement le n et l'on écrit simplement φ_e et $\{e\}$. On omet aussi souvent le n si l'arité des fonctions est claire dans le contexte.

2. $\varphi_{e,s}^n$, aussi notée $\{e\}_s^n$, est *l'approximation finie à s étapes* de φ_e^n :

$$\varphi_{e,s}^n(\vec{x}) = \begin{cases} \varphi_e^n(\vec{x}) & \text{si } (\exists y < s)\big[T_n(e, x_1, \ldots, x_n, y)\big], \\ \uparrow & \text{sinon.} \end{cases}$$

On peut désormais clarifier et préciser la discussion faite au début de la section 2.2.2.

Théorème 2.4.5 (Théorème d'énumération) *L'ensemble des fonctions partielles récursives peut effectivement être énuméré :*

1. *Pour tout n et e, φ_e^n est une fonction partielle récursive n-aire.*

2. *Pour toute fonction partielle récursive n-aire φ il existe un code e tel que $\varphi = \varphi_e^n$.*
3. *Il existe une fonction partielle récursive $n+1$-aire φ telle que pour tout e et $\vec{x}$,*

$$\varphi(e, \vec{x}) = \varphi_e^n(\vec{x}).$$

Preuve. Le dernier point découle du théorème de forme normale 2.4.1 : on définit simplement $\varphi(e, \vec{x}) = U\big(\mu y\, T_n(e, \vec{x}, y)\big)$. □

Le théorème d'énumération aborde un aspect très important de notre sujet : le double rôle que jouent les nombres. D'un côté, un nombre peut être l'entrée d'une fonction, d'un autre côté, un nombre peut être le code d'une fonction. La frontière entre les fonctions et les entrées de fonctions est rendue floue, ce qui ouvre la possibilité de l'autoréférence : un nombre peut être appliqué à lui-même. On gardera ce fait en tête, car il aura des conséquences importantes un peu plus loin.

Lemme 2.4.6 (Lemme de remplissage) *À partir d'un code e d'une fonction partielle récursive φ, on peut énumérer une infinité de code pour φ.*

Preuve. On peut en effet ajouter des règles factices au programme décrit par e ; règles qui ne changent pas le comportement de la fonction mais qui augmenteront la valeur du code. □

Le théorème suivant donne un moyen de faire passer une partie des arguments d'une fonction dans son programme.

Théorème 2.4.7 (Théorème *S-m-n*) *Pour tout m et n il existe une fonction primitive récursive $m+1$-aire injective S_n^m telle que*

$$\{S_n^m(e, x_1, \ldots, x_m)\}(y_1, \ldots, y_n) = \{e\}(x_1, \ldots, x_m, y_1, \ldots, y_n).$$

Preuve. On utilise l'arithmétisation des fonctions partielles récursives de la preuve du théorème 2.4.1. On remarque qu'il existe une fonction récursive primitive f qui, sur une entrée c, renvoie un code pour $\lambda y_1 \ldots y_n.c$. (voir exercice 2.5.4). On définit donc

$$S_n^m(e, x_1, \ldots, x_m) = \big\langle 3, f(x_1), \ldots, f(x_m), \langle 2, n, 1\rangle, \ldots, \langle 2, n, n\rangle, e\big\rangle.$$

À partir du codage de la preuve du théorème 2.4.1 on voit qu'il s'agit d'un code de la composition de la fonction $\lambda x_1 \ldots x_m y_1 \ldots y_n.\{e\}(x_1, \ldots, x_m, y_1, \ldots, y_n)$ avec les fonctions $\lambda y_1 \ldots y_n.x_i$, pour $i = 1 \ldots m$, et $\pi_n^j = \lambda y_1 \ldots y_n.y_j$, pour $j = 1 \ldots n$. Il est clair que cette composition est une fonction d'arguments $y_1, \ldots, y_n$ qui calcule $\{e\}(x_1, \ldots, x_m, y_1, \ldots, y_n)$. □

2.4.4 Codage canonique des ensembles finis

Il sera souvent utile de se référer aux ensembles finis d'une manière canonique. Pour ce faire, on définit un codage canonique comme suit : si $A = \{x_1, \ldots, x_n\}$ on affecte à A le code $e = 2^{x_1} + \ldots + 2^{x_n}$. On écrit alors $A = D_e$ et appelle e le *code canonique* de A. Par convention, on pose $D_0 = \emptyset$.

Il faut remarquer que ce codage est différent du codage des suites finies de la section 2.4.1. La principale différence est que, dans une suite, l'ordre des éléments et leurs multiplicités importent, tandis que ce n'est pas le cas dans un ensemble. De plus, si l'on écrit le code canonique e en binaire, on peut l'interpréter comme la chaîne caractéristique de D_e. Ce codage est donc cohérent avec notre convention d'identifier les ensembles avec leurs fonctions caractéristiques, comme précisé à la section 1.1.

2.5 Exercices

Exercice 2.5.1 Fibonacci découvrit la suite qui porte son nom en considérant le problème suivant. Supposons qu'une paire de lapins nouveau-nés prennent un mois pour être matures, et que chaque paire mature de lapins produit chaque mois une nouvelle paire. En partant d'une paire de lapins nouveau-nés, combien y aura-t-il de paires après n mois ?

Exercice 2.5.2 Soit $F(n)$ le n-ième nombre de Fibonacci. Soit

$$G(n) = \left(\frac{1+\sqrt{5}}{2}\right)^{n+1} - \left(\frac{1-\sqrt{5}}{2}\right)^{n+1}.$$

1. Exprimez $G(n)$ en fonction des solutions de l'équation $x^2 = x + 1$.
2. Utilisez la première question pour montrer que $\frac{\sqrt{5}}{5}G(n) = F(n)$.

Exercice 2.5.3 Montrez avec rigueur en utilisant la définition 2.2.2 que l'addition $+$ et la multiplication $\cdot$ sont récursives primitives.

Exercice 2.5.4 Montrez que les fonctions suivantes sont toutes récursives primitives.

1. Les fonctions constantes : pour toute arité n et toute constante $c \in \omega$, la fonction $\lambda x_1 \dots x_n.c$.
2. La fonction de signe :
$$sg(x) = \begin{cases} 1 & \text{si } x > 0, \\ 0 & \text{si } x = 0. \end{cases}$$
3. La distinction de cas :
$$cases(x, y, z) = \begin{cases} x & \text{si } z = 0, \\ y & \text{si } z > 0. \end{cases}$$
4. La soustraction limitée : $x \dot{-} y$, qui prend la valeur 0 si $y \geq x$ et $x - y$ sinon. Indice : définissez d'abord la fonction prédécesseur
$$\begin{aligned} pd(0) &= 0 \\ pd(x+1) &= x. \end{aligned}$$
5. La fonction caractéristique $\chi_=$ de la relation d'égalité.
6. Les sommes bornées : à partir d'une fonction récursive primitive f, $\sum_{y \leq z} f(\vec{x}, y)$.
7. Les produits bornés : à partir d'une fonction récursive primitive f, $\prod_{y \leq z} f(\vec{x}, y)$.
8. La recherche bornée : à partir d'une fonction récursive primitive f, la fonction $\mu y \leq z\big(f(\vec{x}, y) = 0\big)$ qui renvoie le plus petit $y \leq z$ tel que $f(\vec{x}, y) = 0$, ou 0 si un tel y n'existe pas.
9. Le maximum borné : à partir d'un prédicat récursif primitif R, la fonction $\max_{x \leq y} R(x, y)$.
10. Les quantifications bornées : à partir d'une relation récursive primitive R, les relations $P(x) = \exists y \leq x\ R(y)$ et $Q(x) = \forall y \leq x\ R(y)$.

11. La division : la relation $x|y$, x divise y.
12. Les nombres premiers : la fonction $n \mapsto p_n$, où p_n est le n-ième nombre premier, commençant par $p_0 = 2$.

Exercice 2.5.5 Montrez que la fonction $exp(y, k)$ de la page 11 qui donne l'exposant de k dans la décomposition de y est récursive primitive.

Exercice 2.5.6 (Concaténation de chaîne) Montrez qu'il existe une opération récursive primitive F telle que pour toute suite $\sigma = \langle x_1, \ldots, x_m \rangle$ et $\tau = \langle y_1, \ldots, y_n \rangle$,

$$F(\sigma, \tau) = \sigma^\frown\tau = \langle x_1, \ldots, x_m, y_1, \ldots, y_n \rangle.$$

Exercice 2.5.7 Terminez la preuve du théorème 2.3.5. En particulier :

1. Donnez un programme soigneux pour la fonction constante O.
2. Donnez un programme soigneux pour la fonction successeur S.
3. Donnez un programme soigneux pour les fonctions de projection π^i_n.
4. À partir de programmes soigneux pour les fonctions χ, ψ_1 et ψ_2, donnez un programme soigneux pour $\lambda x.\chi(\psi_1(x), \psi_2(x))$.
5. À partir de programmes soigneux pour les fonctions ψ et χ, donnez un programme soigneux pour $\varphi(x, y)$ définie par récursion primitive à partir de ψ et χ.
6. À partir d'un programme soigneux pour ψ, donnez un programme soigneux pour $\varphi(x) = \mu y(\psi(x, y) = 0)$.

Exercice 2.5.8 Terminez la preuve du théorème 2.4.1 en complétant les définitions des prédicats récursifs primitifs C, D et E de la page 12.

Exercice 2.5.9 Le schéma d'induction complète est

$$\begin{aligned} f(\vec{x}, 0) &= g(\vec{x}) \\ f(\vec{x}, n+1) &= h\big(\langle f(\vec{x}, 0), \ldots, f(\vec{x}, n)\rangle, \vec{x}, n\big). \end{aligned}$$

Montrez, en utilisant un codage des suites, que si g et h sont récursives primitives, alors f l'est aussi.

Exercice 2.5.10 (Récursion croisée) Soient f_1 et f_2 définies par

$$\begin{aligned} f_1(0) &= g_1(0) & \qquad f_2(0) &= g_2(0) \\ f_1(n+1) &= h_1\big(f_1(n), f_2(n), n\big) & \qquad f_2(n+1) &= h_2\big(f_1(n), f_2(n), n\big). \end{aligned}$$

Montrez que si g_1, g_2, h_1, h_2 sont toutes récursives primitives, alors f_1 et f_2 le sont aussi. Indice : utilisez un codage des paires pour faire une seule fonction de f_1 et f_2.

Exercice 2.5.11 (Péter) Montrez qu'il existe une fonction récursive qui énumère toutes les fonctions récursives primitives. C'est-à-dire, construisez une fonction récursive $\lambda e, x.f(e, x)$ telle que pour tout e, la fonction $\lambda x.f(e, x)$ est récursive primitive, et que réciproquement, toute fonction récursive primitive est égale à $\lambda x.f(e, x)$ pour un certain e.

Exercice 2.5.12 Montrez qu'il existe une fonction récursive (totale) qui n'est pas récursive primitive. Indice : utilisez l'exercice 2.5.11 et la discussion du début de la section 2.2.2.

Exercice 2.5.13 La fonction *busy beaver* (castor affairé) $bb(n)$ est définie par l'énoncé suivant : à partir d'un n, considérez toutes les machines de Turing avec un nombre d'états $\leq n$ qui sont définies sur l'entrée 0. Alors $bb(n)$ est la sortie la plus grande de toutes les machines. Montrez que bb n'est pas récursive.

Exercice 2.5.14 Soient g une fonction partielle récursive et R un prédicat récursif. Montrez que la fonction

$$\psi(x) = \begin{cases} g(x) & \text{si } \exists y\ R(x,y), \\ \uparrow & \text{sinon.} \end{cases}$$

est partielle récursive (cette forme de définition, la définition par cas, sera utilisée tout au long de ce document).

Exercice 2.5.15 Montrez ou réfutez que si f et g ne sont pas récursives, alors la fonction $\lambda n.f(n) + g(n)$ n'est pas récursive.

Exercice 2.5.16 Montrez qu'il existe une fonction f récursive (et même primitive) telle que pour tout n et x,

$$\{f(n)\}(x) = \begin{cases} 0 & \text{si } x \in D_n, \\ \uparrow & \text{sinon.} \end{cases}$$

Exercice 2.5.17 Supposons l'existence d'un ensemble r.é. non récursif A (un exemple sera donné dans le chapitre suivant). On définit

$$\psi(x,y) = 0 \iff (y = 0 \wedge x \in A) \vee y = 1$$

et

$$f(x) = (\mu y)\big[\psi(x,y) = 0\big].$$

Montrez que

1. ψ est partielle récursive.
2. $f(x) = 0 \iff x \in A$.
3. f n'est pas partielle récursive.

Cet exercice montre que l'on ne peut pas simplifier le schéma de μ-récursion de la définition 2.2.3 avec $\varphi(x) = (\mu y)\big[\psi(x,y) = 0\big]$.

Chapitre 3

Ensembles récursifs et récursivement énumérables

3.1 Diagonalisation

La méthode de diagonalisation a été inventée par Cantor pour montrer que l'ensemble des nombres réels, ou de manière équivalente l'espace de Cantor 2^ω, est indénombrable. Cette méthode est d'une importance fondamentale dans toute la logique mathématique. Un argument diagonal prend souvent la forme suivante. À partir d'une liste dénombrable d'objets A_i, $i \in \omega$, on souhaite construire un nouvel objet A, distinct de tous les A_i. L'objet A est construit par étapes, où à la i-ième on s'assure que $A \neq A_i$. Dans l'exemple de Cantor, les A_i sont des listes d'éléments de 2^ω, et à l'étape i, le i-ième élément de A est défini comme étant $1 - A_i(i)$, assurant ainsi que A soit différent de A_i. L'ensemble A est alors appelé *ensemble diagonal*, car il est défini en utilisant la diagonale de la matrice infinie $\{A_i(j)\}_{i,j\in\omega}$. L'argument montre qu'aucune liste dénombrable ne peut contenir tous les éléments de 2^ω.

Un corollaire immédiat du théorème 2.4.1 est que l'ensemble des fonctions récursives est dénombrable, puisqu'elles peuvent être codées avec un ensemble dénombrable de codes. À partir du fait que 2^ω est indénombrable, on infère immédiatement *qu'il y a un nombre indénombrable de fonctions non récursives*. L'argument informel justifiant les fonctions partielles du début de la section 2.2.2 était en fait notre premier exemple de diagonalisation. La fonction $F = \lambda n.f_n(n) + 1$ définie alors est donc un autre exemple d'objet construit par diagonalisation. Par ailleurs, les objets diagonaux comme F ont une saveur autoréférente puisque n se comporte comme un objet standard (en son rôle d'argument) et à un niveau méta (en tant que position du n-ième objet de la liste). Il nous a déjà été donné de croiser ces autoréférences page 15, avec le double rôle des nombres dans la théorie de la calculabilité. Dans ce chapitre, nous développons la base de la théorie de la récursivité, ou du calculable, et des ensembles récursivement énumérables. L'un des premiers résultats (théorème 3.3.1) fera apparaître les deux types d'autoréférence que nous avons vus.

3.2 Ensembles récursivement énumérables

Un ensemble récursivement énumérable est un ensemble dont les éléments peuvent être effectivement listés :

Définition 3.2.1 Un ensemble A est *récursivement énumérable (r.é.)* s'il existe une fonction f récursive telle que $A = \mathrm{Img}(f)$, où $\mathrm{Img}(f) = \{y : (\exists x)[f(x) = y]\}$ est

l'image de f. Pour des raisons de cohérence, l'ensemble vide $\emptyset$ sera considéré comme étant récursivement énumérable.

Les ensembles r.é. sont des analogues aux ensembles de Cantor dénombrables. Ils peuvent aussi être vus comme l'analogue pour les ensembles de la notion de fonction partielle récursive, tout comme la notion d'ensemble récursif est l'analogue de la notion de fonction récursive. Une part de l'intérêt porté aux ensembles r.é. est due à leur abondance en mathématiques et informatique. Par exemple, les ensembles suivants sont r.é. :

- L'ensemble de tous les théorèmes (codés par des entiers naturels) de n'importe quel système formel doté d'un ensemble récursif d'axiomes.
- L'ensemble des équations diophantiennes qui ont une solution entière.
- L'ensemble des mots qui peuvent être produits à partir d'une grammaire formelle (dans la théorie des langages formels).

Beaucoup d'autres exemples peuvent être tirés de l'informatique, le plus connu étant le suivant :

Définition 3.2.2 Le *problème de l'arrêt* est l'ensemble

$$H = \big\{\langle x, y\rangle : \varphi_x(y)\!\downarrow\big\}.$$

Le *problème de l'arrêt diagonal* est l'ensemble

$$K = \big\{x : \varphi_x(x)\!\downarrow\big\}.$$

Le problème de l'arrêt est l'ensemble représentant le problème de décider si un calcul donné aboutit. On vérifie que les deux ensembles H et K sont en effet récursivement énumérables. Soit $m \in H$ un élément fixé, H est alors l'image de la fonction récursive suivante :

$$f(n) = \begin{cases} \langle (n)_0, (n)_1\rangle & \text{si } \varphi_{(n)_0,(n)_2}((n)_1)\!\downarrow, \\ m & \text{sinon.} \end{cases}$$

Donc f sur l'entrée $n = \langle x, y, s\rangle$ vérifie que le calcul $\varphi_x(y)$ converge en s étapes et, si c'est le cas, retourne $\langle x, y\rangle$. On utilise m pour rendre f totale. Puisque pour tout calcul convergent $\varphi_x(y)$ il y a un s tel que $\varphi_{x,s}(y)\!\downarrow$, toute paire de la forme $\langle x, y\rangle$ sera dans l'image de f.[1] Le fait que K soit lui aussi r.é. s'obtient d'une manière similaire.

Avant de continuer cette discussion, nous faisons quelques observations générales. Dans un premier temps, posons des manières équivalentes de définir les ensembles r.é. :

Théorème 3.2.3 *Pour tout ensemble A, les propriétés suivantes sont équivalentes :*

(i) *A est r.é.*

(ii) *$A = \mathrm{Img}(\varphi)$ pour une fonction partielle récursive φ.*

(iii) *$A = \mathrm{Dom}(\varphi) = \big\{x : \varphi(x)\!\downarrow\big\}$ pour une fonction partielle récursive φ.*

(iv) *$A = \big\{x : (\exists y)[R(x, y)]\big\}$ pour un prédicat récursif binaire R.*

Preuve. (i)$\Longrightarrow$(ii). Clairement, si A est l'image d'une fonction totale récursive, c'est aussi l'image d'une fonction partielle.

[1] Il faut bien noter que f est totale. De fait, même si l'entrée n n'est pas un code correct, $f(n)$ est définie car la fonction $(x)_i$ est définie pour tout x et i, cf. exercice 2.5.5.

(ii)$\Longrightarrow$(iii). Supposons que $A = \mathrm{Img}(\varphi_e)$. On définit la fonction partielle récursive ψ par

$$\psi(y) = \begin{cases} 0 & \text{si } \exists x \exists s\ \varphi_{e,s}(x) = y, \\ \uparrow & \text{sinon.} \end{cases}$$

On a alors $\mathrm{Img}(\varphi) = \mathrm{Dom}(\psi)$. On notera que cette forme de définition est autorisée par l'exercice 2.5.14.
(iii)$\Longrightarrow$(iv). Il suffit de remarquer que $\mathrm{Dom}(\varphi_e) = \{x : (\exists s)[\varphi_{e,s}(x) \downarrow]\}$ a la forme requise.
(iv)$\Longrightarrow$(i). Soit A comme dans la propriété (iv) et supposons que $A \neq \emptyset$, et soit $a \in A$. On définit

$$f(n) = \begin{cases} (n)_0 & \text{si } R((n)_0, (n)_1), \\ a & \text{sinon.} \end{cases}$$

Alors f est totale et l'on a bien $A = \mathrm{Img}(f)$. □

Le théorème 3.2.3 montre que l'on peut aussi définir les ensembles r.é. comme les domaines de fonctions partielles récursives. Il est donc possible d'associer des codes aux ensembles r.é. de manière canonique comme cela a été fait pour les fonctions partielles récursives :

Définition 3.2.4 On définit

$$W_e = \mathrm{Dom}(\varphi_e)$$

que l'on appellera le e-ième ensemble r.é. Le nombre e est appelé un *code r.é.* ou plus simplement un *code* de W_e. On définit de la même manière

$$W_{e,s} = \mathrm{Dom}(\varphi_{e,s}).$$

On remarque que les ensembles $W_{e,s}$ sont des approximations finies de W_e et que l'on a $W_e = \bigcup_{s\in\omega} W_{e,s}$. De plus, par définition de $\varphi_{e,s}$ il y a au plus un élément dans $W_{e,s+1} - W_{e,s}$ pour tout s.[2] Nous utiliserons ces faits implicitement dans la suite.

Rappelons la définition du problème de l'arrêt $H = \{\langle x, y\rangle : \varphi_x(y) \downarrow\}$. On remarque que pour toute paire $\langle x, y\rangle$ on a $H(\langle x, y\rangle) = W_x(y)$. De ce point de vue, H code les informations de tous les autres ensembles r.é. : il est donc appelé ensemble *universel*.

Par définition, les ensembles r.é. sont les images des fonctions récursives. On peut caractériser les ensembles récursifs de manière équivalente :

Proposition 3.2.5 *Les propriétés suivantes sont équivalentes :*

(i) *A est récursif.*

(ii) *$A = \emptyset$ ou $A = \mathrm{Img}(f)$ pour une fonction récursive f non décroissante.*

Preuve. Supposons que A soit récursif et non vide. On définit f par

$$f(0) = \mu n.\ n \in A$$
$$f(n+1) = \begin{cases} n+1 & \text{si } n+1 \in A, \\ f(n) & \text{sinon.} \end{cases}$$

Alors f est récursive et a pour image A.

Réciproquement, si f est récursive et non décroissante il y a deux cas :

[2] On se rappelle ici que s est une borne sur le calcul complet de $\{e\}$, incluant l'entrée. Ceci en particulier pour les fonctions qui ne lisent pas leur entrée, telles que les fonctions constantes.

Cas 1 : $\mathrm{Img}(f)$ est fini, donc $\mathrm{Img}(f)$ est bien évidemment récursif.
Cas 2 : $\mathrm{Img}(f)$ est infini. On a alors que $n \in \mathrm{Img}(f)$ si et seulement si $n = f(m)$, où m est le plus petit entier tel que $f(m) \geq n$. Un tel m existe toujours puisque l'image de f est infinie. Donc ici aussi $\mathrm{Img}(f)$ est récursif. □

La notion d'appartenance d'un ensemble r.é. A est asymétrique : si $n \in A$, cela peut être vérifié en un nombre fini d'étapes, mais si $n \notin A$ on peut ne jamais s'en rendre compte. Le résultat élémentaire suivant, dû à Post, montre en effet que les ensembles r.é. sont d'une certaine manière « à moitié récursifs ».

Proposition 3.2.6 *Un ensemble A est récursif si et seulement si à la fois A et son complémentaire $\overline{A}$ sont r.é.*

Preuve. De manière informelle, pour décider si $x \in A$ il suffit d'énumérer A et $\overline{A}$ jusqu'à ce que x apparaisse dans l'un d'eux. Pour une preuve formelle, voir l'exercice 3.8.1. □

3.3 Ensembles indécidables

Nous avons nommé récursif un ensemble A dont la fonction caractéristique χ_a était récursive (définition 2.3.1). Si l'on est intéressé, pour un ensemble donné A, par décider pour quels n on a $n \in A$, on dira que A est un *problème de décision*. Un problème de décision A qui est récursif sera dit *décidable*. On remarque que H est un problème de décision.

Théorème 3.3.1 (Indécidabilité du problème de l'arrêt, Turing [29, p. 3]) *H est indécidable.*

Preuve. Si H était décidable, alors K le serait aussi puisque $x \in K \Leftrightarrow \langle x, x\rangle \in H$. Il suffit donc de montrer que K est indécidable. Supposons que K est décidable, son complémentaire $\overline{K}$ est donc décidable, et en particulier r.é., il existe donc un e tel que $\overline{K} = W_e$. On a alors

$$e \in K \iff \varphi_e(e)\downarrow \iff e \in W_e \iff e \in \overline{K},$$

ceci étant bien une contradiction. □

Le théorème 3.3.1 montre *qu'il existe des ensembles r.é. qui ne sont pas récursifs*. Cette preuve de l'indécidabilité de K est un autre exemple de la méthode de diagonalisation.

Il est naturel de requérir d'un ensemble de codes qu'il soit clos par équivalence fonctionnelle de ces derniers :

Définition 3.3.2 Un ensemble A est appelé *ensemble d'indices* si

$$(\forall d, e)[d \in A \wedge \varphi_e = \varphi_d \rightarrow e \in A].$$

Tous les ensembles de codes définis par des propriétés sur les fonctions (plutôt que par des codes) sont des ensembles d'indices. Par exemple, les ensembles suivants sont tous des ensembles d'indices :

- $\mathrm{Fin} = \{e : W_e \text{ est fini}\}$.
- $\mathrm{Tot} = \{e : \varphi_e \text{ est totale}\}$.
- $\mathrm{Rec} = \{e : W_e \text{ est récursif}\}$.

Ces exemples ont leur importance et nous aurons l'occasion de les traiter plus en profondeur au chapitre 4.

Après avoir vu que K était indécidable, il n'est pas surprenant que Fin, Tot et Rec soient eux aussi indécidables, puisque décider si un certain e est membre de l'un de ces ensembles semble plus difficile que de savoir seulement s'il converge pour une entrée donnée. En fait, l'indécidabilité de ces ensembles découle d'un résultat bien plus général : quasiment *tous* les ensembles d'indices sont indécidables :

Théorème 3.3.3 (Théorème de Rice) *Soit A un ensemble d'indices tel que $A \neq \emptyset$ et $A \neq \omega$. Alors A est indécidable.*

Preuve. Soit A comme dans l'énoncé. Soit e un code pour la fonction définie nulle part et supposons que $e \in A$. Soit $d \in \overline{A}$ et f une fonction récursive telle que

$$\varphi_{f(x)} = \begin{cases} \varphi_d & \text{si } x \in K, \\ \uparrow & \text{sinon.} \end{cases}$$

On a alors

$$\begin{array}{rcccl} x \in K & \Longrightarrow & \varphi_{f(x)} = \varphi_d & \Longrightarrow & f(x) \notin A, \\ x \notin K & \Longrightarrow & \varphi_{f(x)} = \varphi_e & \Longrightarrow & f(x) \in A. \end{array}$$

De fait, si A était décidable, K le serait aussi, ce qui est impossible par le théorème 3.3.1. Le cas $e \notin A$ est complètement symétrique en choisissant un $d \in A$. □

Le théorème de Rice énonce que les seuls ensembles d'indices décidables sont $\emptyset$ et ω. On voit donc que l'indécidabilité est en effet omniprésente en informatique.

La proposition 3.2.6 énonçait qu'un ensemble A est récursif si A et $\overline{A}$ étaient r.é. On considère maintenant le cas plus général des paires d'ensembles r.é. disjoints :

Définition 3.3.4 Deux ensembles r.é. disjoints A et B sont dits *récursivement séparables* s'il existe un ensemble récursif C tel que $A \subseteq C$ et $B \subseteq \overline{C}$. A et B sont dits *récursivement inséparables* s'ils ne sont pas récursivement séparables.

Il est à remarquer que deux ensembles récursivement inséparables ne peuvent être récursifs. Le théorème suivant est une forme forte de l'existence d'ensembles r.é. non récursifs.

Théorème 3.3.5 *Il existe une paire d'ensembles r.é. qui sont récursivement inséparables*

Preuve. On définit

$$\begin{aligned} A &= \big\{x : \varphi_x(x)\!\downarrow = 0\big\}, \\ B &= \big\{x : \varphi_x(x)\!\downarrow = 1\big\}. \end{aligned}$$

On suppose qu'il existe un ensemble récursif C séparant A et B. Soit e un code de la fonction caractéristique de C : $\varphi_e = \chi_C$. On obtient alors une contradiction en vérifiant si $e \in C$:

$$\begin{array}{rcccccl} e \in C & \Longrightarrow & \varphi_e(e) = 1 & \Longrightarrow & e \in B & \Longrightarrow & e \notin C, \\ e \notin C & \Longrightarrow & \varphi_e(e) = 0 & \Longrightarrow & e \in A & \Longrightarrow & e \in C. \end{array}$$ □

Les paires d'ensembles récursivement inséparables sont des objets naturels : il peut être montré que pour un système formel $\mathcal{F}$ assez puissant (pouvant au moins exprimer l'arithmétique), l'ensemble des formules que l'on peut montrer vraies et l'ensemble des formules que l'on peut montrer fausses sont récursivement inséparables. La preuve consiste juste en la formalisation de la preuve ci-dessus dans $\mathcal{F}$.

3.4 Uniformité

Le théorème 3.2.3 montrait que plusieurs façons de définir les ensembles r.é. sont équivalentes. Mais il s'avère que certaines de ces équivalences sont plus fortes qu'il n'y parait, comme on va le voir en regardant la preuve de plus près.

Considérons par exemple le sens (ii)$\Longrightarrow$(iii). La fonction ψ a été définie de manière récursive à partir de la fonction φ, ce qui signifie qu'il existe une fonction récursive h telle que si e est un code pour φ, $h(e)$ est un code pour ψ. On dira que l'implication (ii)$\Longrightarrow$(iii) est vraie de manière *uniforme (en les codes)*. On peut montrer que les équivalences de (ii), (iii) et (iv) sont toutes uniformes en les codes (cf. exercice 3.8.2).

En ce qui concerne l'implication (iv)$\Longrightarrow$(i), on a distingué dans la preuve les cas $A = \emptyset$ et $A \neq \emptyset$. Or l'ensemble

$$\{e : W_e = \emptyset\}$$

est un ensemble d'indices, et par le théorème 3.3.3, il est indécidable. Donc on ne peut pas faire une distinction effective entre les deux cas. On a pourtant une forme restreinte d'uniformité :

Proposition 3.4.1 *Il existe une fonction récursive f telle que pour tout e,*

$$W_e \neq \emptyset \Longrightarrow \varphi_{f(e)} \text{ totale } \wedge \operatorname{Img}(\varphi_{f(e)}) = W_e.$$

Preuve. Soit d un code tel que

$$\varphi_d(e,s) = \begin{cases} m & \text{si } m \in W_{e,s+1} - W_{e,s}, \\ \mu n.\ n \in W_e & \text{sinon.} \end{cases}$$

On définit alors $f(e) = S_1^1(d,e)$. La fonction f est alors totale, puisque S_1^1 l'est, et quand W_e est non vide, $\varphi_{f(e)}$ est totale et d'image W_e. □

On va maintenant considérer un exemple de résultat qui n'est pas uniformément vrai. On remarque dans un premier temps qu'il y a deux façons de décrire un ensemble fini A :

- Comme un D_n pour un certain code canonique n (voir section 2.4.4) : décrire A revient alors à donner une liste complète de ses éléments,
- Comme un W_e pour un certain code r.é. e, ce qui revient à donner une énumération de ses éléments.

Proposition 3.4.2 *On peut passer effectivement des codes canoniques aux codes r.é., mais pas réciproquement.*

Preuve. L'exercice 2.5.16 nous permet de dire qu'il existe une fonction récursive f telle que $W_{f(n)} = D_n$ pour tout n, donc on peut effectivement passer des codes canoniques aux codes r.é.

Supposons maintenant que l'on puisse faire le contraire, c'est-à-dire que l'on ait une fonction partielle récursive ψ telle que

$$W_e \text{ fini } \Longrightarrow \psi(e)\downarrow \wedge\, W_e = D_{\psi(e)}.$$

On définit alors une fonction récursive g par

$$W_{g(e)} = \begin{cases} \{e\} & \text{si } e \in K, \\ \emptyset & \text{sinon.} \end{cases}$$

Alors $W_{g(e)}$ est toujours fini, donc $\psi(g(e))\downarrow$ pour tout e. Mais on a alors

$$e \in K \iff W_{g(e)} \neq \emptyset \iff D_{\psi(g(e))} \neq \emptyset.$$

Or $D_n = \emptyset$ est décidable, ce qui implique que K l'est aussi ; contradiction. Donc ψ n'existe pas. □

Dans la proposition 3.7.2, on utilisera le théorème de récursion pour montrer que la proposition 3.2.5 n'est pas uniformément vraie.

3.5 Réduction *many-one*

Quand on considère des problèmes de décision, ou les problèmes plus généralement, il est utile d'avoir un moyen de réduire un problème à un autre. Si l'on peut résoudre le problème A et que le problème B se réduit à A, alors il est possible de résoudre B. Dans l'autre sens, si l'on sait que B n'est pas solvable, alors A ne peut l'être. Dans la suite, on rencontrera plusieurs notions de réduction. La réduction la plus simple entre problèmes de décision est probablement celle qui consiste à transformer effectivement une question de la forme « $n \in A$? » en « $f(n) \in B$? » ; la définition suivante formalise cette idée. Une notion de réduction bien plus générale sera étudiée au chapitre 5.

Définition 3.5.1 Un ensemble A se *réduit par réduction many-one*, ou se *m-réduit*, à un ensemble B, ce que l'on note $A \leq_m B$, s'il existe une fonction récursive f telle que

$$n \in A \iff f(n) \in B$$

pour tout n. La fonction f est alors appelée une réduction *many-one*, ou *m-réduction*. Cette dénomination provient du fait que f n'est pas forcément injective, donc elle peut réduire beaucoup de questions sur l'appartenance à A en une seule question dans B. On dit que A est *m-équivalent* à B, ce que l'on note $A \equiv_m B$, si l'on a à la fois $A \leq_m B$ et $B \leq_m A$. Il est clair que la relation $\equiv_m$ est une relation d'équivalence, et ses classes d'équivalences sont appelées degrés *many-one*, ou plus simplement *m-degrés*.

Remarquons tout d'abord que tous les ensembles récursifs (différents de $\emptyset$ et ω) ont le même m-degré, qui est d'ailleurs le *plus petit* m-degré, puisque les ensembles récursifs se m-réduisent à n'importe quel autre ensemble.

Dans la preuve du théorème 3.3.1 on a montré que H était indécidable en montrant que K l'était, et en m-réduisant K à H. De même, dans la preuve du théorème 3.3.3 de Rice, on a montré que A n'était pas récursif en construisant une m-réduction de $\overline{K}$ à A. On a en fait implicitement utilisé l'observation suivante :

Proposition 3.5.2 *Si B n'est pas récursif et $B \leq_m A$ alors A n'est pas non plus récursif.*

Preuve. Trivial. □

Proposition 3.5.3 *Un ensemble A est r.é. si et seulement si $A \leq_m K$.*

Preuve. (Si) Supposons que $x \in A \Leftrightarrow f(x) \in K$ pour une fonction f récursive. Alors $A = \{x : (\exists s)[f(x) \in K_s]\}$, où K_s est l'approximation finie à s étapes de K. Donc par le théorème 3.2.3, A est r.é.

(Seulement si) En utilisant le théorème S-m-n, on définit f récursif tel que

$$\varphi_{f(e,x)}(z) = \begin{cases} 0 & \text{si } x \in W_e, \\ \uparrow & \text{sinon.} \end{cases}$$

Alors pour tout e et x, $x \in W_e \Leftrightarrow \varphi_{f(e,x)}(f(e,x))\downarrow \Leftrightarrow f(e,x) \in K$. En particulier, $W_e \leq_m K$ pour tout e. □

Un élément d'une classe pour lequel tout autre élément de la classe se réduit à lui (selon une certaine notion de réduction) est dit *complet*. La proposition 3.5.3 montre que K est *m-complet* pour les ensembles r.é., donc le problème de l'arrêt H est aussi m-complet. C'est un fait brièvement aperçu en page 21 quand a été abordé son universalité.

Par un argument de comptage, on peut quantifier les m-degrés :

Proposition 3.5.4 *Il y a $2^{\aleph_0}$ différents m-degrés.*

Preuve. On remarque que pour tout A, il y a seulement $\aleph_0$ ensembles B tels que $B \leq_m A$ puisqu'il n'y a qu'une quantité dénombrable de fonctions récursives pouvant jouer le rôle de m-réduction. En particulier, tout m-degré contient seulement $\aleph_0$ ensembles (cf. aussi exercice 3.8.10). Puisqu'il y a $2^{\aleph_0}$ ensembles, le résultat suit. □

3.6 Ensembles simples

Jusqu'à présent, nous avons vu deux types d'ensemble r.é. : les ensembles récursifs et les ensembles m-complets. Il est naturel de se demander s'il existe d'autre types d'ensembles, c'est-à-dire s'il existe des ensembles r.é. qui ne sont ni récursifs ni m-complets. L'existence de ces ensembles a été montrée par Post en considérant des ensembles qui avaient des complémentaires « minces », de telle sorte qu'il ne soit pas possible de prendre effectivement des sous-ensembles infinis de leurs compléments. On remarque tout d'abord qu'aucun ensemble r.é. n'est mince dans ce sens :

Proposition 3.6.1 (Post [20]) *Tout ensemble r.é. infini contient un sous-ensemble infini récursif.*

Preuve. Voir l'exercice 3.8.5 □

Définition 3.6.2 Un ensemble est *immun* s'il est infini et qu'il ne contient pas de sous-ensemble r.é. infini. Un ensemble est *simple* s'il est r.é. et que son complément est immun.

Il est immédiat, à partir de la définition, de voir que les ensembles simples ne sont pas récursifs. Il ne sont pas non plus complets :

Proposition 3.6.3 (Post [20]) *Si un ensemble est m-complet, il ne peut pas être simple.*

Preuve. On prouve tout d'abord que K n'est pas simple. On remarque que pour tout x on a

$$W_x \subseteq \overline{K} \Longrightarrow x \in \overline{K} - W_x \tag{3.1}$$

car si $x \in W_x$ alors $x \in K$, et donc $W_x \not\subseteq \overline{K}$. On peut alors utiliser (3.1) pour générer un sous-ensemble r.é. infini de $\overline{K}$ de la manière suivante. Soit f une fonction récursive telle que pour tout x

$$W_{f(x)} = W_x \cup \{x\}.$$

Il suffit alors de partir d'un code x tel que $W_x = \emptyset$ et d'appliquer plusieurs fois f pour obtenir un ensemble r.é. infini $V = \{x, f(x), f(f(x)), \ldots, f^{(n)}(x), \ldots\}$. Avec (3.1), $V \subseteq \overline{K}$ et V est infini. Donc $\overline{K}$ contient un ensemble r.é. infini et n'est donc pas immun.

On montre maintenant que si $K \leq_m A$, alors le fait que K ne soit pas simple est une propriété transmise à A. Supposons que $K \leq_m A$ par h. Soit U un sous-ensemble r.é. de $\overline{A}$. Son image inverse par h, $h^{-1}(U)$ est un ensemble r.é. de $\overline{K}$.

De plus, on peut effectivement obtenir un code pour $h^{-1}(U)$ à partir du code de U. En utilisant la procédure pour K ci-dessus, on obtient un élément $x \in \overline{K} - h^{-1}(U)$, et donc $h(x) \in \overline{A} - U$. En itérant, on obtient un sous-ensemble r.é. infini de $\overline{A}$.

Formellement, soit $g(0) = h(x)$ où $W_x = \emptyset$. On a $g(0) \in \overline{A}$ par (3.1). Avec la donnée de $g(0), \ldots, g(n) \in \overline{A}$, soit $\xi(n)$ un code de l'ensemble r.é. $h^{-1}(\{g(0), \ldots, g(n)\})$. Alors, par (3.1), $\xi(n) \in \overline{K} - h^{-1}(\{g(0), \ldots, g(n)\})$ on définit donc $g(n+1) = h(\xi(n))$. Il est alors clair que $\mathrm{Img}(g)$ est un sous-ensemble r.é. infini de $\overline{A}$, de fait, A n'est pas simple. □

Théorème 3.6.4 (Post [20]) *Il existe des ensembles simples.*

Preuve. On veut construire un ensemble r.é. co-infini A tel que pour tout e, on ait la condition suivante :

$$C_e : \qquad W_e \text{ infini} \Longrightarrow A \cap W_e \neq \emptyset.$$

Bien que l'on sache énumérer les ensembles r.é. effectivement, on ne peut pas décider lesquels sont infinis, à cause des résultats de la section 3.3. Mais avec un W_e donné, on peut juste attendre et suivre son évolution, et ainsi énumérer un de ses éléments quand on en voit un assez grand. Plus précisément, pour conserver la propriété d'infinitude de $\overline{A}$, on veut s'assurer qu'au plus e éléments de $\{0, \ldots, 2e\}$ aillent dans A. Comme l'on peut satisfaire C_e en énumérant simplement un élément de W_e, il suffit de permettre à C_e d'énumérer un élément x seulement si $x > 2e$. Au final, on construit l'ensemble A par étapes comme suit :
Étape $s = 0$. Soit $A_0 = \emptyset$.
Étape $s + 1$. À cette étape, on dispose de l'approximation courante A_s de A. On recherche alors le plus petit $e < s$ tel que $A_s \cap W_{e,s} = \emptyset$ et tel qu'il existe un $x \in W_{e,s}$ avec $x > 2e$. Si de tels e et x sont trouvés, on énumère x dans A, c'est-à-dire que l'on fixe $A_{s+1} = A_s \cup \{x\}$. Sinon, on pose $A_{s+1} = A_s$. Ceci conclut la construction.

L'ensemble construit $A = \bigcup_{s \in \omega} A_s$ est alors co-infini car seulement la moitié des éléments plus petits que $2e$ peut être énumérée dans A. De plus, si W_e est infini, il contient un $x > 2e$, ce qui signifie qu'un élément de W_e est énuméré dans A à une certaine étape. De fait, la condition C_e est satisfaite pour tout e. □

La preuve du théorème 3.6.4 fait ressortir plusieurs aspects classiques d'argumentation dans la théorie de la calculabilité. Elle utilise tout d'abord une diagonalisation : la condition globale que l'ensemble ne doit être ni récursif ni complet est décomposée en une infinité de sous-conditions C_e. On construit alors en une infinité d'étapes un ensemble A qui satisfait ces conditions. Une différence importante avec les exemples de la section 3.1, comme l'argument de Cantor, est qu'ici il est impossible de prendre

en compte la condition C_e à l'étape e car *la construction doit être effective* pour rendre A r.é. De fait, au lieu de s'occuper des conditions dans l'ordre, on adopte une approche plus dynamique, en attendant qu'une action puisse être faite. On remarque aussi l'opposition entre les conditions C_e, qui veulent ajouter des éléments *dans* A, et la propriété de co-infinitude de A, qui essaie de maintenir les éléments *à l'extérieur* de A. Le conflit entre ces conditions a été facilement résolu ici en empêchant les C_e d'énumérer des nombres trop petits. En règle générale, le conflit entre les conditions dans une diagonalisation est bien plus compliqué à gérer, nécessitant l'emploi de techniques de résolution plus complexes.

En combinant les résultats de cette section, on obtient :

Corollaire 3.6.5 *Il existe un ensemble r.é. qui n'est ni récursif, ni m-complet.*

Dans la section 7.3, nous verrons un exemple naturel d'un de ces ensembles r.é. de m-degré intermédiaire.

3.7 Le théorème de récursion

Le théorème suivant est aussi appelé le théorème de point fixe. Il met en exergue une intrinsèque autoréférence, mais sa courte et simple preuve n'explique pas grand chose de ce mystère.

Théorème 3.7.1 (Théorème de récursion, Kleene [11]) *Pour toute fonction récursive f, il existe un entier e (appelé* point fixe *de f) tel que $\varphi_{f(e)} = \varphi_e$.*

Preuve. Par le théorème d'énumération 2.4.5.3 la fonction $\lambda x, z.\, \varphi_{f\circ\varphi_x(x)}(z)$ est récursive, elle a donc un code c. Par le théorème S-m-n, soit b un code tel que $\varphi_b(x) = S_1^1(c, x)$. Remarquons que φ_b est totale car S_1^1 est récursive primitive. On a alors :

$$\varphi_{\varphi_b(x)}(z) = \varphi_c(x, z) = \varphi_{f\circ\varphi_x(x)}(z),$$

de telle sorte que $\varphi_b(b)$ soit un point fixe de f. □

Pour mieux saisir la preuve du théorème 3.7.1, nous considérons le point de vue de Owings [19] qui proposa de regarder cette preuve comme un argument diagonal qui échoue. L'énoncé du théorème suggère que nous observions les fonctions de la forme $\varphi_{\varphi_n(m)}$ placées dans une matrice à deux dimensions, comme dans la figure 3.1. Posons que $\varphi_{\varphi_n(m)}$ dénote la fonction définie nulle part si $\varphi_n(m)\uparrow$. Considérons maintenant la diagonale $\varphi_{\varphi_x(x)}$ de cette matrice. Contrairement à la situation habituelle des arguments diagonaux, cette diagonale, comme fonction de x, est elle-même une ligne de la matrice. De plus, chaque fonction f définit une application entre lignes, envoyant la ligne $\varphi_{\varphi_x(y)}$ sur $\varphi_{f\circ\varphi_x(y)}$. Puisque toutes ces opérations sont effectives, il existe un code b tel que

$$\varphi_{\varphi_b(x)} = \varphi_{f\circ\varphi_x(x)}.$$

En prenant $x = b$, on voit que f a le point fixe $\varphi_b(b)$.

La saveur autoréférente du théorème 3.7.1 est due au fait que *l'on puisse définir une fonction à l'aide de son propre code.* Il est souvent utilisé de la manière suivante. On commence par définir un objet comme une fonction partielle récursive ou un ensemble r.é. A en utilisant un code arbitraire e. Si la définition de A dépend de e de manière effective (c'est-à-dire s'il existe une fonction récursive f telle que $A = W_{f(e)}$) alors

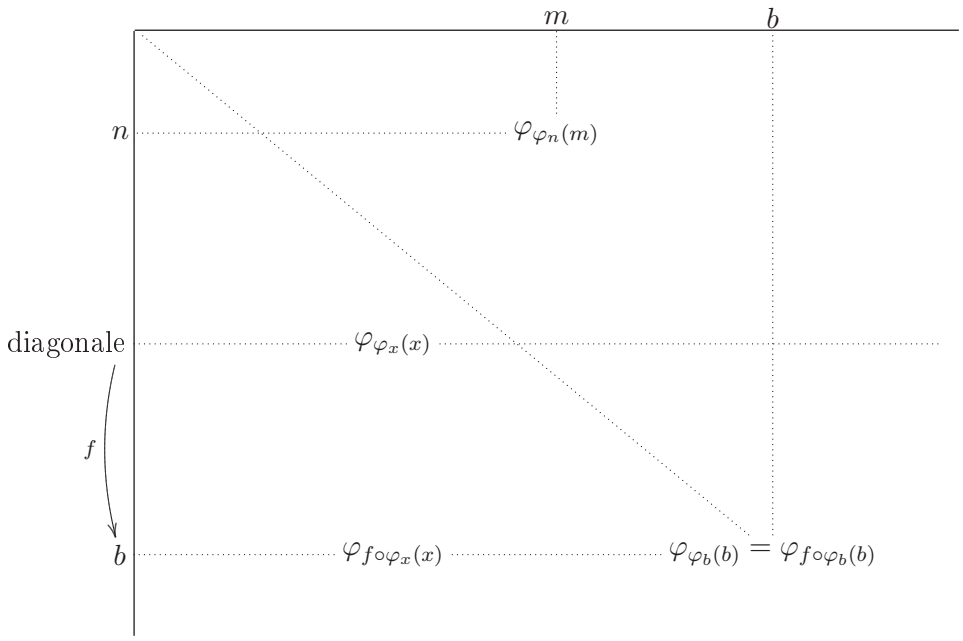


FIG. 3.1 – Matrice des fonctions $\varphi_{\varphi_n(m)}$

par le théorème 3.7.1, on peut conclure qu'il existe un code e tel que $A = W_e$. Cela justifie l'écriture de phrases comme « soit e un code tel que $W_e = A$ », où e apparaît lui-même de manière effective dans la définition de A.

Une conséquence du théorème 3.7.1 est l'existence de programmes qui s'affichent eux-mêmes quand ils sont exécutés. Formellement, par le théorème S-m-n 2.4.7, soit f une fonction récursive telle que, pour tout x, $\varphi_{f(e)}(x) = e$. Le théorème 3.7.1 nous dit alors qu'il existe un point fixe e pour lequel $\varphi_e(x) = e$ pour tout x, c'est-à-dire que le programme e s'affiche lui-même sur n'importe quelle entrée.

Un autre exemple paradoxal : le théorème 3.7.1 implique l'existence d'un e tel que $\varphi_e(x) = \varphi_e(x) + 1$ pour tout x. Mais bien évidemment, ce paradoxe est dû au fait que φ_e puisse être définie nulle part, ce qui rappelle l'importance de considérer les fonctions partielles.

Dans la proposition suivante, le théorème de récursion est appliqué pour montrer que la proposition 3.2.5 n'est pas uniformément vraie.

Proposition 3.7.2 *Il n'existe pas de fonction partielle récursive ψ telle que pour tout e, si φ_e est non décroissante, alors $\psi(e)\downarrow$ et $\varphi_{\psi(e)} = \chi_{\mathrm{Img}(\varphi_e)}$.*

Preuve. Supposons qu'une telle fonction ψ existe. Soit ψ_s l'approximation à s étapes de ψ. Le théorème de récursion nous dit qu'il existe un e tel que

$$\{e\}(s) = \begin{cases} 1 & \text{si } \psi_s(e)\downarrow \text{ et } \varphi_{\psi(e),s}(1)\downarrow = 0, \\ 0 & \text{sinon.} \end{cases}$$

Alors $\{e\}$ est non décroissante, donc $\psi(e)$ doit être définie et $\varphi_{\psi(e)}$ est totale. De fait, ou bien $\varphi_{\psi(e)}(1) = 1$, auquel cas $\mathrm{Img}(\{e\})$ contient seulement 0, ou bien $\varphi_{\psi(e)}(1) = 0$, auquel cas $1 \in \mathrm{Img}(\{e\})$. Dans les deux cas, $\varphi_{\psi(e)}(1) \neq \mathrm{Img}(\{e\})(1)$. □

3.8 Exercices

Exercice 3.8.1 Donnez une preuve formelle de la proposition 3.2.6.

Exercice 3.8.2 Montrez que les équivalences entre (ii), (iii) et (iv) dans le théorème 3.2.3 sont toutes uniformes en les codes.

Exercice 3.8.3 Montrez que :

(i) $\leq_m$ est transitive.

(ii) Si $X \leq_m Y$ et Y est r.é. alors X est r.é.

(iii) $X \leq_m Y$ si et seulement si $\overline{X} \leq_m \overline{Y}$.

Exercice 3.8.4 La *jointure* $A \oplus B$ de deux ensembles A et B est définie par :

$$A \oplus B = \big\{2x : x \in A\big\} \cup \big\{2x+1 : x \in B\big\}.$$

L'ensemble $A \oplus B$ contient précisément toutes les informations de A et B compressées ensemble. Montrez que pour tout C, si on a à la fois $A \leq_m C$ et $B \leq_m C$ alors $A \oplus B \leq_m C$. Puisque l'on a clairement que $A, B \leq_m A \oplus B$, cela montre que l'opérateur $\oplus$ sur les m-degrés donne *la plus petite borne supérieure* de A et B.

Exercice 3.8.5 Montrez la proposition 3.6.1. Indice : proposition 3.2.5.

Exercice 3.8.6 Montrez que les ensembles r.é. sont clos par

(i) Intersection,

(ii) Quantification bornée,

(iii) Image de fonctions partielles récursives.

Exercice 3.8.7 (Uniformisation) Un ensemble $A \subseteq \omega^2$ est *à valeur unique* si

$$\forall x, y, z\big((x,y) \in A \wedge (x,z) \in A \to y = z\big).$$

On définit $\mathrm{Dom}(A) = \{x : \exists y\ (x,y) \in A\}$. Montrez que pour tout ensemble r.é. $A \subseteq \omega^2$, il existe un ensemble r.é. $B \subseteq A$ à valeur unique tel que $\mathrm{Dom}(A) = \mathrm{Dom}(B)$.

Exercice 3.8.8 (Propriété de réduction) Montrez que pour toute paire d'ensembles r.é. A et B il existe des ensembles r.é. $A' \subseteq A$ et $B' \subseteq B$ tels que $A' \cap B' = \emptyset$ et $A' \cup B' = A \cup B$.

Exercice 3.8.9 Montrez que K et H ont même m-degré.

Exercice 3.8.10 Dans la preuve de la proposition 3.5.4, il a été montré que tout m-degré contenait au plus $\aleph_0$ ensembles. Montrez que chaque m-degré contient en fait *exactement* $\aleph_0$ ensembles.

Exercice 3.8.11 Donnez une réduction montrant que $\overline{K} \leq_m \{e : W_e = \emptyset\}$.

Exercice 3.8.12 Donnez des réductions pour $K \leq_m X$ et $\overline{K} \leq_m X$, ceci pour tout $X \in \{\mathrm{Fin}, \mathrm{Tot}, \mathrm{Rec}\}$.

Exercice 3.8.13 Montrez que $\overline{\mathrm{Fin}} \equiv_m \mathrm{Tot} \equiv_m \{e : \mathrm{Img}(\varphi_e) \text{ est infini}\}$.

Exercice 3.8.14 Montrez que l'intersection de deux ensembles simples est un ensemble simple. Prouvez ou infirmez : l'union de deux ensembles simples est un ensemble simple.

Exercice 3.8.15 Utilisez le théorème de récursion pour montrer que les objets suivants existent :

- Un e tel que $W_e = \{e\}$.
- Un e tel que pour tout x, $\varphi_e(x)\downarrow \iff \varphi_{2e}(x)\downarrow$.
- Un e tel que $W_e = D_e$, où D_n est l'énumération canonique de tous les ensembles finis de la section 2.4.4.

Exercice 3.8.16 Une fonction récursive f est dite *auto-descriptive* si le résultat e de la recherche $(\mu x)[f(x) \neq 0]$ existe et $\varphi_e = f$. Montrez qu'il existe des fonctions auto-descriptives.

Exercice 3.8.17* Montrez qu'il existe une fonction récursive f telle que pour tout n, $f(n) < f(n+1)$ et $W_{f(n)} = \{f(n+1)\}$.

Chapitre 4

La hiérarchie arithmétique

Au chapitre 3, la notion de m-réduction a été introduite comme un outil pour étudier la décidabilité. Mais les m-degrés peuvent aussi être vus comme une mesure du *degré d'insolvabilité* d'ensembles ou de fonctions. Dans la section 3.6, nous avons vu par comptage qu'il y avait $2^{\aleph_0}$ m-degrés et qu'il existait plus de deux m-degrés r.é. (un degré est appelé r.é. quand il contient un ensemble r.é.). La classification de divers ensembles des mathématiques et de l'informatique en fonction de leur m-degré est donc une envie naturelle que nous tenterons d'assouvir dans ce chapitre. Il s'avère que pour beaucoup d'ensembles définis de manière naturelle, il existe un lien intéressant entre leur degré et la complexité des formules les définissant. Nous commençons donc par définir une hiérarchie basée sur cette idée.

4.1 La hiérarchie arithmétique

Beaucoup d'ensembles de nombres naturels des mathématiques et de l'informatique peuvent être définis par une formule dans le langage de l'arithmétique $\mathcal{L}$. Habituellement, ce langage est celui du modèle standard de l'arithmétique $\langle \omega, S, +, \cdot \rangle$, c'est-à-dire les nombres naturels équipés des fonctions successeur, plus et fois. Par arithmétisation, on peut représenter toutes les fonctions récursives primitives dans ce modèle (cf. théorème 7.1.4), il n'y a donc aucun danger à étendre le langage $\mathcal{L}$ avec des symboles pour chaque prédicat récursif primitif. Notons cette extension $\mathcal{L}^*$. Ce langage contient aussi un symbole de constante n pour chaque élément n de ω. Le modèle que nous destinons à $\mathcal{L}^*$ est donc les nombres naturels avec toutes les fonctions récursives primitives, il est lui aussi noté ω. Enfin, écrire $\omega \models \varphi$ signifie que la formule φ est vraie dans ce modèle.

Définition 4.1.1 Une relation n-aire R est *arithmétique* si elle est définissable par une formule arithmétique, c'est-à-dire s'il existe une formule φ de $\mathcal{L}^*$ telle que pour tout $x_1, \ldots, x_n$,

$$R(x_1, \ldots, x_n) \iff \omega \models \varphi(x_1, \ldots, x_n).$$

Il est possible d'écrire toute formule φ de $\mathcal{L}^*$ sous *forme normale prénexe*, c'est-à-dire avec tous les quantificateurs au début (le *préfixe* de φ), en quantifiant sur une combinaison propositionnelle de prédicats récursifs (la *matrice* de φ). L'idée principale de ce qui va suivre est de distinguer les définitions de formule par leur complexité en quantificateurs. Remarquons dans un premier temps que deux quantificateurs consécutifs de même type peuvent être réduits en un seul en utilisant un codage de paires. Par

exemple, si φ est de la forme

$$\ldots \exists x \exists y \ldots R(\ldots x \ldots y \ldots)$$

alors φ peut être transformé en la formule équivalente

$$\ldots \exists z \ldots R(\ldots (z)_0 \ldots (z)_1 \ldots).$$

Cette propriété est bien évidemment vraie pour tout bloc de quantificateurs consécutifs de même type. Il est donc naturel de mesurer la complexité en quantificateurs en ne comptant que les *alternances* de ces derniers.

Définition 4.1.2 (La hiérarchie arithmétique) La hiérarchie est définie par induction :

- $\Sigma_0^0 = \Pi_0^0 =$ la classe des relations récursives.
- Σ_{n+1}^0 est la classe des relations que l'on peut définir par une formule de la forme $\exists x R(x, \vec{y})$ où R est un prédicat de Π_n^0.
- Π_{n+1}^0 est la classe des relations que l'on peut définir par une formule de la forme $\forall x R(x, \vec{y})$ où R est un prédicat de Σ_n^0.
- $\Delta_n^0 = \Sigma_n^0 \cap \Pi_n^0$ pour tout n.

De fait, Σ_n^0 est la classe des relations définissables par une formule de $\mathcal{L}^*$ en forme normale prénexe avec n alternances de quantificateurs commençant par un quantificateur existentiel. De même pour Π_n^0, mais en commençant la formule par un quantificateur universel. La proposition 3.2.6 nous permet de dire que Δ_1^0 est précisément la classe de toutes les relations récursives, et le théorème 3.2.3 implique que Σ_1^0 correspond aux relations r.é. L'exposant 0 dans la notation des classes de la hiérarchie correspond au fait qu'elles sont définies en utilisant des formules du premier ordre, c'est-à-dire qui n'utilisent de quantifications que sur les nombres. Il existe en effet des versions d'ordre supérieur, que nous ne traiterons pas, définies en utilisant des formules plus complexes. Par exemple, les classes de la *hiérarchie analytique*, qui utilisent l'exposant 1, sont définies en utilisant des quantifications du second ordre, où l'on quantifie aussi sur des ensembles de nombres naturels.

Proposition 4.1.3 (Propriétés de clôture)

1. *$R \in \Sigma_n^0$ si et seulement si $\overline{R} \in \Pi_n^0$.*
2. *Σ_n^0, Π_n^0 et Δ_n^0 sont clos par intersection, union et quantification bornée.*
3. *Δ_n^0 est clos par complémentation.*
4. *Pour $n \geq 1$, Σ_n^0 est clos par quantification existentielle et Π_n^0 par quantification universelle.*

Preuve. Le point 1 découle de la procédure habituelle pour faire descendre une négation au travers de quantificateurs. Pour les points 2 et 3, voir l'exercice 4.3.1. Le point 4 découle de la compression des quantificateurs évoquée plus haut. □

Le théorème suivant montre que cette hiérarchie ne s'effondre pas et qu'elle est stricte à tous les niveaux.

Théorème 4.1.4 (Théorème de la hiérarchie) *Pour tout $n \geq 1$, on a :*

1. *$\Sigma_n^0 - \Pi_n^0 \neq \emptyset$ et donc $\Delta_n^0 \subsetneq \Sigma_n^0$.*

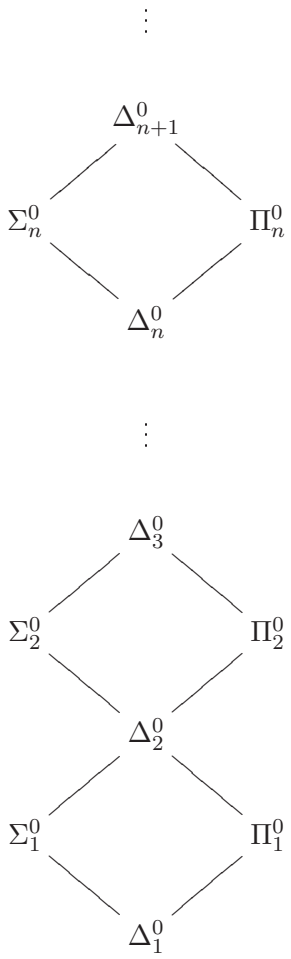


FIG. 4.1 – La hiérarchie arithmétique

2. $\Pi^0_n - \Sigma^0_n \neq \emptyset$ *et donc* $\Delta^0_n \subsetneq \Pi^0_n$.
3. $\Sigma^0_n \cup \Pi^0_n \subsetneq \Delta^0_{n+1}$.

Preuve. Pour chaque niveau Σ^0_n, on définit un K_n analogue à l'ensemble diagonal K, puis on imite la preuve du théorème 3.3.1 pour montrer que K_n n'est pas inclus dans Π^0_n. Soit donc les K_n tels que :

$$K_{2n} = \left\{e : \exists x_1 \forall x_2 \ldots \exists x_{2n-1} \forall s \; \varphi^{2n}_{e,s}(e, x_1, \ldots, x_{2n-1}) \uparrow \right\},$$
$$K_{2n+1} = \left\{e : \exists x_1 \forall x_2 \ldots \forall x_{2n} \exists s \; \varphi^{2n+1}_{e,s}(e, x_1, \ldots, x_{2n}) \downarrow \right\}.$$

Clairement, $K_n \in \Sigma^0_n$ pour tout n, on remarque aussi que $K_1 = K$. Supposons maintenant que $K_n \in \Pi^0_n$ et que n soit impair (le cas n pair est similaire). On a que $\overline{K}_n \in \Sigma^0_n$, donc $\overline{K}_n = \left\{z : \exists x_1 \forall x_2 \ldots \exists x_n \; R(z, x_1, \ldots, x_n)\right\}$ pour un certain prédicat récursif R. Par le théorème 3.2.3, il existe un code e tel que pour tout $z, x_1, \ldots, x_{n-1}$ on ait

$$\exists x_n \; R(z, x_1, \ldots, x_n) \iff \exists s \; \varphi_{e,s}(z, x_1, \ldots, x_{n-1}) \downarrow .$$

On obtient alors une contradiction en fixant $z = e$:

$$\begin{aligned} e \in \overline{K}_n &\iff \exists x_1 \forall x_2 \ldots \exists x_n \; R(e, x_1, \ldots, x_n) \\ &\iff \exists x_1 \forall x_2 \ldots \forall x_{n-1} \exists s \; \varphi_{e,s}(z, x_1, \ldots, x_{n-1}) \downarrow \\ &\iff e \in K_n. \end{aligned}$$

Donc $K_n \notin \Pi^0_n$. Le point 2 découle immédiatement de cela puisque $\overline{K}_n \in \Pi^0_n - \Sigma^0_n$.

Pour le point 3, on considère un prédicat R défini par

$$R(e,i) \iff \big[e \in K_n \wedge i = 0\big] \vee \big[e \in \overline{K}_n \wedge i = 1\big].$$

Ce prédicat n'est pas dans Π^0_n, car on aurait sinon $R'(e) = R(e,0)$ dans Π^0_n ; or ceci est impossible car $R'(e)$ est égal à K_n, et n'est donc pas dans Π^0_n, comme vu précédemment. Le même raisonnement montre que R n'est pas Σ^0_n. De plus, il est facile de vérifier que R est à la fois dans Σ^0_{n+1} et Π^0_{n+1}. D'où $R \in \Delta^0_{n+1} - (\Sigma^0_n \cup \Pi^0_n)$. □

Les ensembles K_n précédemment définis ont un intérêt supplémentaire. Ils sont en effet m-complets à leurs niveaux respectifs :

Proposition 4.1.5 *K_n est m-complet pour Σ^0_n.*

Preuve. Supposons que n soit impair (la preuve est de nouveau similaire pour n pair). Considérons un ensemble Σ^0_n de la forme $A = \big\{e : \exists x_1 \forall x_2 \ldots \exists x_n \; R(e, x_1, \ldots, x_n)\big\}$. On définit alors une fonction récursive f telle que

$$\varphi^n_{f(e)}(z, x_1, \ldots, x_{n-1}) = \begin{cases} 0 & \text{si } \exists x_n \; R(e, x_1, \ldots, x_n), \\ \uparrow & \text{sinon.} \end{cases}$$

Alors

$$\begin{aligned} e \in A &\iff \exists x_1 \forall x_2 \ldots \exists x_n \; R(e, x_1, \ldots, x_n) \\ &\iff \exists x_1 \forall x_2 \ldots \forall x_{n-1} \exists s \; \varphi^n_{f(e),s}(f(e), x_1, \ldots, x_{n-1}) \downarrow \\ &\iff f(e) \in K_n. \end{aligned}$$

□

On considère maintenant une application célèbre de la hiérarchie arithmétique : le théorème de Tarski, qui énonce que la vérité arithmétique n'est pas elle-même une notion arithmétique. Soit $\varphi \mapsto \ulcorner \varphi \urcorner$ un codage récursif primitif des $\mathcal{L}^*$-formules, associant les formules à des nombres. $\ulcorner \varphi \urcorner$ est appelé le *nombre de Gödel* de φ.

Théorème 4.1.6 (Tarski [28]) *L'ensemble*

$$\big\{\ulcorner \varphi \urcorner : \omega \models \varphi\big\}$$

des formules arithmétiques vraies n'est pas arithmétique.

Preuve. Exercice 4.3.3. □

4.2 Calcul de niveaux dans la hiérarchie arithmétique

Donnons nous un ensemble défini arithmétiquement. Il est habituellement facile d'établir une borne supérieure sur son niveau dans la hiérarchie arithmétique : il suffit d'écrire la formule qui le définit en forme normale prénexe et de compter les alternances de quantificateurs. Mais, bien évidemment, on souhaite le plus souvent connaître le niveau exact plutôt qu'une borne supérieure. La proposition 4.1.5 nous offre une méthode pour établir des bornes inférieures : si l'on est capable de montrer que $K_n \leq_m A$ ou $\overline{K}_n \leq_m A$, alors on sait que A ne peut pas apparaître dans un niveau inférieur. Nous donnons maintenant quelques exemples de complexités croissantes illustrant cette méthode.

Proposition 4.2.1 *L'ensemble* $A = \{e : W_e = \emptyset\}$ *est m-complet pour* Π^0_1.

Preuve. Puisque $e \in A \Leftrightarrow \forall x \forall s\ \varphi_{e,s}(x)\uparrow$, on voit que A est au plus Π^0_1. Le théorème de Rice nous montre que A n'est pas récursif, mais on peut en fait montrer sa m-complétude pour Π^0_1. Soient $B = \overline{W}_e$ un ensemble Π^0_1 arbitraire et f une fonction récursive telle que

$$\varphi_{f(x)}(z) = \begin{cases} 0 & \text{si } x \in W_e, \\ \uparrow & \text{sinon.} \end{cases}$$

On a alors $x \in W_e \Leftrightarrow W_{f(x)} \neq \emptyset$, donc $B \leq_m A$ par f. Puisque B était quelconque, ceci montre la complétude de A. Le lecteur qui aurait fait l'exercice 3.8.11 aurait pu prouver la complétude à partir de là. □

Proposition 4.2.2 – Fin $= \{e : W_e$ *est fini*$\}$ *est m-complet pour* Σ^0_2.
– Tot $= \{e : \varphi_e$ *est totale*$\}$ *est m-complet pour* Π^0_2.

Preuve. Le lecteur vérifiera que Fin $\in \Sigma^0_2$ et Tot $\in \Pi^0_2$ (exercice 4.3.4). On montre les deux propriétés de complétude simultanément en donnant une réduction pour les deux. Soit $A = \{e : \forall x \exists y\ R(e,x,y)\}$ un ensemble Π^0_2 quelconque. On définit une fonction récursive f telle que

$$\varphi_{f(e)}(z) = \begin{cases} 0 & \text{si } (\forall x < z)(\exists y)[R(e,x,y)], \\ \uparrow & \text{sinon.} \end{cases}$$

On a alors

$$\begin{array}{lll} e \in \overline{A} & \iff & \exists x \forall y\ \neg R(e,x,y) \iff f(e) \in \text{Fin}, \\ e \in A & \iff & \forall x \exists y\ R(e,x,y) \iff f(e) \in \text{Tot}. \end{array}$$

□

Théorème 4.2.3 (Rogers, Mostowski) Rec $= \{e : W_e$ *est récursif*$\}$ *est un ensemble m-complet pour* Σ^0_3.

Preuve. Le fait que Rec soit Σ^0_3 est montré dans l'exercice 4.3.4. Soit $A = \{e : \exists x \forall y \exists z\ R(e,x,y,z)\}$ un ensemble Σ^0_3 quelconque. À partir de e, on veut construire un ensemble r.é. $W_{f(e)}$ qui est ou bien cofini, donc récursif, ou simple[1], donc non récursif, selon que $e \in A$ ou non. On va énumérer $W_{f(e)}$ dans une construction effective par étapes. On notera le n-ième élément du complément de $W_{f(e),s}$ par a^s_n. À l'étape s, nous avons

$$\overline{W_{f(e),s}} = \{a^s_0 < a^s_1 < a^s_2 < \dots\}$$

On voit les a^s_n comme des marques qui vont sans cesse dans une position plus haute. On remarque que $W_{f(e)}$ est co-infini si et seulement si toutes les marques finissent par s'arrêter, c'est-à-dire si $a_n = \lim_{s\to\infty} a^s_n$ existe pour tout n. On dit que x est *infinitaire* si $\forall y \exists z\ R(e,x,y,z)$; x est *finitaire* sinon. Chaque fois que l'on a une indication que x est infinitaire, on énumérera a^s_x dans $W_{f(e)}$. Donc si x est vraiment infinitaire, $W_{f(e)}$ sera cofini. On va faire en sorte que $W_{f(e)}$ contienne un ensemble simple, qui existe par le théorème 3.6.4, ainsi $W_{f(e)}$ sera lui-même simple si et seulement s'il est co-infini. La construction s'opère comme suit.
Étape $s = 0$. On démarre avec $W_{f(e)} = S$, où S est un ensemble simple. Plus précisément, puisque l'on ne peut pas énumérer tous les éléments de S en une étape, il s'agit

[1] Dans la plupart des preuves de ce point, $W_{f(e)}$ est fait Turing-complet, mais nous n'avons pas encore traité la T-réduction. Cela étant, construire un $W_{f(e)}$ seulement simple facilite grandement la preuve.

en fait d'utiliser le nombre d'étapes nécessaires pour cela. On maintient en parallèle une fonction de longueur $l(x,s)$ qui indique si x est infinitaire. Dans un premier temps, $l(x,0) = 0$ pour tout x.
Étape $s+1$. On définit

$$l(x,s) = \max_{y<s}(\forall y' < y)(\exists z < s)[R(e,x,y',z)].$$

Soit $x \leq s$ minimal tel que $l(x,s+1) > l(e,s)$. Si un tel x est trouvé, on énumère a_x^s dans $W_{f(e)}$, c'est-à-dire que l'on définit $W_{f(e),s+1} = W_{f(e),s} \cup \{a_x^s\}$. Ceci termine la construction.

On vérifie maintenant que $W_{f(e)}$ est bien ce que nous souhaitons. Supposons tout d'abord que tous les x sont finitaires. On a alors que $a_n = \lim_{s\to\infty} a_n^s$ existe pour tout n et donc que $W_{f(e)}$ est co-infini. Puisque $W_{f(e)}$ contient aussi l'ensemble simple S, il est lui-même simple, et donc non récursif. Supposons maintenant le contraire, c'est-à-dire qu'il y ait un x qui soit infinitaire. On a déjà observé précédemment que $W_{f(e)}$ est cofini, donc récursif. Puisque cette construction de $W_{f(e)}$ dépend effectivement de e, la fonction f est récursive, elle est donc une m-réduction, ce qui montre que $A \leq_m \text{Rec}$. Enfin, puisque A était arbitraire, il s'ensuit que Rec est m-complet pour Σ_3^0. □

La construction de la preuve du théorème 4.2.3 est en fait notre premier exemple d'une *construction par priorité*, une méthode que nous étudierons plus en détail au chapitre 6. Le corollaire suivant découle de la preuve :

Corollaire 4.2.4 – *L'ensemble* $\text{Cof} = \{e : W_e \text{ est cofini}\}$ *est m-complet pour* Σ_3^0.
– *L'ensemble* $\{e : W_e \text{ est simple}\}$ *est m-complet pour* Π_3^0.

Il s'avère que la plupart des ensembles arithmétiques naturels sont m-complets pour un niveau de la hiérarchie arithmétique, de telle sorte que la méthode fonctionne presque tout le temps (bien qu'elle puisse devenir plutôt compliquée pour de hauts niveaux). Il y a cependant des exceptions naturelles, provenant par exemple de la théorie de l'aléa (cf. proposition 7.3.3), d'ensembles qui ne sont m-complets à aucun niveau, donc avec lesquels la méthode ne fonctionne pas.

4.3 Exercices

Exercice 4.3.1 Montrez les points 1 – 4 de la proposition 4.1.3.

Exercice 4.3.2 Complétez la proposition 4.1.3 en montrant que :

(i) Π_n^0 et Δ_n^0 ne sont pas clos par $\exists$.

(ii) Σ_n^0 et Δ_n^0 ne sont pas clos par $\forall$.

(iii) Σ_n^0 et Π_n ne sont pas clos par complémentation.

Exercice 4.3.3* Montrez le théorème 4.1.6.

Exercice 4.3.4 Vérifiez les points suivants :

1. $\text{Fin} \in \Sigma_2^0$.
2. $\text{Tot} \in \Pi_2^0$.
3. $\text{Rec} \in \Sigma_3^0$.
4. $\text{Cof} \in \Sigma_3^0$.

5. $\{e : W_e \text{ est simple}\} \in \Pi^0_3$.

Exercice 4.3.5 Avec la notion de T-complétude définie au chapitre 5, montrez que $\{e : W_e \text{ est T-complet pour } \Sigma^0_1\} \in \Sigma^0_4$.

Exercice 4.3.6 Avec la définition des fonctions auto-descriptives de l'exercice 3.8.16, classifiez $\{e : \varphi_e \text{ est auto-descriptive}\}$ dans la hiérarchie arithmétique.

Chapitre 5

Calcul relatif et degrés de Turing

5.1 Réduction de Turing

Dans la définition 3.5.1, nous avons défini une notion de réduction où, pour décider si $n \in A$, nous avions le droit de poser *une* question sur B : « est-ce que $f(x) \in B$? » Cette notion peut évidemment être généralisée à deux, trois ou un nombre fini de questions sur B. Plutôt que d'étudier ces généralisations de la m-réduction, nous passons directement à une notion de réduction des plus libres où les questions de la forme « $n \in A$? » sont résolues en utilisant *n'importe quelle quantité finie d'information* de B, de n'importe quelle manière calculable. Donc lorsque l'on réduit A à B, on suppose que toutes les informations de B sont données et on les utilise pour résoudre A. Pour représenter le fait que B est entièrement connu, on peut par exemple se donner sa fonction caractéristique χ_B dans les fonctions initiales de la définition 2.2.3 :

Définition 5.1.1 (Turing [30]) Soit A un ensemble quelconque. La classe des *fonctions partielles A-récursives* est la plus petite classe de fonctions

1. contenant les fonctions initiales et χ_A,
2. close par composition, récursion primitive et par le schéma de μ-récursion.

Une *fonction A-récursive* est une fonction partielle A-récursive qui est totale.

Alternativement, on pourrait définir la classe des fonctions A-calculables en utilisant une extension des machines de Turing dans laquelle l'information de A est écrite sur un ruban d'entrée infini supplémentaire. On peut alors encore montrer l'équivalence avec l'approche de la définition 5.1.1 en utilisant l'arithmétisation, tout comme au chapitre 2. Nous suivrons donc notre habitude de ne parler que de *fonctions A-récursives* pour désigner tous les objets issus de ces concepts équivalents. Tous les résultats élémentaires sur les fonctions récursives se portent directement sur les fonctions A-récursives. En particulier, il existe un codage similaire à celui du théorème 2.4.1 et l'on peut définir :

Définition 5.1.2 La e-ième fonction partielle A-récursive est notée φ_e^A ou $\{e\}^A$.

Définition 5.1.3 (Post [21]) Un ensemble A se *réduit par réduction de Turing*, ou se *T-réduit*, à un ensemble B, ce que l'on note $A \leq_T B$, si A est B-récursif. On dit que A est *T-équivalent* à B, ce que l'on note $A \equiv_T B$, si l'on a à la fois $A \leq_T B$ et $B \leq_T A$. Les classes d'équivalence de $\equiv_T$ sont appelées degrés de Turing, ou plus simplement *T-degrés*.

Quand $A \leq_T B$ on dira aussi que A est récursif *relativement à* B. L'ensemble B est parfois appelé un *oracle*, puisque l'information qu'il contient est donnée gratuitement. La notion de récursivité relative est centrale dans la théorie de la calculabilité puisqu'elle englobe toutes les manières calculables d'utiliser de l'information. On remarque enfin que cette définition est plus générale que la m-réduction, puisque l'on a

$$A \leq_m B \Longrightarrow A \leq_T B,$$

comme peut facilement vérifier le lecteur (exercice 5.6.1).

De nombreuses notions de la théorie de la calculabilité peuvent être *relativisées* à un oracle A arbitraire, c'est-à-dire en considérant une extension de leurs définitions aux fonctions A-récursives. Par exemple, un ensemble A-r.é. est un ensemble qui peut être énuméré par une fonction A-récursive. De même, de nombreux résultats vus jusqu'à présent peuvent être relativisés, en ce sens qu'ils restent vrais quand ils sont relativisés à un oracle donné. Par exemple, toutes les notions et résultats fondamentaux du chapitre 3 peuvent être relativisés (cf. exercice 5.6.3).

L'observation qui suit sur les calculs relativisés est fondamentale : elle exprime le fait que n'importe quel calcul convergent, qu'il soit relativisé ou non, peut n'utiliser qu'une quantité finie d'information d'un oracle.

Proposition 5.1.4 (Principe d'utilisation) *Soit A un oracle et supposons que l'on ait $\{e\}^A_s(x)\downarrow$. Il y a alors une chaîne finie $\sigma \sqsubset A$ telle que $\{e\}^\sigma_s(x)\downarrow$. En particulier, $\{e\}^B_s(x)\downarrow$ pour tout $B \sqsupset \sigma$.*

Preuve. Tout calcul convergent est constitué d'un nombre fini d'étapes de calcul. De fait, seul un nombre fini de bits de A peut y apparaître. □

Si $\{e\}^A(x)\downarrow$, le nombre maximal appartenant à A utilisé dans le calcul (qui existe par la proposition 5.1.4) est appelé *l'utilisation* du calcul.

5.2 L'opérateur de saut

On peut aussi relativiser l'ensemble diagonal du problème de l'arrêt K à un oracle arbitraire A, ce que l'on notera A' (« A prime ») :

$$A' = \{x : \varphi^A_x(x)\downarrow\}.$$

Il se vérifie aisément que la preuve du théorème 3.3.1 peut être relativisée, de manière à montrer que A' n'est pas un ensemble A-récursif. De plus, on a que $A \leq_m A'$; enfin, l'opérateur $'$ est appelé l'*opérateur de saut*. On remarque par ailleurs que $\emptyset' = K$ ($\emptyset'$ se lit « zéro prime »). On note alors la n-ième itération du saut de A par $A^{(n)}$. On remarque enfin que $\emptyset^{(n)} \in \Sigma^0_n$ (exercice 5.6.4).

La différence entre la réduction *many-one* et celle de Turing peut ainsi être élégamment caractérisée en termes de la hiérarchie arithmétique :

Proposition 5.2.1 (i) $\Delta^0_{n+1} = \{A : A \leq_T \emptyset^{(n)}\}$ *pour tout n.*

(ii) $\Sigma^0_n = \{A : A \leq_m \emptyset^{(n)}\}$ *pour tout n.*

(iii) Σ^0_{n+1} *est composé des ensembles qui sont $\emptyset^{(n)}$-r.é., c'est-à-dire énumérables par une fonction $\emptyset^{(n)}$-récursive.*

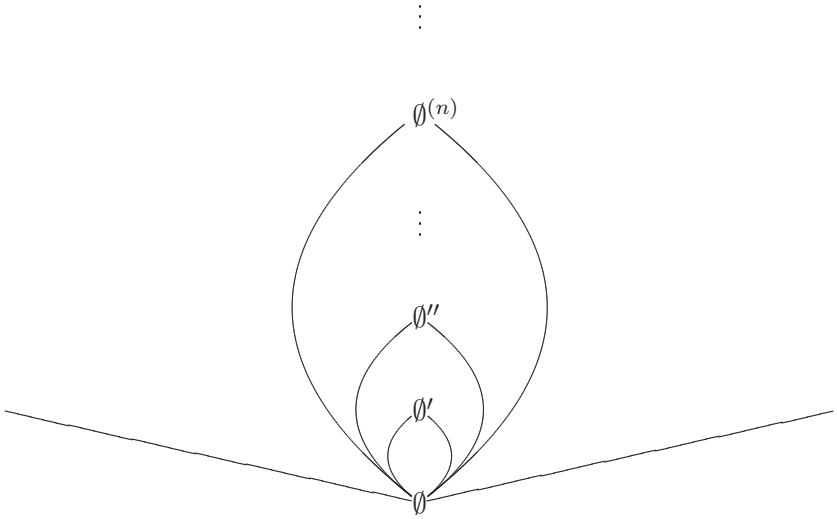


FIG. 5.1 – L'univers de Turing

Preuve. Pour (i), on montre que $\Delta_2^0 = \{A : A \leq_T K\}$ et on remarque que le résultat peut se relativiser. Supposons que $A = \{e\}^K$. On remarque que l'on peut approximer le calcul de $\{e\}^K(e)$ en approximant de manière récursive K. Notons K_s l'approximation à s étapes de K. La suite des calculs $\{e\}^{K_s}(e)$ n'est pas nécessairement stable : il peut arriver que $\{e\}^{K_s}(e)\downarrow$ et que $\{e\}^{K_{s+1}}(e)\uparrow$. Cela étant, en utilisant le principe d'utilisation (proposition 5.1.4), si $\{e\}^K(e)\downarrow$ avec une utilisation u alors $\{e\}^{K_s}(e)\downarrow$ pour tout s tel que $K_s{\restriction}u = K{\restriction}u$. On voit donc que

$$\{e\}^K(e)\downarrow \iff (\exists s)(\forall t > s)\big[\{e\}_t^{K_t}(e)\downarrow\big],$$

ce qui est un prédicat Σ_2^0, d'où $A \in \Sigma_2^0$. Puisque $\overline{A} \leq_T A$, le même argument montre que $\overline{A} \in \Sigma_2^0$, et donc que $A \in \Delta_2^0$.

Pour la réciproque, supposons que A et $\overline{A}$ soient dans Σ_2^0. Puisque tout ensemble Σ_1^0 est K-récursif, A et $\overline{A}$ peuvent être définis par une quantification existentielle sur un prédicat K-récursif. Par la version relativisée du théorème 3.2.3, on a que A et $\overline{A}$ sont K-r.é. Il suit alors par la version relativisée de la proposition 3.2.6 que A est K-récursif.

Le point (ii) est la relativisation de la proposition 3.5.3.

Pour le point (iii), supposons tout d'abord que $A \in \Sigma_{n+1}^0$. Le point (i) implique que tout ensemble dans Π_n^0 est $\emptyset^{(n)}$-récursif. Donc A est définissable par une quantification existentielle sur un ensemble $\emptyset^{(n)}$-récursif. Enfin, par la version relativisée du théorème 3.2.3, A est $\emptyset^{(n)}$-r.é.

Pour la réciproque, supposons que A est $\emptyset^{(n)}$-r.é. L'ensemble A est alors aussi $\overline{\emptyset^{(n)}}$-r.é., donc r.é. dans un ensemble Π_n^0. Par la version relativisée du théorème 3.2.3, A est définissable par une quantification existentielle sur un ensemble Π_n^0, et donc Σ_{n+1}^0. □

La proposition 5.2.1 nous permet de voir que K est Turing-complet pour Δ_2^0, et que la suite $\emptyset$, $\emptyset'$, $\emptyset''$, ..., $\emptyset^{(n)}$, ... constitue une sorte de colonne vertébrale de la hiérarchie arithmétique.

5.3 Ensembles calculables à la limite

Il arrive souvent, en mathématiques et en informatique, que des objets nouveaux soient définis comme étant des limites d'objets plus simples ; par exemple la définition des réels comme les limites de suites de rationnels, l'approximation de fonctions continues par des polynômes ou les ensembles W_e comme limites des $W_{e,s}$, $s \in \omega$. Nous nous intéressons ici à la complexité des objets définis comme limites de procédures *calculables*. Il s'avère qu'une fois de plus la hiérarchie arithmétique est utile pour cette étude.

Définition 5.3.1 Une fonction unaire f est *calculable à la limite* s'il existe une fonction binaire récursive g telle que pour tout n, on ait

$$f(n) = \lim_{s\to\infty} g(n,s).$$

Un ensemble A est calculable à la limite si sa fonction caractéristique χ_A l'est.

Proposition 5.3.2 (Lemme de la limite, Shoenfield [25]) *Un ensemble A est calculable à la limite si et seulement si $A \in \Delta^0_2$.*

Preuve. (Si) On utilise le fait que $\Delta^0_2 = \{A : A \leq_T K\}$ (proposition 5.2.1). Supposons que $A \leq_T K$, mettons $A = \{e\}^K$. On définit

$$g(x,s) = \begin{cases} \{e\}^{K_s}_s(x) & \text{si } \{e\}^{K_s}_s(x)\downarrow, \\ 0 & \text{sinon.} \end{cases}$$

La limite $\lim_{s\to\infty}\{e\}^{K_s}_s(x)$ n'existe pas forcément, puisqu'il peut arriver que $\{e\}^K(x)\uparrow$ mais pour autant, $\{e\}^{K_s}_s(x)\downarrow$ pour une infinité de s. Cela étant, dans notre cas, on sait que $\{e\}^K(x)\downarrow$ pour tout x, puisqu'il est égal à $A(x)$. Donc on a que $\lim_s g(x,s) = A(x)$ pour tout x.

(Seulement si) Supposons que g soit une fonction récursive telle que $A(x) = \lim_s g(x,s)$ pour tout x. Alors

$$\begin{aligned} x \in A &\iff (\exists s)(\forall t > s)[g(x,t) = 1] \\ &\iff (\forall s)(\exists t > s)[g(x,t) = 1]. \end{aligned}$$

A est à la fois Σ^0_2 et Π^0_2, donc Δ^0_2. □

5.4 Degrés incomparables

Jusqu'à présent, nous avons vu qu'il existait plus de deux m-degrés r.é., et plus généralement, qu'il y a une quantité indénombrable de m-degrés. Le même argument de comptage que nous avions utilisé pour les m-degrés montre qu'il y a $2^{\aleph_0}$ T-degrés (exercice 5.6.5). A priori, il pourrait être possible que ces T-degrés soient linéairement ordonnés, mais nous montrons dans cette section que ce n'est pas le cas. Plus précisément, nous montrons qu'il existe des ensembles A et B tels que $A \not\leq_T$ et $B \not\leq_T A$. De tels ensembles sont dits *Turing-incomparables*, ce que l'on notera par $A|_T B$.

Théorème 5.4.1 (Kleene et Post [12]) *Il existe des ensembles Turing-incomparables.*

Preuve. On construit deux ensembles incomparables A et B par extensions finies. On définit deux séries σ_s et τ_s, pour $s \in \omega$, telles que $\sigma_s \sqsubset \sigma_{s+1}$ et $\tau_s \sqsubset \tau_{s+1}$, puis on pose $A = \bigcup_{s\in\omega} \sigma_s$ et $B = \bigcup_{s\in\omega} \tau_s$. La propriété d'incomparabilité de A et B peut être découpée en une infinité de conditions :

$$R_{2e} : \quad A \neq \{e\}^B,$$

$$R_{2e+1} : \quad B \neq \{e\}^A.$$

On construit A et B en une infinité d'étapes s, où à l'étape $s = 2e$ on prend soin de satisfaire R_{2e} et à l'étape $s = 2e + 1$, R_{2e+1}. On commence avec $\sigma_0 = \tau_0 = \emptyset$.
Étape $s = 2e$. À cette étape, σ_s et τ_s sont déjà définis, on définit alors σ_{s+1} et τ_{s+1}. Pour satisfaire R_{2e}, on a besoin d'un témoin x tel que

$$A(x) \neq \{e\}^B(x). \tag{5.1}$$

Soit x le plus petit nombre pour lequel A ne peut pas être défini, c'est-à-dire, soit $x = |\sigma_s|$. Si la partie droite de l'équation (5.1) était non définie, nous pourrions prendre n'importe quoi pour $A(x)$, et si elle était définie, on prendrait un A qui en serait différent. Dans un premier temps, on vérifie s'il existe un $\tau \sqsupseteq \tau_s$ tel que $\{e\}^\tau(x)\downarrow$. Si un tel τ n'existe pas, on pose $A(x) = 0$. Dans le cas contraire, on choisit le plus petit τ (dans un ordonnancement calculable fixé de toutes les chaînes binaires) de taille plus grande que x et l'on définit $\tau_{s+1} = \tau$. On définit ensuite $\sigma_{s+1} = \sigma_s{}^\frown\big(1 - \{e\}^{\tau_{s+1}}(x)\big)$.
Étape $s = 2e + 1$. On effectue les mêmes actions qu'à l'étape $2e$, à ceci près que A et B sont intervertis. Ceci complète la construction.

Notons tout d'abord que $\bigcup_{s\in\omega} \sigma_s$ et $\bigcup_{s\in\omega} \tau_s$ sont totaux, puisque à chaque étape ils sont définis sur un nouveau nombre x. De plus, R_{2e} est satisfaite à l'étape $2e$ par le choix de σ_{s+1} et τ_{s+1}, et similairement pour R_{2e+1}. □

La méthode utilisée dans la preuve du théorème 5.4.1 est puissante et amène une bonne quantité d'information sur la structure des degrés de Turing. Les arguments où les ensembles sont construits en utilisant des extensions finies comme dans la preuve précédente sont appelés des *arguments de Kleene-Post*. La preuve du théorème 5.4.1 peut être approfondie pour montrer bien plus, en particulier qu'il existe un ensemble de $2^{\aleph_0}$ T-degrés incomparables deux à deux (exercice 5.6.8). Ceci montre que l'univers de Turing est au moins aussi large qu'il est haut.

5.5 Inversion du saut

Puisque pour le saut A' de n'importe quel ensemble A on a trivialement que $A' \geq_T \emptyset'$, l'image de l'opérateur de saut est incluse dans le cône $\{B : B \geq_T \emptyset'\}$. On montre dans cette section, en utilisant un argument de Kleene-Post, que l'image de l'opérateur de saut est en fait égale à ce cône.

Théorème 5.5.1 (Théorème d'inversion du saut, Friedberg [6]) *L'image de l'opérateur de saut est* $\{B : B \geq_T \emptyset'\}$.

Preuve. On montre que pour tout $B \geq_T \emptyset'$ il existe un A tel que $A' = B$. Donc soit $B \geq_T \emptyset'$. On définit $A = \bigcup_{s\in\omega} \sigma_s$ par extensions finies, comme dans la preuve du théorème 5.4.1. On commence avec $\sigma_0 = \emptyset$.
Étape $s = 2e$. À cette étape, σ_s est déjà défini, et l'on définit σ_{s+1}. On regarde s'il y a un $\sigma \sqsupseteq \sigma_s$ tel que

$$\{e\}^\sigma(e)\downarrow .$$

Si un pareil σ existe, on prend le plus petit d'entre eux et on définit $\sigma_{s+1} = \sigma$.
Étape $s = 2e + 1$. À cette étape, on code B dans A en posant

$$\sigma_{s+1} = \sigma_s{}^\frown B(e).$$

Ceci complète la construction. On vérifie maintenant que A est tel que désiré. On remarque dans un premier temps que la construction est récursive dans B : les étapes paires peuvent être calculées par $\emptyset'$, et donc par B puisque $\emptyset' \leq_T B$, et les étapes impaires peuvent être calculées par B directement. Dans un deuxième temps, on remarque que la construction est aussi $A \oplus \emptyset'$-récursive, puisque les étapes paires sont $\emptyset'$-récursives, donc avec $\emptyset'$ on peut pour tout e calculer σ_{2e+1} à partir de σ_{2e} et avec l'aide de A calculer σ_{2e+2}, donc $B(e)$. Nous avons donc

$$B \leq_T A \oplus \emptyset' \leq_T A'.$$

La deuxième inégalité est vraie car nous avons $A \leq_T A'$ et $\emptyset' \leq_T A'$ (cf. exercice 5.6.2).

Réciproquement, $A' \leq_T A \oplus \emptyset'$ puisque pour décider si $\{e\}^A(e)\downarrow$, on simule la construction jusqu'à ce qu'à l'étape $2e$ nous puissions calculer l'existence de σ. Si on le peut, on sait qu'on l'a pris, donc $e \in A'$, dans le cas contraire, on a nécessairement $e \notin A'$. Maintenant, nous avons aussi $A \leq_T B$, puisque la construction est B-récursive, comme déjà remarqué. En rassemblant ces résultats, nous avons que

$$A' \leq_T A \oplus \emptyset' \leq_T B.$$

En combinant ceci avec ce que nous avions avant, on voit que $A' \equiv_T B$, comme désiré. □

Les preuves du théorème 5.4.1 et du théorème 5.5.1 sont des extensions de la méthode de diagonalisation vue au début de la section 3.1. En particulier, la preuve du théorème 5.5.1 contient des embryons de la méthode de *forcing*, qui est une autre incarnation de la méthode de diagonalisation et qui a été menée à des sommets en théorie des ensembles. Nous reviendrons sur la puissance de la diagonalisation dans le chapitre suivant.

5.6 Exercices

Exercice 5.6.1 Vérifiez les points suivants :

1. $A \leq_m B \Rightarrow A \leq_T B$.
2. $\overline{A} \leq_T A$.
3. $A \leq_T B \not\Rightarrow A \leq_m B$.

Exercice 5.6.2 En se remémorant l'opérateur de jointure défini à l'exercice 3.8.4, montrez que $\oplus$ est aussi un opérateur de plus petite borne supérieure pour les degrés de Turing, c'est-à-dire, montrez que pour tout C, si $A \leq_T C$ et $B \leq_T C$ alors $A \oplus B \leq_T C$.

Exercice 5.6.3
- Vérifiez que le théorème 3.2.3 peut être relativisé.
- Montrez que $A \leq_m A'$ et que le théorème 3.3.1 peut être relativisé.

Exercice 5.6.4 Montrez que $\emptyset^{(n)}$ est dans Σ^0_n, et qu'il est m-complet pour Σ^0_n.

Exercice 5.6.5 Montrez que tout T-degré est dénombrable et qu'il y a $2^{\aleph_0}$ T-degrés.

Exercice 5.6.6 Montrez qu'il existe des ensembles Turing-incomparables dans Δ^0_2. Indice : montrez que la construction de la preuve du théorème 5.4.1 est récursive dans K.

Exercice 5.6.7 Montrez que les m-degrés ne sont pas linéairement ordonnés. Indice : construire des ensembles A et B tels que pour tout e, la propriété suivante soit satisfaite :

$$R_{2e} : \quad \varphi_e \text{ totale} \Longrightarrow A \not\leq_m B \text{ via } \varphi_e,$$

$$R_{2e+1} : \quad \varphi_e \text{ totale} \Longrightarrow B \not\leq_m A \text{ via } \varphi_e.$$

Pour satisfaire R_{2e}, trouvez un *témoin* x tel que $A(x) \neq B(\varphi_e(x))$. Idem pour R_{2e+1}.

Exercice 5.6.8* Montrez qu'il existe un ensemble de $2^{\aleph_0}$ T-degrés qui sont incomparables deux à deux.. Indice : construire un ensemble parfait, c'est-à-dire un sous-arbre de 2^ω, tel que n'importe quelle paire de chemins infinis satisfasse les conditions du théorème 5.4.1.

Exercice 5.6.9* Un ensemble A est *bas* si le saut de A est le plus petit possible, à savoir $A' \leq_T \emptyset'$. Soit A un ensemble r.é. et supposons que A a une énumération récursive $\{A_s\}_{s\in\omega}$ telle que pour tout $e \in \omega$

$$(\exists^\infty s)\big[\{e\}^{A_s}_s(e)\downarrow\big] \implies \{e\}^A(e)\downarrow .$$

Montrez alors que A est bas. Indice : utilisez le lemme de la limite.

Exercice 5.6.10 (Fonctionnelles continues) Une *fonctionnelle calculable* (ou *fonctionnelle de Turing*) $F : 2^\omega \to 2^\omega$ est une fonction totale de la forme $F(X)(n) = \{e\}^X(n)$, c'est-à-dire dont le n-ième bit de la sortie est donné par le calcul $\{e\}^X(n)$. De manière similaire, une *fonctionnelle A-calculable* est une fonction totale de la forme $F(X)(n) = \{e\}^{A\oplus X}(n)$.

1. Montrez que toute fonctionnelle A-calculable est continue. Indice : utilisez la proposition 5.1.4.
2. Réciproquement, montrez que pour toute fonctionnelle continue F il existe un A tel que F soit A-calculable. Indice : faire en sorte que A code le module de continuité de F.
3. En conclure qu'il y a $2^{\aleph_0}$ fonctionnelles continues.

Chapitre 6

La méthode des priorités

6.1 Diagonalisation, encore

Dans la section 3.1, nous abordions la méthode de diagonalisation. Nous avons ensuite rencontré cette méthode dans certains de nos résultats fondamentaux, tels que l'indécidabilité du problème de l'arrêt (théorème 3.3.1), l'existence d'ensembles récursivement inséparables (théorème 3.3.5), l'existence d'ensembles simples (théorème 3.6.4), le théorème de récursion (théorème 3.7.1) et l'existence de degrés de Turing incomparables (théorème 5.4.1).

Dans un argument diagonal, on essaie en général de construire un objet A avec une propriété globale P en découpant P en une infinité de sous-conditions C_e. Cet objet A est alors construit en utilisant une infinité d'étapes de construction. Par exemple, dans la construction diagonale de Cantor, A est un ensemble avec la propriété globale de ne pas appartenir à une liste dénombrable d'ensembles A_i. Cette propriété peut être découpée en une infinité de sous-conditions C_i statuant que $A \neq A_i$ pour tout i. Dans cette construction, on peut satisfaire tous les C_i dans l'ordre en définissant $A(i) = 1 - A_i(i)$.

Après la preuve du théorème 3.6.4, nous avons vu qu'il existait des cas où nous ne pouvons satisfaire les sous-conditions de la diagonalisation dans l'ordre, mais que nous étions obligé d'adopter une approche plus dynamique pour réussir. Dans ce chapitre, nous généralisons la méthode afin de répondre à une importante question à propos des degrés de Turing r.é.

Mis à part le problème d'arriver à voir comment effectivement satisfaire une condition, il peut arriver que les conditions d'une construction par diagonalisation soient en conflit entre elles. Dans le théorème 3.6.4, par exemple, nous avons déjà vu un conflit entre des conditions qui voulaient garder des éléments à l'extérieur de A, et des conditions qui voulaient en ajouter. Autre exemple, dans la preuve du théorème 4.2.3, il y avait des conditions en conflit, bien qu'il ne fut pas nécessaire de l'expliciter à ce moment. Mais pour ce qui suit, il devient instructif d'analyser ces conditions. En utilisant les notations de la preuve, nous construisions un ensemble r.é. $W_{f(e)}$ qui était cofini si un x infinitaire existait, et co-infini dans le cas contraire. La condition selon laquelle $W_{f(e)}$ doive être co-infini peut être découpée dans des sous-conditions C_x demandant que la marque a_x^s vienne à s'arrêter, c'est-à-dire que $\lim_{s\to\infty} a_x^s$ est finie pour tout x. Pour autant, si x est infinitaire, alors $\lim_s a_x^s = \infty$ et donc on a aussi que $\lim_s a_y^s = \infty$ pour tout $y > x$. En d'autres termes, l'action d'énumérer a_x^s est en conflit avec C_y. Ce conflit peut être résolu en donnant à la première action la *priorité* sur la seconde. De fait, si x est infinitaire, toutes les conditions C_y avec $y > x$ ne sont pas satisfaites. Si, d'un autre côté, tous les x sont finitaires, alors toutes les conditions

C_x sont satisfaites.

En général, la méthode des priorités est une méthode de résolution de conflits entre conditions. Si, dans une construction, une action prise pour satisfaire une condition C_i est en conflit avec une condition C_j, on dit que C_j est *blessée*. Dans ce chapitre, nous ne considérons que les constructions où les conflits entre les conditions sont de nature bénigne, c'est-à-dire où les conditions ne sont blessées qu'un nombre fini de fois. Des arguments de ce type sont appelés des arguments de priorités *finiment blessés*. La méthode de blessure finie est significativement plus puissante que la méthode des extensions finies de Kleene-Post du chapitre 5 et peut être utilisée pour répondre à beaucoup plus de questions. Dans la section suivante nous présentons un premier exemple de ce type.

6.2 Une paire d'ensembles r.é. Turing-incomparables

Dans le théorème 3.6.4, nous avons montré qu'il y a des ensembles r.é. de m-degrés intermédiaires. Après avoir prouvé ceci, Post posa cette fameuse question [20] :

Problème de Post *Existe-t-il des ensembles r.é. de degrés de Turing intermédiaires ? C'est-à-dire, existe-t-il un ensemble r.é. qui ne soit ni récursif ni Turing-complet pour les ensembles r.é. ?*

Dans le théorème 5.4.1, nous avons montré l'existence d'une paire d'ensembles qui étaient Turing-incomparables. Par l'exercice 5.6.6, une telle paire existe aussi sous K. Si nous réussissions à construire ces ensembles de manière à ce qu'ils soient r.é., nous répondrions par l'affirmative au problème de Post. C'est exactement cette méthode qu'utilisèrent Friedberg [5] et Muchnik [16] pour résoudre le problème, indépendamment l'un de l'autre.

Théorème 6.2.1 (Solution au problème de Post, Friedberg [5] et Muchnik [16]) *Il existe une paire d'ensembles r.é. Turing-incomparables.*

Preuve. On construit une paire d'ensembles r.é. incomparables A et B, en utilisant les mêmes conditions que dans la preuve du théorème 5.4.1 : pour tout e, on veut s'assurer que

$$C_{2e} : \quad A \neq \{e\}^B,$$

$$C_{2e+1} : \quad B \neq \{e\}^A.$$

Comme précédemment, pour C_{2e} nous avons besoin d'un témoin x tel que $A(x) \neq \{e\}^B(x)$. On peut prendre un témoin x et définir initialement $A(x) = 0$, et attendre que $\{e\}_s^{B_s}(x)$ converge pour un certain s. Si ceci n'arrive jamais, C_{2e} est automatiquement satisfaite. Si $\{e\}_s^{B_s}(x)\downarrow = 1$, nous avons terminé et si $\{e\}_s^{B_s}(x)\downarrow = 0$, on peut énumérer x dans A donc $A(x) = 1$ et encore une fois C_{2e} est satisfaite. La différence avec le théorème 5.4.1 est qu'ici nous n'avons pas de contrôle complet sur l'ensemble B. En fait, nous savons à l'avance que A et B vont être non-récursifs, et il est typiquement impossible de savoir si, pour un ensemble non-récursif, un élément pourra plus tard apparaître dans l'ensemble (cf. proposition 3.2.5). En ayant satisfait C_{2e} comme précédemment, il peut ainsi arriver que C_{2e+1} veuille énumérer des éléments dans B en dessous de l'utilisation du calcul $\{e\}_s^{B_s}(x)\downarrow$, gâchant la construction par la même

occasion. Pour résoudre un pareil conflit, on donne aux conditions l'ordre de priorité suivant

$$C_0 > C_1 > C_2 > C_3 > \dots$$

Dans le cas d'un conflit, on permet ainsi la condition de plus haute priorité C_i (celle avec le plus petit index i) d'agir, en blessant possiblement les conditions de plus basse priorité. D'un autre côté, si $i < j$, on n'autorisera pas la condition de basse priorité C_j à blesser un calcul de C_i. Nous donnons maintenant la construction formelle de A et B en décrivant leurs énumérations A_s et B_s pour $s \in \omega$. Pour surveiller les nombres que la condition C_e veut garder en dehors de A ou B, nous utiliserons une *fonction de contrainte* $r(e, s)$. On dira qu'une condition C_e est *initialisée* à une étape s si son témoin courant $x_{e,s}$ est redéfini par un nombre plus grand que n'importe quel nombre utilisé précédemment dans la construction, et on fixera la contrainte $r(e, s)$ à 0. On dira que C_{2e} a *besoin d'attention* à une étape s si depuis la dernière étape où C_{2e} a été initialisée, aucune action n'a été prise pour elle, et que pour le témoin courant $x_{2e,s}$, il est vrai que $\{e\}_s^{B_s}(x_{2e,s})\downarrow$. Similairement pour C_{2e+1}.
Étape $s = 0$. Soit $A_0 = B_0 = \emptyset$. On initialise toutes les conditions en posant $x_{e,0} = e$ et $r(e, 0) = 0$ pour tout e.
Étape $s+1$. Soit e le plus petit nombre inférieur à s tel que C_e ait besoin d'attention. On définit

$$r(e, s+1) = \mathrm{use}\big(\{e\}_s^{B_s}(x_{e,s})\big)$$

et on dit que C_e *agit*. Supposons que e soit pair. Si $\{e\}_s^{B_s}(x_{e,s}) = 0$, on définit $A_{s+1}(x_{e,s}) = 1$. Puisqu'une pareille action pourrait blesser une condition de priorité inférieure, on initialise tous les C_j avec $j > e$. Le cas où e est impair est symétrique, en interchangeant les rôles de A et B. Ceci complète la construction.

On vérifie maintenant que toutes les conditions sont satisfaites. On prouve par induction que chaque condition agit et n'est initialisée qu'un nombre seulement fini de fois. Supposons que s soit une étape telle qu'aucune condition C_j, $j < e$ n'agisse après l'étape s. Dans ce cas, C_e n'est jamais initialisée après l'étape s et $\lim_{t\to\infty} x_{e,t} = x_{e,s}$. Si C_e n'a jamais besoin d'attention après l'étape s alors elle est automatiquement satisfaite. Si C_e a besoin d'attention à une étape $t \geq s$ alors sa contrainte $r(e, t)$ est posée comme étant l'utilisation du calcul $\{e\}_s^{B_s}(x_{e,s})$ (dans le cas où e est pair). Puisque toutes les conditions C_j avec $j > e$ sont initialisées à l'étape t, en particulier leurs témoins sont redéfinis comme étant plus grands que $r(e, t+1)$, donc aucune condition de ce type ne peut changer ce calcul. Puisque de plus aucune des conditions C_j avec $j < e$ n'agit après t, le calcul est préservé pour toujours, et donc C_e est satisfaite. Nous voyons donc que toutes les conditions ne sont initialisées qu'un nombre fini de fois, et sont satisfaites ou bien directement, ou par une action supplémentaire, n'agissant donc qu'un nombre fini de fois aussi. □

6.3 Exercices

Exercice 6.3.1* Montrez que les m-degrés r.é. ne sont pas linéairement ordonnés. Indice : construire les ensembles A et B comme dans l'exercice 5.6.7, à ceci près que la construction doit maintenant être effective.

Exercice 6.3.2* Montrez qu'il existe un ensemble simple bas A. Remarquez que l'existence d'un pareil ensemble offre une autre solution au problème de Post. Indice : utilisez la méthode des priorités pour construire A par étapes. Utilisez la caractérisation de l'exercice 5.6.9 pour faire en sorte que A soit bas, et utilisez la stratégie du théorème 3.6.4 pour rendre A simple.

Chapitre 7

Applications

La théorie de la calculabilité est un ingrédient classique de n'importe quel cours de logique mathématique. Bien que des branches telles que la théorie des modèles, la théorie des ensembles et la théorie de la démonstration se soient développées comme des sujets à part entière, les liens avec la théorie de la calculabilité restent nombreux et de nouveaux liens continuent à être découverts. De plus, avec la montée en puissance de l'informatique, elle est devenue un fondement indispensable de ce domaine relativement nouveau. Plusieurs sujets d'informatique ont découlé de la théorie de la calculabilité, comme la théorie de la complexité, la théorie de l'apprentissage, les langages formels et la théorie des automates (cette dernière étant construite sur une autre caractérisation d'ensembles récursifs : la génération à partir d'une *grammaire*). Il n'est jamais très simple de distinguer tous ces domaines et leurs variantes ; par exemple, le point de vue d'Odifreddi [17, 18] est que la complexité, la calculabilité et la théorie descriptive des ensembles sont les mêmes champs.

Dans ce chapitre, nous présentons brièvement trois autres domaines où la calculabilité est centrale : la théorie de la démonstration, le constructivisme et l'aléa.

7.1 Indécidabilité en logique

On appellera *système formel* un ensemble r.é. $\mathcal{F}$ de formules logiques. C'est, bien entendu, une version très épurée de la notion de système formel utilisée habituellement en mathématiques et en informatique (avec les axiomes, les règles de déductions, etc.) mais cela nous suffira ici, et permettra de voir des résultats plus généraux. Par arithmétisation, nous pourrons traiter tout système formel $\mathcal{F}$ comme un ensemble de nombres naturels (en utilisant la numérotation de Gödel, page 35). En particulier, on pourra dire que $\mathcal{F}$ est *décidable* ou non. La propriété que $\mathcal{F}$ soit r.é. correspond au fait que la plupart des systèmes formels ont un ensemble d'axiomes récursif à partir desquels tous les théorèmes du système sont déduits, de telle sorte que l'ensemble des formules déduites soit r.é. Un pareil système est aussi appelé *axiomatisable*. Si une formule φ est énumérée dans $\mathcal{F}$, on dit que $\mathcal{F}$ *prouve* φ, et on appelle φ un *théorème* de $\mathcal{F}$. On écrit alors $\mathcal{F} \vdash \varphi$. Cette précédente définition s'applique aux systèmes formels de n'importe quel langage, mais comme au chapitre 4, nous nous concentrerons sur les systèmes formels dans le langage de l'arithmétique. On rappelle enfin que l'on écrit $\omega \models \varphi$ pour dénoter qu'une formule arithmétique φ est vraie dans le modèle standard de l'arithmétique.

Parmi les propriétés essentielles d'un système logique, on trouve sa véracité (la capacité à ne prouver que ce qui est vrai), sa consistance, et s'il décide de la vérité de toute formule :

Définition 7.1.1 Soit $\mathcal{F}$ un système formel dans le langage de l'arithmétique.
- $\mathcal{F}$ est *adéquate* (on dit aussi correct, fidèle ou fiable, et en anglais *sound*) si $\mathcal{F} \vdash \varphi$ implique que $\omega \models \varphi$.
- $\mathcal{F}$ est *consistant* s'il n'existe pas de formule φ telle qu'à la fois φ et $\neg\varphi$ appartiennent $\mathcal{F}$.
- $\mathcal{F}$ est *complet* si pour toute formule φ, ou bien $\mathcal{F} \vdash \varphi$ ou $\mathcal{F} \vdash \neg\varphi$. Sinon, $\mathcal{F}$ est dit *incomplet*.

Proposition 7.1.2 *Un système formel consistant et complet est décidable.*

Preuve. Ceci est une forme de la proposition 3.2.6. Si $\mathcal{F}$ est complet, on peut énumérer les théorèmes de $\mathcal{F}$ jusqu'à ce que φ ou $\neg\varphi$ apparaisse. Par consistance, une seule de ces deux possibilités peut arriver, et par complétude, une va arriver. On peut donc décider de l'appartenance de φ aux théorèmes de $\mathcal{F}$. □

Dans le reste de cette section, nous travaillerons avec *l'arithmétique de Robinson* $\mathcal{Q}$, qui contient la constante 0, les symboles de fonction S, $+$ et $\cdot$, et les axiomes suivants (en plus des axiomes de la logique du premier ordre avec l'égalité) :

A1. $S(x) = S(y) \to x = y$

A2. $S(x) \neq 0$

A3. $x \neq 0 \to (\exists y)[x = S(y)]$

A4. $x + 0 = x$

A5. $x + S(y) = S(x + y)$

A6. $x \cdot 0 = 0$

A7. $x \cdot S(y) = x \cdot y + x$

Remarquons que les 4 derniers axiomes ne sont que les relations récursives pour $+$ et $\cdot$ que nous avons déjà rencontrées page 5. Il serait possible de travailler avec des systèmes plus faibles que $\mathcal{Q}$, mais $\mathcal{Q}$ sera pratique pour nos besoins. Le système $\mathcal{PA}$ de *l'arithmétique de Peano*, qui est très souvent utilisé comme système d'arithmétique de base, est obtenu en ajoutant à Q le schéma d'axiomes d'induction.

La puissance dont dispose un système formel pour représenter des fonctions est une notion cruciale :

Définition 7.1.3 Une fonction f est *représentable* dans un système formel $\mathcal{F}$ s'il existe une formule φ telle que pour tout x et y,

$$\begin{aligned} f(x) = y &\iff \mathcal{F} \vdash \varphi(x, y), \\ f(x) \neq y &\iff \mathcal{F} \vdash \neg\varphi(x, y). \end{aligned}$$

Ici, x et y qui apparaissent dans $\varphi(x, y)$ sont des raccourcis pour les termes $S^x(0)$ et $S^y(0)$.

Le résultat suivant ajoute une nouvelle méthode à notre arsenal de caractérisations des fonctions récursives.

Théorème 7.1.4 *Toute fonction récursive est représentable dans $\mathcal{Q}$. Réciproquement, toute fonction $\mathcal{Q}$-représentable est récursive.*

Preuve. Les deux sens peuvent être prouvés par arithmétisation (cf. section 2.4). Nous ne prouverons pas ceci ici, une preuve pouvant être trouvée dans Odifreddi [17]. Que toute fonction représentable soit récursive est une application directe de l'arithmétisation : on arithmétise les preuves dans $\mathcal{Q}$. Que toute fonction récursive soit représentable ne requiert qu'une idée de plus : l'élimination des primitives de récursion en utilisant un codage. Ceci peut être effectué de la même manière que dans la preuve du théorème 2.4.1, à ceci près que l'on ne peut plus utiliser les fonctions de codage de la section 2.4.1 puisqu'elles étaient définies en utilisant la récursion primitive. On doit à la place définir des fonctions de codage sans utiliser la récursion primitive. Gödel [7] a défini une telle fonction de codage β en utilisant le théorème des restes chinois (cf. Odifreddi [17]). □

Le résultat suivant est le célèbre (premier) théorème d'incomplétude de Gödel. Il montre que tout système formel est ou bien trop faible pour contenir l'arithmétique élémentaire, ou bien incomplet.

Théorème 7.1.5 (Premier théorème d'incomplétude, Gödel [7]) *Tout système formel consistant $\mathcal{F}$ étendant $\mathcal{Q}$ est indécidable et incomplet.*

Preuve. Soit $\mathcal{F}$ tel qu'énoncé et supposons le décidable. Soit $\{\psi_n\}_{n\in\omega}$ une énumération effective de toutes les formules à une variable. On définit la diagonale D par

$$n \in D \iff \mathcal{F} \vdash \psi_n(n).$$

Puisque $\mathcal{F}$ est décidable, $\overline{D}$ est récursif, donc par le théorème 7.1.4 il est représentable. Mettons que ψ_e représente $\overline{D}$ dans $\mathcal{Q}$. Alors, puisque $\mathcal{F}$ étend $\mathcal{Q}$, on a

$$n \in \overline{D} \iff \mathcal{F} \vdash \psi_e(n).$$

Alors, pour $n = e$, on obtient une contradiction. Ceci montrant que $\mathcal{F}$ est indécidable. Le fait que $\mathcal{F}$ soit incomplet découle maintenant de la proposition 7.1.2. □

Pour un système formel consistant $\mathcal{F}$ étendant $\mathcal{Q}$, Gödel a aussi montré qu'une formule spécifique n'est pas prouvable dans $\mathcal{F}$: il s'agit de la formule $\mathrm{Con}_{\mathcal{F}}$ exprimant la consistance de $\mathcal{F}$. Ce résultat est appelé le *second théorème d'incomplétude*.

Par la proposition 3.5.3, l'ensemble $\overline{K}$ est Π^0_1-complet. Soit $\mathcal{F}$ un système formel adéquat. Il est impossible que pour tout $n \in \overline{K}$ le système $\mathcal{F}$ prouve que $n \in \overline{K}$, puisqu'on aurait alors, du fait que l'ensemble des théorèmes de $\mathcal{F}$ est r.é. et que $\mathcal{F}$ est cohérent, que $\overline{K}$ soit Σ^0_1. On voit donc qu'il y a des (en fait une infinité de) formules de la forme $n \in \overline{K}$ qui sont vraies mais que $\mathcal{F}$ ne peut pas prouver.

Le résultat suivant est la réponse négative au problème de la décision de la page 3 :

Théorème 7.1.6 (Insolvabilité du problème de la décision, Church [2] et Turing [29]) *La logique des prédicats du premier ordre est indécidable.*

Preuve. On utilise le fait que $\mathcal{Q}$ ait une axiomatisation finie. Soit $\wedge\mathcal{Q}$ la conjonction de tous les axiomes de $\mathcal{Q}$. On a alors pour toute formule φ que

$$\mathcal{Q} \vdash \varphi \iff \vdash \wedge\mathcal{Q} \to \varphi,$$

où le symbole $\vdash$ à droite correspond à la déductibilité dans la logique du premier ordre. Ainsi, si la logique du premier ordre était décidable, il s'ensuivrait que $\mathcal{Q}$ le serait aussi, contredisant le théorème 7.1.5. □

7.2 Constructivisme

En mathématiques, il est souvent possible de prouver l'existence d'objets indirectement en montrant que supposer leur inexistence amène à une contradiction, ou en dérivant leur existence d'une disjonction $A \vee \neg A$ sans savoir lequel des deux cas, A ou $\neg A$ est vrai. Au final, il est tout à fait possible d'ainsi montrer l'existence d'un objet sans avoir la moindre idée de la marche à suivre pour obtenir un exemple concret. En mathématiques constructivistes, on essaie de prouver les théorèmes ou de construire les objets de manière explicite. Il existe plusieurs écoles du constructivisme, dépendant de la signification accordée au mot « constructible ». L'école intuitionniste, fondée par le mathématicien néerlandais L. E. J. Brouwer, restreint les moyens de raisonner de la logique classique, en interdisant, par exemple, les méthodes de réflexions indirectes des exemples précédents. Une autre interprétation importante consiste à considérer « constructible » tout ce qui est récursif. Les relations entre ces deux approches du constructivisme est un sujet d'étude intéressant en soi, mais nous ne regarderons que cette dernière approche dans cette section, en indiquant simplement comment la théorie de la calculabilité peut être utilisée pour analyser le contenu constructif des théorèmes de mathématiques classiques. Nous discuterons seulement un exemple, le lemme de König.

Définition 7.2.1 Un *arbre* (binaire) est un sous ensemble T de $2^{<\omega}$ qui est clos par la relation de postériorité $\sqsubseteq$, c'est-à-dire que si $\tau \sqsubseteq \sigma$ et $\sigma \in T$ alors $\tau \in T$. Un arbre est *récursif* si une fois codé comme un ensemble de nombres (en utilisant n'importe quelle codage de $2^{<\omega}$) il est récursif. Un *chemin* dans T est une branche infinie dans l'arbre, c'est-à-dire un ensemble $A \in 2^{\omega}$ tel que $A\restriction n \in T$ pour tout n.

Théorème 7.2.2 (Lemme de König) *Tout arbre binaire infini a un chemin.*

Preuve. Soit T un arbre binaire infini. On « construit » inductivement un chemin dans T comme suit. Soit $\sigma_0 = \emptyset$. Avec σ_n tel que l'ensemble

$$\{\tau \in T : \tau \sqsupseteq \sigma_n\}$$

soit infini, au moins une chaîne entre ${\sigma_n}^\frown 0$ et ${\sigma_n}^\frown 1$ a une infinité d'extensions dans T. On définit $\sigma_{n+1} = {\sigma_n}^\frown 0$ si cela est vrai pour ${\sigma_n}^\frown 0$, et $\sigma_{n+1} = {\sigma_n}^\frown 1$ sinon. On a donc clairement que $\bigcup_n \sigma_n$ est un chemin dans T. □

Le lemme de König est vrai plus généralement pour tout arbre à branchement fini, pas seulement binaire, c'est-à-dire pour tout arbre dans lequel tous les nœuds ont un nombre fini de successeurs. Le résultat suivant montre qu'en général, le lemme de König n'est pas constructif.

Proposition 7.2.3 *Il existe un arbre binaire récursif infini dépourvu de chemin récursif.*

Preuve. Par l'exercice 7.4.3, pour toute paire disjointe A, B d'ensembles r.é., il existe un arbre récursif tel que les chemins dans T soient précisément les ensembles séparants A et B. En prenant A et B une paire d'ensembles récursivement inséparables (théorème 3.3.5), on constate que T n'a pas de chemin récursif. □

Nous voulons maintenant analyser la difficulté du calcul d'un chemin dans un arbre infini. Le résultat suivant montre qu'avoir K pour oracle est suffisant.

Proposition 7.2.4 (Lemme de la base de Kreisel) *Tout arbre binaire récursif infini a un chemin dans* Δ^0_2.

Preuve. Pour un arbre récursif, la décision de la preuve du théorème 7.2.2 est K-récursive. Le chemin construit l'est donc tout autant. □

La proposition 7.2.4 peut être significativement améliorée comme suit. Dans l'exercice 5.6.9, nous introduisions les ensembles bas comme étant des ensembles qui ont le même saut que les ensembles récursifs. Sous cet angle, ils sont assez proches des ensembles récursifs. Par exemple, un ensemble bas ne peut être Turing-complet, puisque les ensembles complets sautent au dessus de $\emptyset''$. La proposition 7.2.3 nous annonce que l'on ne peut s'attendre à ce qu'un arbre récursif ait un chemin récursif, mais le résultat suivant se rapproche de cette possibilité.

Théorème 7.2.5 (Théorème de la base basse, Jockusch et Soare [9]) *Tout arbre binaire récursif infini a un chemin bas.*

Preuve. La preuve est un magnifique exemple de la méthode de *tree-forcing*. Une discussion sur cette méthode nous mènerait, cela étant, bien trop loin. Une preuve peut être trouvée dans Odifreddi [17] ou Soare [27]. □

Remarquons que tous les résultats de cette section peuvent être relativisés en augmentant la complexité de l'arbre, de manière à ce qu'ils ne concernent plus seulement les arbres récursifs, mais permettent d'exprimer la difficulté du calcul d'un chemin dans *n'importe quelle* sorte d'arbre.

Les résultats précédents illustrent comment il est possible d'analyser le contenu constructif des théorèmes mathématiques en utilisant des mesures et des notions de la théorie de la calculabilité. Au final, remarquons que nous ne sommes pas seulement capables de conclure que le lemme de König n'est pas constructif, mais aussi d'obtenir une borne précise de son non-constructivisme en terme de degrés de Turing.

Il existe d'autres domaines dans lesquels la force des théorèmes mathématiques est étudiée, par exemple les mathématiques inversées où, bien que les outils principaux soient d'ordre axiomatiques, la théorie de la calculabilité joue aussi un rôle important. Beaucoup d'autres exemples comme le précédent peuvent être trouvés dans l'ouvrage de Simpson [26].

7.3 Chaînes aléatoires et complexité de Kolmogorov

Si l'on ne se réfère qu'à la théorie des probabilités classique, toute chaîne de caractères de taille n est aussi probable qu'une autre. Pourtant on sent une réelle différence qualitative entre deux chaînes données. Par exemple, la suite $000\ldots0$ de 100 zéros nous semble spéciale, au sens usuel, parmi toutes les chaînes de 100 caractères. Cette intuition est une des motivations pour l'introduction d'une mesure de complexité pour les chaînes et les réels. Une introduction générale à ce domaine pourra être trouvée dans l'ouvrage de Li et Vitányi [14].

Soit une machine universelle de Turing U, c'est-à-dire une machine qui peut simuler toutes les autres machines de Turing. Une telle machine existe pour la même raison qu'il existe des ensembles r.é. universels, cf. page 21. À partir d'une chaîne $\sigma \in 2^{<\omega}$, on définit la *complexité de Kolmogorov* de σ par

$$C(\sigma) = \min\{|\tau| : U(\tau) = \sigma\}.$$

Voici quelques faits à propos de la fonction C :

- On remarque que C dépend du choix de la machine universelle U. Pour autant, ce choix ne fait varier la valeur de la fonction que d'une constante additive ; en effet, pour deux machines universelles U et V et les fonctions de complexité correspondantes C_U et C_V, on a que $C_V(\sigma) \leq C_U(\sigma) + O(1)$ pour tout σ, où la constante $O(1)$ ne dépend que de U et V. Puisque le choix de U n'est pas vraiment important, on notera plus simplement C que C_U.
- Puisque toute chaîne est une description (totale) d'elle-même, la complexité d'une chaîne est bornée par sa taille. C'est-à-dire que pour tout σ, on a

$$C(\sigma) \leq |\sigma| + O(1).$$

L'idée est qu'une chaîne *aléatoire* n'a pas de structure qui pourrait être utilisée pour la décrire de manière plus succincte. Par exemple, la chaîne $000\ldots0$ de 1000 zéros est longue mais peut être décrite facilement. A contrario, une suite obtenue par 1000 jets de pièce n'aura probablement pas de description beaucoup plus courte que 1000 bits. On définit donc :

Définition 7.3.1 Une chaîne σ est *k-aléatoire* si $C(\sigma) \geq |\sigma| - k$.

Proposition 7.3.2 *Pour tout k, il existe des chaînes k-aléatoires.*

Preuve. Cela peut être montré par un simple comptage. Pour toute taille donnée n, il y a

$$\sum_{i=0}^{n-k-1} 2^i = 2^{n-k} - 1$$

programmes de taille $< n-k$, donc il y a au moins $2^n - 2^{n-k} + 1$ chaînes k-aléatoires de taille n. □

Dans le théorème 3.6.4, on a construit un ensemble simple par force brute. Le résultat suivant montre qu'il existe aussi des exemples *naturels* d'ensemble simple :

Proposition 7.3.3 (Kolmogorov) *Pour tout k, l'ensemble de toutes les chaînes non k-aléatoires est simple.*

Preuve. Nous laissons au lecteur la vérification du fait que l'ensemble des chaînes non aléatoires est r.é. (exercice 7.4.5). On doit montrer que pour tout k, l'ensemble des chaînes k-aléatoires est immun[1], c'est-à-dire qu'il n'admet pas de sous-ensembles r.é. infinis. Supposons qu'il ne soit pas immun, alors pour tout m, on peut calculer une chaîne $\psi(m)$ avec $C(\psi(m)) \geq m$. Alors on a que $m \leq C(\psi(m)) \leq \log m + O(1)$, ce qui ne peut être vrai que pour un nombre fini de m, d'où contradiction. □

Corollaire 7.3.4
1. *C n'est pas une fonction récursive.*
2. *Si $m(x) = \min\{C(y) : y \geq x\}$ alors m est non bornée.*
3. *m croît plus lentement que n'importe quelle fonction monotone récursive non bornée.*

Preuve. Voir l'exercice 7.4.6. Que m soit non bornée découle du fait que l'on épuise le stock de programmes courts. Qu'elle grandisse plus lentement que n'importe quelle fonction récursive est dû au fait que dans le cas contraire on pourrait énumérer une infinité de chaînes de grande complexité, ce qui est impossible par la proposition 7.3.3. □

[1] Le terme « immun », emprunté du vocabulaire médicinal par Post, est synonyme de « immunisé ».

7.4 Exercices

Exercice 7.4.1 Mettons que « $e \in \text{Tot}$ » abrège la formule $\forall x \exists s \; \{e\}_s(x)\downarrow$. Soit $\mathcal{F}$ un système formel adéquat. Utilisez la proposition 4.2.2 pour montrer qu'il existe des $e \in \text{Tot}$ pour lesquels $\mathcal{F} \nvdash e \in \text{Tot}$.

Exercice 7.4.2 Prouver le résultat de l'exercice 7.4.1 en construisant un tel e par diagonalisation directe. Indice : soit $\{e\}(x)\downarrow$ pour tout x pour lesquels $\mathcal{F}$ ne prouve pas en x étapes que $e \in \text{Tot}$. Dès que $\mathcal{F}$ prouve que $e \in \text{Tot}$, posez $\{e\}$ comme étant indéfini pour tous les arguments futurs.

Exercice 7.4.3 Soit A et B une paire d'ensembles r.é. disjoints. Montrez qu'il existe un arbre récursif T tel que les ensembles qui séparent A et B sont précisément les chemins dans T.

Exercice 7.4.4 Soit $\mathcal{F}$ un système formel. Une *extension complète* de $\mathcal{F}$ est un ensemble séparant $\{\varphi : \mathcal{F} \vdash \varphi\}$ et $\{\varphi : \mathcal{F} \vdash \neg\varphi\}$.

Montrez qu'il existe une extension complète de $\mathcal{F}$ de degré bas. Indice : utilisez l'exercice précédent et le théorème de la base basse.

Pour un système formel fort $\mathcal{F}$ tel que $\mathcal{Q}$ ou $\mathcal{PA}$, ceci montre que, malgré le fait que l'ensemble $\{\varphi : \mathcal{F} \vdash \varphi\}$ soit Σ^0_1-complet, il existe toujours une extension complète qui a un degré de Turing incomplet. En général, cela étant, une telle extension basse ne peut pas être r.é.

Exercice 7.4.5 Montrez que pour tout k, l'ensemble des chaînes k-aléatoires est Π^0_1.

Exercice 7.4.6 Vérifiez les faits du corollaire 7.3.4.

Approfondissements

Le premier ouvrage intégralement dédié à la théorie de la calculabilité est dû à Rogers [22]. Bien qu'il soit de nos jours obsolète sur bien des points, il n'en reste pas moins une bonne lecture, parfaitement acceptable comme une introduction. Une autre référence de la même époque, plus courte mais plus avancée et d'une influence non négligeable, est due à Sacks [23]. Un deuxième ouvrage du même auteur, [24], traite de la calculabilité sur des ordinaux différents de ω.

L'impressionnant travail en deux volumes d'Odifreddi [17, 18] est l'ouvrage le plus considérable et étendu de nos jours. Le nombre de sujets couverts et son excellente présentation en font une bible du domaine. Son caractère encyclopédique peut le rendre moins adapté pour une introduction, mais il vous sera rapidement indispensable comme ouvrage de référence.

Le livre de Soare [27] a été écrit pour remplacer le livre de Rogers. C'est une excellente référence pour les sujets les plus avancés en théorie de la calculabilité, avec un intérêt particulier pour les degrés de Turing r.é. et la méthode des priorités.

Au chapitre 7, nous avons déjà mentionné quelques références pour les applications de la calculabilité. Une collection étendue de papiers de couverture sur les mathématiques calculables peut être trouvée dans [4]. On pourra trouver des ouvrages traitant de la théorie de la calculabilité, souvent de paire avec un sujet comme la complexité ou la théorie des automates, dans n'importe quelle bibliothèque de mathématiques ou d'informatique.

Du reste, le lecteur pourra se référer profitablement aux papiers originaux des pionniers du domaine. Cela incluera alors les papiers fondateurs de Gödel [7], Church [2], Kleene [10], Turing [29] et Post [20].

Bibliographie

[1] G. Cantor, *Über eine Eigenschaft des Inbegriffs aller reellen algebraischen Zahlen*, J. Math. 77 (1874) 258–262.

[2] A. Church, *A note on the Entscheidungsproblem*, Journal of Symbolic Logic 1 (1936) 40–41.

[3] R. Dedekind, *Was sind und was sollen die Zahlen ?*, Braunschweig, 1888.

[4] Yu. L. Ershov, S. S. Goncharov, A. Nerode, J.B. Remmel et V. W. Marek (eds.), *Handbook of Recursive Mathematics*, Studies in logic and the foundations of mathematics Vol's 138 and 139, North-Holland, 1998.

[5] R. M. Friedberg, *Two recursively enumerable sets of incomparable degrees of unsolvability (solution of Post's problem, 1944)*, Proc. Nat. Acad. Sci. 43 (1957) 236–238.

[6] R. M. Friedberg, *A criterion for completeness of degrees of unsolvability*, Journal of Symbolic Logic 22(1957) 159–160.

[7] K. Gödel, *Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I*, Monatshefte für Mathematik und Physik 38 (1931) 173–198.

[8] D. Hilbert, *Mathematische Probleme*, Proc. Int. Congr. Math. (1900) 58–114.

[9] C. G. Jockusch, Jr. et R. I. Soare, Π^0_1 *classes and degrees of theories*, Transactions of the American Mathematical Society 173 (1972) 35–56.

[10] S. C. Kleene, *General recursive functions of natural numbers*, Math. Ann. 112 (1936) 727–742.

[11] S. C. Kleene, *On notations for ordinal numbers*, Journal of Symbolic Logic 3 (1938) 150–155.

[12] S. C. Kleene et E. L. Post, *The upper semi-lattice of degrees of recursive unsolvability*, Annals of Mathematics, Ser. 2, Vol. 59 (1954) 379–407.

[13] G. W. Leibniz, Dissertatio de arte combinatoria, 1666.

[14] M. Li et P. Vitányi, *An introduction to Kolmogorov complexity and its applications*, Springer-Verlag, 1993.

[15] Yu. V. Matijasevich, *Enumerable sets are Diophantine*, Dokl. Acad. Nauk. 191 (1970) 279–282.

[16] A. A. Muchnik, *On the unsolvability of the problem of reducibility in the theory of algorithms*, Dokl. Akad. Nauk SSSR, N.S. 108 (1956) 194–197 (Russian).

[17] P. G. Odifreddi, *Classical recursion theory*, Vol. 1, Studies in logic and the foundations of mathematics Vol. 125, North-Holland, 1989.

[18] P. G. Odifreddi, *Classical recursion theory*, Vol. 2, Studies in logic and the foundations of mathematics Vol. 143, North-Holland, 1999.

[19] J. C. Owings, *Diagonalization and the recursion theorem*, Notre Dame Journal of Formal Logic 14 (1973) 95–99.

[20] E. L. Post, *Recursively enumerable sets of positive integers and their decision problems*, Bull. Amer. Math. Soc. 50 (1944) 284–316.

[21] E. L. Post, *Degrees of recursive unsolvability*, Bull. Amer. Math. Soc. 54 (1948) 641–642.

[22] H. Rogers Jr., *Theory of recursive functions and effective computability*, McGraw-Hill, 1967.

[23] G. E. Sacks, *Degrees of unsolvability*, Annals of Mathematics Studies 55, Princeton University Press, 1963.

[24] G. E. Sacks, *Higher recursion theory*, Springer Verlag, 1990.

[25] J. R. Shoenfield, *On degrees of unsolvability*, Annals of Mathematics 69 (1959) 644–653.

[26] S. G. Simpson, *Subsystems of second-order arithmetic*, Springer-Verlag, 1999

[27] R. I. Soare, *Recursively enumerable sets and degrees*, Springer-Verlag, 1987.

[28] A. Tarski, *Der Wahrheitsbegriff in den formalisierten Sprachen*, Studia Phil. 1 (1936) 261–405.

[29] A. M. Turing, *On computable numbers with an application to the Entscheidungsproblem*, Proc. London Math. Soc. 42 (1936) 230–265, corrections ibid. 43 (1937) 544–546.

[30] A. M. Turing, *Systems of logic based on ordinals*, Proc. London Math. Soc. 45 (1939) 161–228.

Index