# Chapitre 1

# Complexité de Kolmogorov

**Par *Bruno Durand* et *Alexander Zvonkin***

Le terme « complexité » peut prendre des sens bien différents suivant le contexte où il est employé. Pour modéliser cette notion, on aboutit naturellement à deux formalisations bien différentes, l'une s'appliquant à un *calcul*, la seconde s'appliquant à un *objet*. La *complexité de calcul* est une mesure de temps et/ou d'espace nécessaires pour effectuer le calcul en question. L'autre notion est celle de la *complexité de description*. Cette dernière a pour but de mesurer la quantité d'information nécessaire pour décrire/construire un objet[1] ; on verra par la suite qu'elle s'intéresse à l'espace mémoire minimal pour effectuer cette description. Il n'y a pas de relation directe entre les deux notions : un objet peut être « simple » – sa description est alors courte – alors que sa construction à partir de sa description peut être « complexe » – i.e. nécessiter un calcul gourmand en temps et en espace mémoire.
La *complexité de Kolmogorov* est une théorie de la complexité de description. Comme à son habitude, Kolmogorov a publié, en 1965, une petite note de quelques pages [10], et ainsi fondé une nouvelle branche de recherche. Sa démarche initiale était de formaliser les notions de « simple » et de « compliqué », mais aussi de proposer des fondements supplémentaires à la théorie des probabilités (qu'il avait axiomatisée quelques décennies auparavant) en expliquant la notion de suite aléatoire par l'idée suivante :

Est *aléatoire* ce qui est le plus difficile à décrire.

Les fondements de la complexité de Kolmogorov résident dans la théorie générale des algorithmes. Ce mariage inattendu de deux domaines *a priori* éloignés – dans

[1] Il est vrai que quand on construit effectivement l'objet, la complexité de calcul joue un rôle ; mais la notion de construire en tant que telle ne fait intervenir que la complexité de description, car dès que le programme permettant de construire l'objet est lancé on n'a pas besoin d'autre information.

notre cas, la théorie des probabilités et le monde des algorithmes – est aussi un trait caractéristique de l'œuvre de Kolmogorov.

## 1.1 Algorithmes

Aujourd'hui il est naturel de parler d'algorithmes. Ce n'était pas le cas il y a cent ans, époque où les ordinateurs n'existaient pas plus que les langages de programmation.[2] On était capable de dire, d'une manière plutôt informelle et imprécise, que l'on cherchait « une règle automatique formelle et précise » pour résoudre tel ou tel problème. D'après [19] Émile Borel a le premier décrit en 1912 cette notion d'algorithme mais la théorie des algorithmes s'est développée dans les années 1930 (Turing, Post, Church). On a alors découvert que certains algorithmes avaient une propriété d'universalité et que pour certains problèmes naturels de nature algorithmique il n'existait pas d'algorithme capable de les résoudre. Pour démontrer qu'un algorithme n'existe pas, encore faut-il savoir exactement de quoi on parle et formaliser la notion. C'est l'objet de la théorie générale des algorithmes, appelée aussi *théorie des fonctions récursives* ou *théorie de la calculabilité*.

La suite de cette section s'intéresse à la notion d'algorithme et elle est destinée aux lecteurs qui souhaiteraient avoir des précisions à ce sujet. Elle n'est pas indispensable à la suite de ce chapitre et il est donc possible de se rendre directement au début de la section 1.2.

La notion d'algorithme est très intuitive mais un peu difficile à formaliser. De nos jours, on dispose d'un grand nombre de formalisations dont une est due à Kolmogorov. Dans chacune de ces formalisations on définit une notion de programme. La notion d'algorithme est plus philosophique et peut être considérée comme le dénominateur commun à toutes ces formalisations. Le fait que ces formalisations soient équivalentes en un certain sens que l'on verra par la suite est connu comme étant la thèse de Church–Turing–Post. Dans toutes les formalisations, un algorithme travaille sur des objets de nature discrète.

[2] L'histoire du terme « algorithme » est intéressante en soi. Il provient du nom du savant perse Al-Khwārizmī (787 – c. 850), auteur du livre par l'intermédiaire duquel les Européens ont appris le système de numération de position et les règles de calcul arithmétique (addition, multiplication, etc.). Le nom d'Al-Khwārizmī (qui signifie « de Khorezm », une ville d'Ouzbékistan aujourd'hui appelée Khiva) a été transposé en latin sous la forme *Algorithmus*. Le terme « algorithmes » a d'abord désigné « les règles des quatre opérations arithmétiques ». Puis par extension, il en est venu à désigner toute méthode de calcul systématique. Leibniz appelait « algorithmes » l'ensemble des règles de calcul sur les différentielles et les intégrales. C'est au fur et à mesure que le mot a pris son sens actuel ; il y a cent ans ce processus n'était pas encore terminé. (N.d.A.)

### 1.1.1 Modèles de calcul

Un modèle de calcul est une façon d'écrire les algorithmes, de préciser comment et sur quoi ils agissent, comment on lit leur résultat, etc. Certains modèles de calcul ressemblent à des langages de programmation, d'autres ressemblent davantage à des ordinateurs vus comme des machines physiques, d'autres encore à des systèmes d'exploitation. On peut même les voir comme certains systèmes formels sur les mots ou les entiers.

Quel que soit le modèle choisi, on souhaite s'affranchir des limitations en espace mémoire disponible et en temps de calcul. (L'étude des ressources – temps et espace – nécessaires à la résolution d'un problème relève d'une autre théorie qu'on appelle la *complexité algorithmique*, ou complexité de calcul, ou complexité en ressources. Mentionnons au passage que la notion principale de la complexité de calcul, celle de NP-complétude, a été inventée au début des années 1970 indépendamment par trois chercheurs dont un, Leonid Levin, est un élève de Kolmogorov. Les premiers travaux de Levin portaient sur la complexité de Kolmogorov [21]. On trouvera dans le livre [17] une courte biographie de Levin, ainsi qu'une brève histoire de l'influence que Kolmogorov a eue sur lui.)

Pour décrire précisément ce qu'est un algorithme, il est nécessaire de décrire un modèle de calcul et d'en étudier ses propriétés pas à pas. Cette démarche se retrouve dans quasiment tous les ouvrages traitant du sujet. Kolmogorov a dès le début adopté une autre approche qui se révèle de plus en plus moderne : chacun est supposé intuitivement savoir ce qu'est un algorithme et quelles sont ses principales propriétés. Les propriétés sont admises, considérées comme des axiomes et justifiées (de nos jours) par le comportement des ordinateurs. Cette approche permet de s'affranchir du modèle de calcul et de ses fastidieux détails. Elle est suffisante pour traiter toute la théorie générale des algorithmes y compris la complexité de Kolmogorov mais pas la complexité en ressources (temps, espace) qui nécessite de rentrer dans les détails du modèle de calcul et qui d'ailleurs en dépend dans une large mesure.

Si on veut vraiment écrire des programmes, il est naturel d'utiliser une vraie machine et un langage de programmation bien adapté. En revanche, si l'on cherche à démontrer des théorèmes, alors il est préférable de travailler dans un modèle de calcul abstrait ; un modèle très simple avec peu de primitives très élémentaires est alors bien adapté. Cependant il n'existe pas de modèle canonique pour les preuves car pour démontrer tel ou tel résultat, un modèle de calcul peut s'avérer plus simple qu'un autre.

Ainsi, les modèles les plus populaires sont les machines de Turing, modèles dans lesquels il est assez aisé de montrer l'universalité, mais qui nécessitent des détails techniques de rubans et symboles assez fastidieux à manipuler. Il en existe de nombreuses variantes ; le modèle de la machine de Turing – dans sa version la plus standard actuellement – a d'ailleurs été présenté par Post et non par Turing. Les fonctions récursives « à la Church » sont un modèle plus mathématique et

agréable à manipuler, mais où les preuves des théorèmes de base sont rébarbatives et rebutantes.
Les algorithmes de Markov ressemblent à des systèmes de règles de réécriture avec conditions d'arrêt ; c'est un modèle un peu difficile à manipuler.
Les machines RAM (*Random Access Machines* : « machines à accès direct ») ressemblent plus à des ordinateurs...

### 1.1.2 Tous les modèles de calcul sont équivalents

Ce qu'on appelle communément la thèse de Church est un ensemble d'assertions, chacune relative à un modèle de calcul : dans chaque modèle, la thèse consiste à dire que les fonctions calculables qui y sont définies sont exactement les fonctions intuitivement calculables. Ce n'est pas une assertion mathématique mais une sorte de croyance sur ce qu'est la notion de fonction intuitivement calculable. En revanche la cohérence entre ces différentes assertions est démontrée formellement, puisqu'on prouve que l'ensemble des fonctions calculables est la même dans les différents modèles existants (machine de Turing, fonctions récursives, etc.).
Cette thèse est assez improprement nommée. Church a écrit que les fonctions *totalement définies* calculables intuitivement sont celles de son modèle. Une longue polémique a suivi, polémique dans laquelle Gödel a pris des positions parfois étonnantes [1]. La première équivalence entre modèles (fonctions récursives « à la Church » et machines de Turing) a été démontrée par Turing dans son article fondateur, et la thèse a été exprimée de façon générale (i.e. pour tout modèle de calcul) par Post, d'où l'appellation plus correcte de « thèse de Church–Turing–Post ».
Enfin, la contribution de Kolmogorov à cette thèse aura été de définir un modèle de calcul dans lequel tous les autres puissent s'inscrire. Il est en ce sens plus général que tous les autres, mais comme il est impossible de prouver que tout modèle de calcul qu'on pourrait inventer s'y exprime bien, on obtient une idée philosophique intéressante, une thèse philosophique qui étend la précédente : *tout modèle de calcul qu'on pourrait inventer pour formaliser la notion d'algorithme est exprimable directement à travers les machines de Kolmogorov–Uspensky* (c'est le nom du modèle). L'idée est plus forte que l'équivalence des modèles au sens de l'égalité de la famille des fonctions calculées. Il s'agit d'inclusion directe des modèles dans les machines de Kolmogorov–Uspensky.

### 1.1.3 Machines de Kolmogorov–Uspensky

Ces machines n'ont pas grand-chose à voir avec la complexité de Kolmogorov mais elles ont à voir avec Kolmogorov lui-même et nous allons en dire quelques mots.
Les machines de Kolmogorov–Uspensky sont conceptuellement très simples : elles sont définies sur des graphes appelés agrégats qui représentent les objets calcu-

latoires (les objets sur lesquels porte le calcul), et un pas de calcul consiste en une transformation locale dans ce graphe selon une règle de substitution qui dépend de la machine. Elles ont notamment l'intérêt philosophique de bien cerner le caractère local de l'action de calculer, ce qui n'est pas évident dans tous les modèles mais qui est une des propriétés les plus importantes du calcul. Au sens de la complexité en ressources, elles ont un inconvénient, celui de changer la topologie de l'espace et de mener à la possibilité de calculer sur des arbres, ce qui est un peu trop puissant (en termes de vitesse de calcul). Si on les restreint habilement à l'espace $\mathbb{Z}^3$, elles reprennent une puissance « normale » [19, 8, 1]. Il convient de citer ici le développement des GASM (Gurevich Abstract State Machines) qui sont inspirées de ces modèles mais ont d'autres objectifs et ne jouent pas de rôle spécifique dans la théorie classique de la calculabilité. Le premier exposé complet du modèle de Kolmogorov–Uspensky est dans [11] mais une variante plus moderne est présentée dans [19].

### 1.1.4 Universalité

Voici un aspect de la théorie qui était à la fois surprenant et énigmatique jusqu'au développement des ordinateurs et qui est devenu relativement naturel et couramment utilisé.

Un programme universel $U$ prend en entrée le code d'un programme $p$ et une donnée $x$ ; le résultat $U(p, x)$ qu'il fournit est ce qu'aurait produit l'exécution du programme sur la donnée.

En d'autres termes, imaginez un interpréteur d'un langage de programmation quelconque. Il constitue lui-même un programme universel en ce qu'il est capable d'exécuter tout autre programme.

### 1.1.5 Fonctions non calculables

Ayant accepté l'existence du programme universel, on se heurte à un petit paradoxe. Considérons la fonction $F(p) = U(p, p) + 1$. Cette fonction est clairement calculable car la fonction $U$ l'est. Elle doit donc avoir un programme associé que l'on notera $q$. Alors, d'une part, l'application du programme $q$ à $q$ lui-même donne comme résultat $U(q, q)$ ; mais, d'autre part, le même résultat doit être égal à $F(q) = U(q, q) + 1$. Ce qui conduit à $U(q, q) = F(q) = U(q, q) + 1$. La seule façon de se sortir du paradoxe est de considérer que certains programmes *a priori* valides calculent des fonctions non partout définies. Pour la même raison, déterminer par un algorithme si un calcul va donner un résultat n'est pas possible.

### 1.1.6 Retour sur les algorithmes

Enfin, on pourrait penser que la notion d'algorithme est l'idée commune à toutes les modélisations précédentes. Ce ne serait que partiellement vrai car on exclurait ainsi les algorithmes qui font autre chose que transformer une entrée en une sortie. Par exemple les algorithmes interactifs, les algorithmes de jeux et les algorithmes randomisés.

## 1.2 Descriptions et tailles

Toute information peut être codée par une suite de bits. Pour cette raison, on va dans la suite considérer que les objets avec lesquels on travaille sont de telles suites.
Soit $\mathbb{B} = \{0, 1\}$ ; cet ensemble est communément appelé l'alphabet et l'ensemble des mots finis sur cet alphabet (suites finies de lettres) est noté $\mathbb{B}^*$. On va identifier $\mathbb{B}^*$ avec $\mathbb{Z}^+ \setminus \{0\} = \{1, 2, 3, \ldots\}$ en suivant l'ordre lexicographique, en commençant par le mot vide, associé à 1, puis $0 \mapsto 2$, $1 \mapsto 3$, $00 \mapsto 4$, $01 \mapsto 5$, etc. Pour le dire plus brièvement : on fait précéder chaque mot binaire d'un 1 et on identifie le mot ainsi obtenu à l'entier dont il est l'écriture en base 2. (Par exemple, le mot 00 devient 100 qui est l'écriture en base 2 de l'entier 4, donc on identifie le mot 00 à l'entier 4.) La longueur $|u|$ d'un mot binaire $u$, i.e. son nombre de lettres, peut être identifiée avec la partie entière $\lfloor \log u \rfloor$ du logarithme en base 2 de l'entier associé à $u$. (Attention, $|u|$ représente la longueur du mot et non la valeur absolue de l'entier associé.)

**Définition 1.2.1.** *Soit $f : \mathbb{B}^* \to \mathbb{B}^*$ une fonction calculable. On définit la* complexité de $x \in \mathbb{B}^*$ par rapport à $f$ *comme étant la valeur*

$$K_f(x) = \begin{cases} \min |t| & \text{tel que } f(t) = x, \\ \infty & \text{si un tel } t \text{ n'existe pas.} \end{cases}$$

*On dit aussi que $t$ est une* description *de $x$ par rapport à $f$.*

Le défaut principal de cette définition est que la complexité dépend du choix de la fonction $f$. On ne peut totalement se passer de $f$ mais le théorème ci-dessous (dû à Kolmogorov mais qui est déjà informellement présent dans l'article de Solomonoff [18]) explique en quoi cette dépendance n'est pas incontournable. Ce théorème a été prouvé ultérieurement et indépendamment par Chaitin mais il ne figure que dans son second article sur le sujet [2, 3] – la revendication de la paternité de la théorie a provoqué de longues polémiques stériles expliquées dans [13].

**Théorème 1.2.1** (Existence d'une fonction optimale)**.** *Il existe une fonction $f_0$ (appelée* fonction optimale*) telle que, quelle que soit une autre fonction $f$, il existe*

*une constante $C$ telle que*

$$\forall x \quad K_{f_0}(x) \leq K_f(x) + C. \tag{1.2.1}$$

(*La constante $C$ dépend de $f$ mais pas de $x$.*)

**Preuve.** Soit $t$ une description de $x$ par rapport à $f$, i.e. $f(t) = x$. Alors $f_0$ utilise comme description de $x$ la paire $(p, t)$ où $p$ est un programme qui calcule la fonction $f$. Pour coder cette paire, on peut coder d'abord $p$ puis $t$ à la suite. La constante $C$ est donc la longueur du codage de $p$, tandis que $f_0$ n'est rien d'autre qu'une version un peu spécifique d'une fonction universelle. □

**Remarque 1.2.1.** La façon dont on code $p$ a peu d'importance : elle n'influe que sur la constante $C$. En revanche il est indispensable de pouvoir coder $t$ sans perte d'espace. Par exemple si on réserve un octet pré-défini pour marquer la fin des fichiers (comme on le fait dans certains mauvais systèmes d'exploitation largement utilisés) le théorème précédent n'est pas vrai : on ne peut utiliser que 255 octets sur les 256 existants et la taille du codage de $t$ (donc aussi sa contribution à la complexité $K_{f_0}(x)$) est augmentée d'un facteur multiplicatif plus grand que 1 (plus exactement $\log(256)/\log(255)$). Pour pouvoir reconstruire $p$ et $t$ à partir du mot d'entrée, il faut savoir où s'arrête le codage de $p$. On peut utiliser à cet effet un quelconque système auto-délimité pour coder $p$ puisqu'on vient de voir que la taille du codage de $p$ importe peu.

**Corollaire 1.2.1.** *Si $f_1$ et $f_2$ sont deux fonctions optimales, alors il existe une constante $C$ telle que*

$$\forall x \quad |K_{f_1}(x) - K_{f_2}(x)| \leq C. \tag{1.2.2}$$

Étant donné que les complexités selon deux fonctions optimales diffèrent d'une constante, on peut choisir une fonction optimale $f_0$ et la fixer une fois pour toutes ; l'indice $f_0$ dans $K_{f_0}$ est désormais supprimé. Une autre façon équivalente de voir les choses est de ne pas fixer la fonction optimale mais de considérer que la complexité de Kolmogorov est définie à une constante additive près et ne prend un sens qu'asymptotiquement :

**Définition 1.2.2.** *La* complexité de Kolmogorov $K(x)$ *est la complexité $K_{f_0}(x)$ par rapport à une fonction optimale $f_0$. La complexité $K(x)$ est « définie à une constante additive près ».*

**Proposition 1.2.1.**

$$K(x) \leq |x| + C, \quad \textit{ou bien} \quad K(x) \leq \log x + C. \tag{1.2.3}$$

**Preuve.** Il suffit de prendre $f(x) = x$ dans la formule (1.2.1), i.e. on prend $x$ lui-même comme description de $x$. □

**Proposition 1.2.2** (distribution des complexités)**.** *Parmi les mots de longueur $|x| = n$, la part de ceux dont $K(x) < n - k$ ne dépasse pas $2^{-k}$.*

**Preuve.** Le nombre des mots de longueur $n$ est $2^n$, tandis que le nombre des descriptions (éventuelles) de longueur inférieure à $n - k$ est

$$1 + 2 + \ldots + 2^{n-k-1} < 2^{n-k}.$$

□

Il existe des mots qu'on appelle *incompressibles* : il est impossible, pour les décrire, d'économiser un seul bit. En effet, le nombre des mots de longueur $n$ est $2^n$, tandis que le nombre de descriptions potentielles de longueur strictement inférieure à $n$ est $1+2+\ldots+2^{n-1} = 2^n - 1$. Par conséquent, pour chaque $n$ il existe des mots $x$ de longueur $n$ tels que $K(x) \geq n$. Comme il est de coutume dans cette théorie, il s'agit là d'« existence pure » : ce type de mot existe mais il est impossible d'en trouver par une quelconque procédure algorithmique. D'ailleurs, si on pouvait les construire ils auraient justement une complexité faible (bornée par le programme de construction). Mais le seul fait de leur existence a des conséquences très importantes pour l'informatique théorique (théorie des automates, des langages formels etc.).
Aujourd'hui tout un chacun utilise des logiciels de compression de données, ce qui n'était pas le cas dans les années 1960. La complexité de Kolmogorov nous permet de raisonner sur les compressions optimales pour des familles d'objets. Ainsi, si un marchand de logiciels veut vous vendre un de ses produits sous l'argument qu'il compresse tout d'un certain facteur (même constant) nous vous suggérons d'être circonspect.

Enfin, pour rendre la section suivante sur le théorème d'incomplétude de Gödel plus facile à comprendre, nous vous proposons une variation sur le thème bien connu du « castor affairé » (en anglais : *busy beaver*)[3].

[3]Ainsi dénommé parce que formulé initialement dans un modèle où une machine de Turing écrivait des bâtons sur un ruban (l'alphabet était {blanc, bâton}). Le problème du castor affairé peut être présenté de la façon suivante. Soit n'importe quel modèle de calcul. Notons $\mathcal{P}_n$ l'ensemble des programmes (dans ce modèle) de taille au plus $n$. Certains de ces programmes s'arrêtent sur l'entrée vide ; d'autres non. Soit $\mathcal{A}_n \subseteq \mathcal{P}_n$ le sous-ensemble de ceux qui s'arrêtent. Soit enfin $T(n)$ le nombre maximum d'opérations effectuées par un programme de la classe $\mathcal{A}_n$ sur l'entrée vide (et $T(n) = 0$ si $\mathcal{A}_n = \emptyset$). Étant donné que l'ensemble $\mathcal{A}_n$ est fini, le maximum $T(n)$ est bien défini. Cet ensemble grandit avec $n$, la fonction $T$ est donc monotone. Or, cette fonction n'est pas calculable. Qui plus est, elle ne peut pas être bornée par une quelconque fonction calculable (sinon le problème de l'arrêt serait décidable). Ainsi, comme la fonction $n!^{n!^{\cdot^{\cdot^{\cdot^{n!}}}}}$ ($n!$ étages) est calculable, $T(n)$ grandit plus vite que cette fonction ! (N.d.A.)

Définissons $\delta(n)$ comme le plus grand entier possédant une description de taille inférieure (strictement) à $n$. Le nombre de descriptions $t$ de taille $|t| < n$ étant fini, un tel entier existe certainement. Nous avons donc par définition $n \leq K(\delta(n)+1)$. Si la fonction $\delta$ était calculable, nous aurions aussi $K(\delta(n)+1) \leq \log n + C$ car $n$ pourrait servir comme description de $\delta(n)+1$. La contradiction est évidente. Donc nous avons là une fonction non calculable. Nous pouvons montrer pire encore : $\delta$ croît plus vite que toute fonction calculable. Nous remplaçons tout simplement dans les inégalités précédentes $\delta$ par une fonction calculable qui la majore.

## 1.3 Théorème de Gödel

### 1.3.1 Il est prouvé qu'on ne peut pas tout prouver

La fonction $K(x)$ n'est pas calculable. Alors à quoi sert-elle ? – par exemple, à démontrer des théorèmes. L'exemple le plus remarquable, peut-être, est la preuve du théorème d'incomplétude de Gödel. Ce théorème affirme, *grosso modo*, que toutes les vérités ne sont pas démontrables. Les mathématiques ont leurs limites intrinsèques : il existe des propositions qui sont vraies, mais qui sont impossibles à démontrer.
Nous vous proposons une forme plus « concrète » d'une proposition vraie mais non démontrable ; elle a été inventée par Gregory Chaitin [4].

**Théorème 1.3.1** (Théorème d'incomplétude de Gödel)**.** *Il existe $m_0$ tel que pour tout $x$ et tout $m \geq m_0$ la proposition*

$$\boxed{K(x) \geq m}$$

*n'est pas démontrable.*

Remarquons que l'ensemble des $x$ pour lesquels $K(x) < m$ est fini, et ceci quel que soit $m$. Donc, la proposition $K(x) \geq m$ est *vraie* pour une infinité de valeurs de $x$. Et pour toutes ces vérités, il n'y a pas de preuve.

**Démonstration du théorème de Gödel.** Pour prouver ce théorème, nous utilisons une méthode assez analogue à la solution du problème du *castor affairé*, introduite à la fin de la section précédente. Supposons que pour tout entier $m$ (ou pour une infinité d'entiers $m$, ce qui revient au même), il existe un $x$ tel qu'on puisse montrer « $K(x) \geq m$ ». L'algorithme suivant trouve un tel $x$ en fonction de $m$ :
- entrée : un entier $m$ ;
- passer en revue et examiner l'un après l'autre tous les théorèmes (un théorème est une proposition admettant une preuve) ;
- dès qu'un théorème « $K(x) \geq m$ » est trouvé, rendre $x$ comme résultat de l'algorithme.

Par rapport à cet algorithme, le nombre $m$ pris en entrée est une description de la sortie $x$, d'où, selon (1.2.3), $K(x) \leq \log m + C$. Mais d'autre part $K(x) \geq m$ (puisqu'on vient de trouver un théorème qui affirme cela !). Par conséquent

$$m \leq \log m + C.$$

La constante $C$ ne dépend ni de $m$ ni de $x$. Il en résulte que l'ensemble des valeurs de $m$ vérifiant cette dernière inégalité est fini. On en déduit qu'une théorie logique ne peut démontrer « $K(x) \geq m$ » que pour un nombre fini de valeurs de $m$. Les hypothèses sous lesquelles cette preuve est valide sont vérifiées dans tous les systèmes de preuves usuels (possibilité de faire des opérations arithmétiques usuelles, etc.). □

Pour un néophyte il est difficile d'apprécier à sa juste valeur ce qui vient de se passer sous ses propres yeux. Le théorème de Gödel a littéralement bouleversé tous les esprits dans la communauté mathématique du début des années 1930. Les projets et les espoirs d'aussi grands mathématiciens que David Hilbert de formaliser une fois pour toutes les mathématiques étaient tous réduits à néant. Jusqu'à nos jours ce théorème reste un résultat monumental et l'un des joyaux de la logique mathématique. Des générations de logiciens ont aspiré à comprendre tous les *pourquoi* et *comment*. Beaucoup de livres volumineux ont été écrits ; le célèbre ouvrage de vulgarisation de Douglas Hofstadter [9] fait à peu près 800 pages. Et voilà, nous avons démontré ce théorème en un seul paragraphe et d'une manière tout à fait élémentaire !
Un philosophe a remarqué que « toute grande idée passe, dans son développement, par trois stades : (1) c'est impossible ; (2) c'est peut-être possible mais incompréhensible ; (3) c'est trivial ». Apparemment le théorème de Gödel est arrivé au troisième stade.

### 1.3.2 Systèmes formels

Ceci était donc notre commentaire pour un novice. Mais un lecteur plus averti peut s'inquiéter. Il a peut-être entendu parler de *systèmes formels*. En effet, nous parlons de preuves et de théorèmes sans pour autant préciser les axiomes, les règles logiques, les règles d'inférence et tout ce qui va avec (pas mal de choses en fait). Or, en réalité tout cela n'est pas nécessaire. Il y a une seule condition qui doit être imposée sur un système de preuves pour qu'il soit assujetti au théorème de Gödel, et cette condition saute presque aux yeux quand on regarde la démonstration du théorème de plus près. Cette condition est explicitée dans la définition suivante.

**Définition 1.3.1** (Système formel)**.** *Un système de preuves est dit* formel *s'il existe un algorithme qui peut énumérer les théorèmes du système.*

Tous les systèmes de preuves (souvent appelés théories logiques) usuels sont bien des systèmes formels au sens qui précède. En effet, énumérer les théorèmes se fait de la façon suivante : on écrit dans l'ordre tous les mots. Par mot on entend ici texte mathématique. On vérifie ensuite si le mot écrit est une preuve dans notre système. C'est possible en vérifiant que le mot est construit selon les règles formelles de construction des preuves à partir des axiomes de la théorie. Ensuite on extrait de la preuve la formule démontrée qui est le théorème (c'est en général la dernière formule de la preuve). Il faut donc que ces opérations soient réalisables par des algorithmes et c'est le cas pour tous les systèmes logiques usuels (énumération des mots, vérification des preuves, extraction de la formule démontrée i.e. le théorème).
Exiger l'existence de telles règles est tout à fait raisonnable, au moins pour les mathématiques ; sinon, comment pourrait-on s'assurer qu'une preuve est correcte ? Par un vote de jurés ? Par un recours à l'avis d'un oracle, d'un prophète ou d'une autre sorte d'autorité ? Par un tournoi à l'instar des chevaliers au Moyen Âge ? Par un tirage au sort ? En résumé, dans tout système formel à l'usage des mathématiques figure l'idée que l'exactitude des preuves peut être vérifiée mécaniquement, i.e. par un algorithme.

### 1.3.3 Paradoxe de Berry

Gregory Chaitin, l'auteur de cette démonstration remarquable, explique que sa preuve est une formalisation du « paradoxe de Berry » publié par Bertrand Russell en 1908. Russell propose de considérer la phrase suivante :

$N$ est le plus petit entier dont une description nécessite plus de mille mots.

Premièrement, remarque Russell, des entiers qui nécessitent plus de mille mots pour être décrits, existent – tout simplement parce que le nombre de descriptions plus courtes que mille mots est fini. Deuxièmement, le plus petit d'entre eux possède cette propriété. Donc la phrase citée caractérise le nombre $N$ sans ambiguïté ; en d'autres termes, elle est une description de $N$. Or, elle contient moins de mille mots !
Un paradoxe est souvent la conséquence d'une mauvaise formalisation des notions qui le constituent. Par exemple, la notion de plus petit élément utilisée ci-dessus est bien formalisée : un des axiomes généralement utilisé pour définir l'ensemble $\mathbb{N}$ est que tout sous-ensemble non vide de $\mathbb{N}$ admet un plus petit élément. En revanche, pour le paradoxe de Berry, la notion mal formalisée est celle de description. La complexité de Kolmogorov nous donne un cadre formel pour cette notion. Ainsi, remplaçons les mots par les bits, et considérons la phrase suivante :

$N$ est le plus petit entier tel que $K(N) > m_0$.

Ici $m_0$ est la constante du théorème de Gödel. Alors un tel entier existe, et il est bien unique. Mais la phrase elle-même ne peut pas servir de description pour $N$ : aucun algorithme ne permet de trouver $N$ en utilisant cette phrase.

### 1.3.4 Propositions gödéliennes : exemples « concrets »

Les bons étudiants demandent souvent : peut-on donner un exemple concret d'une proposition non démontrable ?
Cette question nous tend un piège. Bien sûr, il ne s'agit pas d'une proposition fausse, n'est-ce pas ? Le fait qu'une proposition fausse n'ait pas de démonstration ne doit étonner personne. Donc la question elle-même doit être précisée : peut-on donner un exemple concret d'une proposition *vraie mais non démontrable* ? Mais alors comment savoir que la proposition en question est vraie s'il n'existe pas de démonstration pour la démontrer ? ! Apparemment, il faut avoir d'autres raisons, et de très fortes raisons d'ailleurs, pour *croire* qu'elle est vraie.
La première approche a été trouvée par Gödel lui-même. Il a démontré qu'en n'utilisant que l'arithmétique il est impossible de démontrer que cette même arithmétique est non contradictoire. Les théoriciens des nombres ne s'en sont pas inquiétés. Après une expérience de plus de deux millénaires de travail sur l'arithmétique on peut raisonnablement croire qu'on n'arrivera pas un jour à une contradiction.
La complexité de Kolmogorov nous donne un autre procédé de fabrication de propositions « gödéliennes » (vraies mais non démontrables). Supposons, pour fixer les idées, que $m_0 = 100$. Alors on peut jeter une pièce de monnaie, disons, 500 fois, puis affirmer que la complexité de la suite de bits ainsi obtenue est supérieure à 100. Cette affirmation sera impossible à démontrer, mais il est pratiquement sûr qu'elle soit vraie : la probabilité qu'elle soit fausse est à peu près $2^{-400}$ (voir la proposition sur la distribution des complexités, p. 8). Nous obtenons ainsi une proposition d'arithmétique à laquelle nous croyons pour des raisons de nature probabiliste.

## 1.4 Définition du hasard

### 1.4.1 Questions, questions, questions...

Quand on commence à réfléchir plus profondément sur les notions de probabilité et de hasard, on trouve beaucoup de difficultés pour expliquer les choses les plus « basiques » qui soient. Commençons par un exemple tiré de la vie quotidienne. Supposons que vous voyiez une voiture avec une plaque d'immatriculation 7777 ZZ 77. Ce numéro vous semble assez extraordinaire, n'est-ce pas ? Quant au numéro, disons, 7353 NY 42, il vous semble « normal ». Pourquoi ?

On a envie de dire : parce que le premier numéro est moins probable. Or, cette réponse n'est pas valable : la probabilité du premier numéro est exactement la même que celle du deuxième, et que celle de n'importe quel autre numéro d'ailleurs. En supposant tous les chiffres et toutes les lettres équiprobables et indépendants, on trouve que cette probabilité est égale à $1/(10^6 \times 26^2)$. Quand on jette une pièce de monnaie mille fois, la probabilité d'obtenir la face mille fois est $2^{-1000}$, mais la probabilité de toute autre suite de piles et de faces est exactement la même ! Alors pourquoi la suite des mille tirages identiques éveille-t-elle des soupçons quant à son caractère aléatoire ?

En réfléchissant davantage sur ce phénomène, nous comprenons qu'en fait, en parlant de « numéros », nous n'entendons pas des numéros indiviuels mais des *ensembles* de numéros « comme celui-ci ». Pour le premier, il appartient à l'ensemble des numéros dont « les chiffres se répètent, les lettres aussi ». Cet ensemble est simple à décrire et sa probabilité est petite. En ce qui concerne le deuxième numéro, il est « comme tout le monde ». On n'arrive pas à cerner une propriété qui lui serait spécifique et qui décrirait un ensemble peu probable. (Ou bien si on y arrive, cela n'a pas été prévu par les auteurs.) En gros, chaque particularité réduit significativement la complexité et fait appartenir l'objet en question à un ensemble peu probable.

Allons plus loin : qu'est-ce que la *probabilité* ?

Malgré ce qu'on a pu croire, la théorie des probabilités (dont les bases mathématiques rigoureuses ont été données par Kolmogorov lui-même en 1933) ne répond pas à cette question. Cette théorie énonce, sous forme d'axiomes, les propriétés des probabilités. Elle permet aussi de calculer les probabilités de certains événements quand d'autres probabilités sont connues. Ainsi, elle permet aux mathématiciens de travailler dans le cadre de cette théorie exactement de la même manière que dans le cadre de toute autre branche des mathématiques, sans se soucier beaucoup de questions philosophiques et « inutiles ». Tout le monde en était content ; pas Kolmogorov.

Imaginez que votre interlocuteur possède de grandes capacités intellectuelles mais ne connaisse pas la notion de probabilité. Vous lui affirmez que, lorsqu'on jette une pièce de monnaie, la probabilité d'obtenir pile ou d'obtenir face est 1/2. Il vous questionne alors :

— Je ne comprends pas ce que signifie le mot « probabilité » dans votre phrase.

— Je veux dire par cela que les chances d'obtenir pile ou face sont égales.

— Vous n'avez dit rien de nouveau, à part de remplacer le mot *probabilité* par le mot *chance*, que je ne comprends pas plus.

— Bon, d'accord. Je veux dire par cela que si vous jetiez cette pièce de monnaie un millier de fois, vous obtiendriez à peu près une moitié de « pile » et une moitié de « face ».

— Ah... Il semble que je commence à comprendre quelque chose. Laissons pour l'instant de côté les mots « à peu près » qu'on pourra préciser plus tard. Mais dites-moi : est-il *toujours* vrai que le nombre de tirages est réparti moitié-moitié ?

— Hélas, non ; ce n'est pas toujours le cas, mais c'est bien le cas avec une très grande probabilité. Il reste toujours une chance mais très faible d'obtenir uniquement des piles.
— « Avec une très grande probabilité... » Ce mot encore une fois ! Vous m'aviez donc expliqué cette notion par elle-même mais cette fois dans un contexte plus compliqué, celui des suites de milliers de jets au lieu d'un seul jet. Tout ceci ne fait pas très sérieux !
— Mais attendez, attendez ! Je peux vous donner les axiomes qui décrivent les propriétés de la probabilité...
— Connaître les propriétés de quelque chose est sans doute très bien. Mais il serait souhaitable, avant de parler de propriétés, de comprendre quel est l'objet dont on envisage d'étudier les propriétés. *« Les sapettes servent au sapettage, on les met dans un sapettarium, on peut en faire des chapelets, et elles peuvent siffler »* : cela vous dit quelque chose ?[4]
— Notre discussion est déjà assez longue, mais il me reste encore une approche pour essayer de vous convaincre. Voyez-vous, la propriété d'avoir la proportion de 0 et de 1 proche de 1/2 est vraie non seulement avec une grande probabilité mais encore pour *toutes* les suites aléatoires. Les suites qui ne vérifient pas cette propriété ne sont pas aléatoires.
— La suite qui consiste en des 0 et 1 alternés est-elle selon vous aléatoire ?
— Bien sûr elle ne l'est pas : elle est évidemment trop régulière pour être aléatoire !
— Alors je ne comprends pas du tout ce nouveau terme que vous venez d'introduire : *aléatoire*. Que signifie ce mot ?
— Heu...
— Existe-t-il au moins des axiomes pour régir les propriétés d'objets dits aléatoires ?
— Heu...

### 1.4.2 Suites aléatoires

L'approche basée sur la complexité de Kolmogorov permet, en considérant les suites binaires, d'éviter le recours à la notion de probabilité. Chaque suite binaire *individuelle* peut être *aléatoire* ou non. *Toutes* les suites aléatoires (et non pas « la majorité » comme avant) contiennent à peu près une moitié de bits 1 et une moitié de bits 0. Être aléatoire est le synonyme d'avoir une complexité proche de sa longueur. Autrement dit, la meilleure (ou plutôt la moins mauvaise) manière de décrire une telle suite est de la présenter telle quelle. D'autre part, si une suite de 1000 bits possède, disons, 300 bits 1 et 700 bits 0, cela réduit significativement sa complexité, et elle n'est pas donc aléatoire. Mais si on pousse ce même raisonnement un peu plus loin, si cette suite avait exactement le même nombre

[4]Voir Stanislas Lem, *les voyages électriques d'Ijon Tichy*, présence du futur, Denoël, 1980 : quatorzième voyage. (N.d.A.)

de 0 et de 1, on aurait ainsi un peu d'information sur elle et elle ne pourrait être parfaitement aléatoire. Dans une suite aléatoire, le nombre de 0 et de 1 doit être très légèrement déséquilibré (un petit logarithme apparaît ici).
Revenons à notre question : comment définir le caractère aléatoire d'une suite ? L'idée développée ci-dessous (et attribuée à Kolmogorov) est qu'une suite est aléatoire si elle est « quasiment » incompressible. Mais pour que cette notion prenne un sens, il est indispensable de passer des suites finies aux suites infinies. Donc attention, à partir de maintenant, les suites (notées typiquement $x$) sont finies ou infinies.
Plus précisément, si la suite (infinie) est notée $x$ et si $x_{1:n}$ représente la suite finie constituée des premiers bits de $x$ jusqu'au $n$-ième, on dira que $x$ est aléatoire si et seulement si

$$\exists C \ \forall n \quad K(x_{1:n}) > n - C.$$

Le terme constant $C$ est habituel et provient de la définition de $K$ à une constante additive près. Mais cette définition ne fonctionne pas bien (il n'existe pas de suite aléatoire en ce sens alors qu'on voudrait que presque toutes les suites le soient). Il a fallu dix ans pour résoudre ce problème. On lit trop souvent dans la littérature qu'il faut un « artifice technique » pour résoudre le problème. Mais ce qui est qualifié par les mathématiciens d'artifice technique correspond à une réalité bien connue des informaticiens : pour mémoriser un objet quelconque de $n$ bits dans un système de fichiers, il faut un peu plus de $n$ bits (et même un peu plus que $n+\log n$ bits). Ceci est valable quel que soit le système de fichiers et pour être plus correct on devrait dire qu'il n'est pas possible de concevoir un système de fichiers dans lequel on puisse enregistrer n'importe quel objet de $n$ bits en utilisant moins de $n + \log n$ bits.
Mathématiquement ce qu'on va demander à nos descriptions c'est d'être « auto-délimitées » (ou encore d'avoir une propriété de préfixe). Cela s'exprime de la façon suivante : si une description (un mot fini) $t$ produit $x$, alors toute continuation de $t$ (i.e. tout mot – fini – qui prolonge $t$) produit aussi $x$. En d'autres termes on sait où est la fin « utile » du mot $t$. Dans nos ordinateurs, cette fin est donnée par le système de gestion des fichiers. Si on exprime la complexité de Kolmogorov dans ce modèle où les descriptions sont auto-délimitées (ce qui nécessite quelques précautions mais est faisable) on obtient par la formule présentée au paragraphe précédent une « bonne » définition de la notion de suite aléatoire. On obtient aussi que la complexité d'un couple d'entiers est bornée par la somme des complexités des deux entiers ce qui n'était pas vrai pour la complexité de Kolmogorov originale où un terme logarithmique « parasite » apparaissait en sus.
Une autre idée de Kolmogorov est de définir une suite aléatoire comme étant *une suite qui échappe à tout ensemble de mesure constructivement nulle*. Pour définir cette expression de « mesure constructivement nulle » on prend la définition usuelle (i.e. l'une quelconque des définitions usuelles) des ensembles de mesure nulle et on la rend « constructible », au sens suivant : lorsqu'on a par exemple une

formule du type $\forall x\ \exists y\ \ P(x,y)$ on remplace dans la formule $P(x,y)$ la variable $y$ par une fonction calculable de $x$ (i.e. on suppose qu'il existe un algorithme qui produit $y$ à partir de la donnée de $x$). Ceci donne la définition suivante :
$A$ est un ensemble de mesure constructivement nulle si et seulement s'il existe un programme $p$ qui, pour tout entier $n$ donné en entrée, produit une suite (infinie) de séquences (finies)

$$x_0^{(n)}, x_1^{(n)}, \ldots$$

telle que pour tout $n$

$$\sum_i 2^{-|x_i^{(n)}|} < 1/n$$

et que tout $w \in A$ a comme préfixe l'un des $x_i^{(n)}$.
Cette idée a été développée par Martin-Löf [15], élève de Kolmogorov. Les ensembles de mesure constructivement nulle correspondent à des tests de « non-aléatoirité » et la suite est donc aléatoire si elle résiste à tous ces tests. L'existence de programmes universels permet aisément de déduire qu'il existe un test *universel* (et même il en existe plusieurs) : toute suite qui résiste à un tel test résiste aussi à tous les autres tests et est donc aléatoire.
Un des principaux résultats de la théorie algorithmique de l'information est d'établir l'équivalence entre l'incompressibilité des suites infinies, et leur caractère aléatoire vu comme leur résistance à tout test algorithmique. Cette équivalence est un théorème prouvé indépendamment dans les années 1970 par Levin et Schnorr [12, 16] dans des contextes de définitions légèrement différents (Schnorr se ralliera aux définitions de Levin par la suite). On aboutit ainsi à une bonne définition du caractère aléatoire d'une suite infinie, bonne en ce qu'elle est indépendante des approches considérées (le texte de référence est [20]). De plus, tout théorème *usuel* de probabilité s'exprimant par « pour presque tout $x$, la propriété suivante est vraie » est vrai sur toutes les suites aléatoires au sens précédent. Il n'y a pas de preuve générale de ce dernier résultat : différents auteurs ont étudié différents théorèmes (par exemple le théorème ergodique) et ont montré qu'ils restaient vrais pour toute suite algo-aléatoire. C'est dans certains cas un travail très délicat (par exemple pour le théorème ergodique) et il a fallu une dizaine d'années pour l'achever.
Cette équivalence donne aussi un outil de preuve puissant en combinatoire. La démarche est en général de montrer que si une formule est (par exemple) fausse sur une suite, alors sa complexité de Kolmogorov ne peut être maximale (voire est assez faible) et donc pour presque toute suite la formule est vraie.

### 1.4.3 Suites de faible complexité

On a vu que les suites de forte complexité sont les suites aléatoires. Il est naturel de se demander quelles sont celles de faible complexité. Il existe un très beau

théorème, démontré indépendamment par plusieurs auteurs au début du développement de la théorie de la complexité de Kolmogorov. Si on compte la date de première parution, ce théorème doit être attribué à Albert Meyer et il a été publié dans un article de Loveland [14]. La meilleure preuve figure dans l'article de référence [21]. Nous utilisons les notations de la section précédente.

**Théorème 1.4.1.** *La suite $x$ est récursive (i.e. calculable par algorithme) si et seulement si $\exists C\ \forall n\ \ K(x_{1:n} \mid n) < C$.*

Ici nous utilisons une forme un peu plus générale dite *conditionnelle* de la complexité de Kolmogorov. Dans un but de simplification nous l'avions jusqu'à présent éclipsée mais elle est en réalité indispensable. Nous définissons $K(x \mid y)$ (complexité de $x$ sachant $y$) en donnant aux descriptions un accès à $y$ en entrée. Cette information supplémentaire $y$ est là pour aider à la description de $x$.

$$K_f(x \mid y) = \begin{cases} \min |t| & \text{tel que } f(t, y) = x, \\ \infty & \text{si un tel } t \text{ n'existe pas.} \end{cases}$$

L'existence de fonctions optimales se démontre comme précédemment. La complexité étant définie à une constante additive près on en déduit immédiatement que si $y$ est constant en fonction de $x$, la notion n'a pas d'intérêt. On pourra en revanche prendre pour $y$ la taille de $x$, le nombre de ses bits à 0, la suite de ses bits d'indice pair, etc.
Le théorème exprime que les suites « simples » sont exactement les suites calculables. Il convient de remarquer qu'on mesure $K(x_{1:n}|n)$ et non $K(x_{1:n})$ car la longueur de $x_{1:n}$ contient une petite information : celle contenue dans l'entier $n$. Pour les suites aléatoires (forte complexité) ce terme n'est pas nécessaire mais on pourrait l'ajouter sans modifier la notion.
L'une des implications présentées dans ce théorème est triviale : si la suite est récursive, la formule est évidemment vraie, $C$ étant par exemple pris égal à la taille du programme calculant $x_{1:n}$ à partir de $n$.
La réciproque est subtile et amusante. C'est un des exemples qui apparaissent de temps en temps en informatique théorique, où on montre qu'un algorithme existe sans pour autant être capable de le construire. La situation est une des pires qu'on puisse envisager : en effet, on montre que la suite est récursive sans pouvoir borner la taille du programme générant $x$ en fonction de la constante $C$. Il a été montré récemment qu'il s'agit non seulement d'une preuve non constructive d'existence d'algorithme, mais qu'en plus aucune preuve constructible n'existe. C'est un corollaire d'un résultat qui peut s'exprimer en termes plus « humains » : appelons *complexité de génération* de $x$ la taille du plus petit programme calculant $x$ (calculant $x_{1:n}$ à partir de $n$). C'est la taille minimale qu'il faut pour un programme qui produit $x$. Appelons *complexité d'observation* de $x$ la limite supérieure de la complexité des segments initiaux (i.e. de $K(x_{1:n}|n)$). C'est la complexité maximale qu'on observe à un temps donné pour $x$, et la limite supérieure correspond au temps infini. La complexité d'observation est trivialement

inférieure à celle de génération. Mais on ne peut majorer la complexité de génération par une fonction calculable de la complexité d'observation. Autrement dit, si on note $o$ la complexité d'observation et $g$ la complexité de génération, pour toute fonction calculable $f$ il existe des suites pour lesquelles $g > f(o)$. Il existe donc des suites $x$ (récursives) dont on observe des complexités très faibles lorsqu'on les coupe à un temps $t$ quelconque mais qui, cependant sont *très* difficiles à calculer dans leur ensemble [6, 7].

### 1.4.4 Retour sur la définition de « suite aléatoire »

Des critiques importantes peuvent être formulées à l'égard de notre définition de « suite aléatoire » (on dira dans cette section « algo-aléatoire »). La première est qu'elle introduit la notion d'algorithme dans les mathématiques des probabilités. Or ces mathématiques n'utilisent jamais d'algorithmes. La théorie des ensembles suffit à la théorie des probabilités, et la notion d'algorithme est, elle, extérieure à la théorie des ensembles. D'où la question naturelle suivante : la notion d'algorithme est-elle indispensable pour donner une définition robuste de la notion de suite aléatoire ?

La deuxième critique est que certaines suites aisément définissables sont algo-aléatoires. Par exemple la suite $\omega$ de Chaitin est algo-aléatoire. Cette suite représente la suite des bits de la probabilité qu'un programme s'arrête, en prenant une distribution uniforme[5] sur les programmes. Le théorème « pour presque tout $x$, $x \neq \omega$ » est clairement vrai (presque tout ensemble évite un point isolé fixé) mais « pour tout $x$, $x \neq \omega$ » est faux sur les suites algo-aléatoires. Même si ceci peut être considéré comme un peu artificiel, un vrai problème est soulevé.

La première possibilité est de généraliser la notion d'algo-aléatoire aux formules arithmétiques ; on obtient ainsi une notion de suites « arithmo-aléatoires. » Deux définitions formelles sont possibles : une en considérant la théorie classique basée sur les algorithmes et en relativisant ces algorithmes par l'usage d'oracles, l'autre en définissant directement la notion par des formules arithmétiques. Ces deux définitions mènent à la même notion, plus générale que celle des suites algo-aléatoires. Le problème qui reste est que cette définition n'est pas close : l'ensemble des suites arithmo-aléatoires n'est pas arithmétique. Ceci est dû à une différence structurelle importante entre les ensembles énumérables et les ensembles arithmétiques : il existe un phénomène d'universalité pour les ensembles énumérables mais pas pour les ensembles arithmétiques.

Ainsi on peut faire une autre proposition plus radicale de définition [5] : consi-

[5] On affecte à tous les programmes de longueur $n$ la même probabilité $2^{-2n}$, pour la raison suivante. Il y a $2^n$ programmes de longueur fixée $n$, et on affecte à l'ensemble des programmes de longueur $n$ la probabilité $2^{-n}$ (pour que la série converge) qu'on répartit uniformément entre chaque programme. Autrement dit, pour tirer un programme au hasard suivant cette loi, on tire d'abord sa longueur $n$ (avec la probabilité $2^{-n}$), puis on tire au hasard parmi les $2^n$ programmes de longueur $n$, suivant la loi uniforme. (N.d.A.)

dérons tous les théorèmes de la forme « pour presque tout $x$, $P(x)$ » où $P$ est une formule logique. Ces théorèmes sont dénombrables (et même énumérables). À chacun d'eux correspond donc un ensemble « de vérité » de mesure 1. Intéressons-nous à l'intersection de tous ces ensembles. Par $\sigma$-additivité (additivité dénombrable) de la mesure, cette intersection est de mesure 1. Il suffit donc de la prendre comme ensemble de suites aléatoires. Cette intersection est en particulier contenue dans les ensembles des suites algo-aléatoires et arithmo-aléatoires. Hélas cette définition ne fonctionne pas du tout car elle se heurte à un paradoxe classique de la théorie des ensembles : il n'existe pas d'ensemble de tous les ensembles. Ainsi l'intersection proposée n'est pas définissable ! Pire encore, on prouve qu'il est impossible d'échapper à ce paradoxe en établissant une définition fondée sur la prouvabilité au lieu de la définissabilité.
Pour échapper à ce problème, il faut effectuer une très jolie pirouette et proposer une définition dont l'idée est la suivante : remplaçons l'ensemble prouvablement minimal (l'intersection des théorèmes) par un ensemble minimal de façon consistante (on ne peut pas prouver qu'il n'est pas minimal). Cet ensemble minimal existe mais ce n'est pas évident du tout ; la preuve met en œuvre des techniques fines de théorie des ensembles. Expliquons un peu mieux cette notion en revenant à l'intuition des tests de « non-aléatoirité ». On va dire qu'une suite $s$ n'est pas aléatoire si on peut d'une part prouver un théorème portant sur un ensemble de mesure 1 (pour presque tout $x$ $P(x)$) et prouver aussi que $s$ n'appartient pas à cet ensemble (« non $P(s)$ »). Alors pour l'ensemble que nous avons introduit, une suite est aléatoire si on ne peut pas trouver de test l'excluant : elle est aléatoire si on ne peut pas prouver qu'elle ne l'est pas ! C'est un peu comme la présomption d'innocence : une suite doit toujours être présumée aléatoire ; si elle est soupçonnée de ne pas l'être, une sorte de procès est intenté contre elle ; mais en l'absence de preuve définitive de sa « culpabilité » la suite est relaxée au bénéfice du doute.

# Bibliographie


[1] A. Blass et Y. Gurevich, *Algorithms : a quest for absolute definitions*, Bull. EATCS **81** (oct. 2003), p. 195-225.

[2] G. Chaitin, *On the length of programs for computing finite binary sequences by bounded-transfer Turing machines*, AMS Notices **13** (1966), p. 133.

[3] G. Chaitin, *On the length of programs for computing finite binary sequences by bounded-transfer Turing machines II*, AMS Notices **13** (1966), p. 228-229.

[4] G. Chaitin, *Computational complexity and Gödel's incompleteness theorem*, AMS Notices **17** (1970), p. 672.

[5] B. Durand, V. Kanovei, V. Uspensky et N. Vereshchagin, *Do stronger definitions of randomness exist ?*, Theoret. Comput. Sci. **290** (2003), no. 3, p. 1987-1996.

[6] B. Durand et S. Porrot, *Comparison between the complexity of a function and the complexity of its graph*, Theoret. Comput. Sci. **271** (2002), no. 1-2, p. 37-46.

[7] B. Durand, A. Shen et N. Vereshchagin, *Descriptive complexity of computable sequences*, Theoret. Comput. Sci. **271**, (2002), no. 1-2, p. 47-58.

[8] Y. Gurevich, *On Kolmogorov machines and related issues*, Bull. EATCS **35** (juin 1988), p. 71-82.

[9] D. Hofstadter, *Gödel, Escher, Bach : les brins d'une guirlande éternelle*, InterÉditions (Paris), 1985.

[10] A. N. Kolmogorov, *Three approaches to the definition of the concept of « quantity of information »* (en russe), Problemy Peredachi Informatsii **1** (1965), no. 1, p. 3-11.

[11] A. N. Kolmogorov et V. A. Uspensky, *On the definition of an algorithm*, Translations, series 2, Amer. Math. Soc. **29** (1963), p. 217-245. (Traduction de Uspekhi Mat. Nauk **13** (1958), no. 4, p. 3-28.)

[12] L. Levin, *The concept of a random sequence*, Soviet Math. Doklady **212** (1973), p. 1413-1416.

[13] M. Li et P. Vitányi, *An introduction to Kolmogorov complexity and its applications*, Graduate Texts in Computer Science, Springer-Verlag (New York), 1997 (2-ième édition).

[14] D. W. Loveland, *A variant of Kolmogorov concept of complexity*, Information and Control **15** (1969), p. 510-526.

[15] P. Martin-Löf, *The definition of random sequences*, Information and Control **9** (1966), p. 602-619.

[16] C. P. Schnorr, *Process complexity and effective random sets*, J. Comput. System Sci. **7** (1973), p. 376-388.

[17] D. Shasha et C. Lazere, *Out of their minds : The lives and discoveries of 15 great computer scientists*, Copernicus, 1995.

[18] R. J. Solomonoff, *A formal theory of inductive inference, I*, Information and Control **7** (1964), p. 1-22.

[19] V. Uspensky et A. Semenov, *Algorithms : main ideas and applications* (trad. du russe par A. Shen), Kluwer, 1993.

[20] V. Uspensky, A. Semenov et A. Shen, *Can an (individual) sequence of zeros and ones be random ?*, Russian Math. Surveys **45** (1990), no. 1, p. 121-189.

[21] A. K. Zvonkin et L. Levin, *The complexity of finite objects and the development of the concepts of information and randomness by means of the theory of algorithms*, Russian Math. Surveys **25** (1970), no. 6, p. 83-124.