Laboratoire d'Informatique Fondamentale de Lille

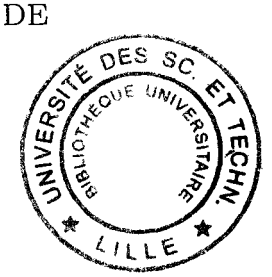

Numéro d'ordre : 2320

THÈSE

Nouveau Régime

Présentée à

L'UNIVERSITÉ DES SCIENCES ET TECHNOLOGIES DE LILLE



Pour obtenir le titre de

DOCTEUR EN INFORMATIQUE

par

Sylvain PORROT

COMPLEXITÉ DE KOLMOGOROV ET ANALYSE DE FLOTS DE DONNÉES

Directeurs de thèse

Max DAUCHET et Bruno DURAND

Thèse soutenue le 25 Septembre 1998, devant la Commission d'Examen :

Président	:	Jean-Paul DELAHAYE	Université de Lille I
Rapporteurs	:	André ARNOLD	Université de Bordeaux I
		Serge GRIGORIEFF	Université de Paris VII
Examinateurs	:	Max DAUCHET	Université de Lille I
		Bruno DURAND	École Normale Supérieure de Lyon
		Denis POMORSKI	Université de Lille I
		Marcel STAROSWIECKI	Université de Lille I

UNIVERSITÉ DES SCIENCES ET TECHNOLOGIES DE LILLE
U.F.R. d'I.E.E.A. Bât. M3. 59655 Villeneuve d'Ascq CEDEX
Tél. 03.20.43.47.24 Fax. 03.20.43.65.66

Remerciements

Je tiens à remercier en premier lieu Max Dauchet et Bruno Durand, qui m'ont encadré avec patience et compréhension, sans ménager ni leur temps ni leurs efforts. J'adresse également mes remerciement aux professeurs A. Shen, N.K. Vereshchagin et G. Senizergues dont l'aide a été essentielle pour l'établissement de certains résultats.

Je suis très reconnaissant à Marcel Staroswiecki et Denis Pomorski de m'avoir accueilli dans leur équipe du LAIL, m'offrant ainsi un environnement de travail agréable.

J'ai une pensée particulière pour chacun des doctorants de cette équipe, leur amitié, leur soutien, leur bonne humeur quotidienne ayant une part importante dans l'aboutissement de ce travail. A ceux d'entre eux qui bientôt achèveront leurs travaux, je souhaite bon courage et bonne chance.

Sommaire

Première partie

Introduction

Introduction

Bien souvent, l'objet d'étude de l'ingénieur, du chercheur, du médecin est un système, physique ou biologique, extrêmement complexe : il nécessite pour être décrit de manière précise et exhaustive une énorme quantité d'information. Toutefois, l'objectif recherché par le spécialiste conduit celui-ci à ne s'intéresser qu'à un aspect bien particulier du système. Ainsi un chimiste et un mécanicien auront une vision partielle différente d'une même colonne de distillation. En réalité, tout se passe pour le spécialiste comme s'il était confronté à un système plus simple que le système réel, et par conséquent plus facilement utilisable, mais qui en reste suffisamment proche pour conserver, du point de vue du spécialiste, le même comportement. Ce système plus simple, appelé modèle, est généralement choisi parmi une classe de modèles que l'on sait bien manipuler : par exemple la classe de graphes, des automates... Le choix du bon modèle dans la classe est fait de façon à ce que le modèle et le système réel aient le même comportement vis à vis des phénomènes qui intéressent l'observateur. Pour obtenir cette adéquation entre le système réel et le modèle, le spécialiste procède à une identification. L'observation de la réaction du système à certaines stimulations permettent d'estimer les paramètres du modèle, de telle sorte que si celui-ci était soumis aux mêmes stimulations, il présenterait les mêmes réactions. D'une part, la réaction du système à une entrée apporte à l'observateur une certaine quantité d'information, que nous appelons information sortie par le système. D'autre part, l'ensemble des valeurs des paramètres qui permettent de fixer le modèle constitue l'information intrinsèque, information contenue dans le modèle et qui le définit complètement. L'identification consiste à mettre en relation ces deux types d'information, à exploiter l'information sortie pour retrouver l'information intrinsèque.

L'objectif initial et informel de cette thèse a été d'étudier les relations entre, d'une part, l'information intrinsèque du système et, d'autre part, l'information sortant de ce système, à laquelle on accède par l'observation. Nous nous sommes dans un premier temps intéressés à l'aspect suivant de ces relations : toute l'information contenue dans le système se retrouve-t-elle dans l'observation ? Ceci nous a amené à étudier les graphes de fonctions récursives (voir à ce propos la quatrième partie, intitulée *Observation d'un flot de données : complexité d'une fonction et complexité de son graphe*, ainsi

que [DP98]). Dans un second temps, nous nous sommes intéressés à des systèmes particuliers, se comportant comme des transducteurs rationnels déterministes, et nous avons recherché les propriétés structurelles de ces systèmes pouvant être déduites de leur observation (voir à ce propos la cinquième partie, intitulée *Séquences aléatoires et transducteurs déterministes*, ainsi que [PDDV98]).

Cette étude nécessitait un cadre mathématique permettant de donner un sens précis à certaines notions, comme la notion d'information ou la notion d'aléatoire. Nous avons choisi pour cela de nous placer dans le cadre de la théorie de la complexité de Kolmogorov.

Le terme « information » est un terme bien banal *a priori*, doté d'une signification évidente, tant son emploi quotidien est courant. Établir une définition rigoureuse et mathématique de ce concept n'est cependant pas chose aisée. Le premier a l'avoir tenté est C.E. Shannon, dans un célèbre article de 1948 [Sha48], en optant pour une approche probabiliste. Puis ce furent, de manière indépendante, les travaux de Kolmogorov [Kol65] d'une part et de Chaitin [Cha66] d'autre part, qui fondèrent les prémices d'une théorie de l'information, connue maintenant sous le nom de complexité de Kolmogorov ou encore *Théorie Algorithmique de l'Information.* Cette seconde dénomination met d'ailleurs en valeur l'approche choisie. Cette théorie définit en effet l'information contenue dans un objet comme étant la connaissance minimale nécessaire à la construction *effective* de cet objet. Nous présentons succinctement cette théorie, ainsi que celle de Shannon, dans la seconde partie, intitulée *Les théories de l'information.*

Le concept d'aléatoire est lui aussi un concept extrêmement difficile à formaliser mathématiquement. Il a été l'objet de nombreuses tentatives de définition, qui se sont révélées par la suite insatisfaisantes [Del94]. Ce n'est que lorsque le mathématicien Martin-Löf proposa sa définition que le but sembla être atteint [ML66]. Elle est en effet reconnue aujourd'hui comme la « bonne » définition de l'aléatoire. Il se trouve que la complexité de Kolmogorov permet de donner une caractérisation en termes de compression des objets aléatoires au sens de Martin-Löf, caractérisation qui peut se résumer en la formule

$$\textit{aléatoire} = \textit{incompressible.}$$

Ce cadre mathématique étant posé, nous avons mené notre étude en nous intéressant à deux classes de modèles, à savoir les fonctions récursives dans un premier temps, puis les transducteurs rationnels déterministes.

Les fonctions récursives sont des fonctions de $\mathbb{N}$ dans $\mathbb{N}$, définies partout, et calculables par ordinateur. Ceci signifie que pour chaque fonction récursive, il existe un programme d'ordinateur qui s'arrête en affichant $f(n)$ quand on lui fournit un n quelconque comme valeur d'entrée. Ces fonctions représentent bien des systèmes physiques (supposés se comporter comme un ordinateur) que l'on laisse évoluer librement et qui, de temps à autre,

provoquent un événement observable. Le n-ième événement observé est représenté par une valeur entière $f(n)$. Un tel système peut, par exemple, être un ordinateur qui effectue un calcul et produit de temps à autre une nouvelle décimale du nombre π. Dans le cadre de la complexité de Kolmogorov, il est naturel de considérer comme modèle de ce système le plus petit programme calculant f. La quantité d'information contenue dans le système est alors mesurée par la taille de ce plus petit programme.

L'observation de ce système pendant un temps infini permettrait de dessiner complètement le graphe $\mathcal{G}_f$ de la fonction, graphe dans lequel se retrouverait toute l'information contenue dans le système. Une telle observation est cependant irréaliste. En pratique, l'observateur ne peut dessiner qu'une partie $\mathcal{G}_f^n$ de ce graphe, de taille finie et correspondant aux n premières observations. L'information sortie pendant la période d'observation est celle qui est contenue dans cette portion du graphe. Nous avons choisi de la mesurer par la complexité conditionnelle de Kolmogorov $K(\mathcal{G}_f^n|n)$, longueur d'un plus petit programme qui, prenant en entrée la largeur de la fenêtre d'observation, calcule cette partie du graphe. Fournir au programme la largeur n de la fenêtre permet d'éviter d'introduire artificiellement dans cette définition l'information contenue dans n : celle-ci n'intéresse en effet aucunement l'observateur qui la choisit librement.

Nous avons pu mettre en lumière des résultats tout à fait surprenants, contraires à l'intuition que l'on peut avoir. Nous avons d'abord remarqué que, pour tous les systèmes de ce type, sauf un nombre fini, l'information sortie $K(\mathcal{G}_f^n|n)$ fluctue sans cesse en fonction de la largeur de la fenêtre d'observation et ne possède pas de limite. Quelque soit le système, elle descend infiniment souvent en dessous d'un certain seuil (indépendant du système), ceci ayant lieu quand, accidentellement, toute l'information intrinsèque est déjà encodée dans la largeur de la fenêtre d'observation ! Intuitivement, il pourrait sembler qu'en dehors de ces cas accidentels, la quantité maximale d'information sortie infiniment souvent quand la fenêtre d'observation s'étend constitue une bonne approximation de la quantité d'information contenue dans le système. Cette intuition s'avère fausse, comme nous le montrons dans le théorème 4.4. Dans toute famille « raisonnable » de systèmes, il est en effet toujours possible de trouver un système pour lequel l'information sortie est arbitrairement plus petite que l'information contenue dans le système. Il s'avère toutefois que ces cas sont rares, comme nous le montrons dans le théorème 4.5. Pour la plupart des systèmes d'une famille « raisonnable », l'information maximale sortie constitue donc bien une bonne approximation de la quantité d'information intrinsèque au système, et l'intuition est finalement sauve.

Ces résultats n'auraient pu être obtenus sans l'aide du Professeur Alexander Shen et du Professeur Nikolaï Vereshchagin à l'origine, respectivement, des preuves des théorèmes 4.3 et 4.5. Nous tenons à les en remercier ici vivement.

Les fonctions récursives représentent des systèmes sans entrées qui produisent des événements sans aucune contrainte sur l'intervalle de temps séparant l'apparition de deux événements successifs. Cependant, un système réel est généralement soumis à un flux d'entrées $e_1,...,e_n...$ Sa réaction à ces sollicitations se traduit par un certain flux de sorties observées $s_1,...,s_n...$ Dans le cas où deux événements de sortie successifs se produisent dans un intervalle de temps borné, ces systèmes se comportent comme des transducteurs rationnels déterministes (voir la figure 1).

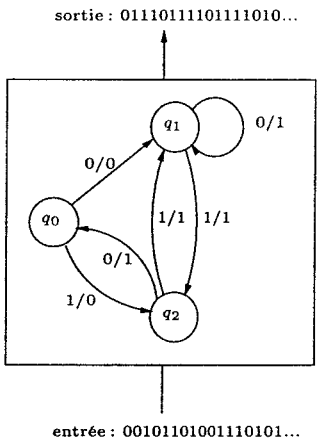


FIG. 1: *Un exemple de transducteur avec son flux d'entrée et son flux de sortie. L'état initial est* q_0.

Il s'agit de graphes orientés dont chaque sommet correspond à un état du système. À chaque fois qu'une entrée survient, le système est susceptible de changer d'état et de provoquer un événement en sortie. Dans le transducteur, le passage d'un état à un autre est représenté par un arc orienté, appelé transition. À partir d'un état, plusieurs états peuvent être atteints. Le choix de l'état à atteindre se fait de façon déterministe à partir de l'entrée. Pour cela, chaque transition est munie d'une étiquette d'entrée. Quand une entrée survient, la transition étiquetée par cette entrée détermine le nouvel état du système. Chaque transition est également munie d'une étiquette de sortie, qui indique la sortie du transducteur quand celui-ci change d'état en empruntant cette transition. Il peut arriver que le transducteur soit dans un état où aucune transition n'est étiquetée par l'entrée qui survient. Dans ce cas le transducteur se bloque. Il est à noter que nous nous sommes limités dans notre étude à des transducteurs lettre à lettre, c'est-à-dire produisant exactement une lettre en sortie pour chaque lettre d'entrée, une lettre correspondant à l'occurrence d'un événement.

Pour ce type de système nous nous sommes intéressés à l'aspect qualitatif de la relation entre information intrinsèque et information sortie : en

quoi cette dernière nous renseigne-t-elle sur la première? Que peut-on apprendre de la structure interne du système (*i.e.* du transducteur sous-jacent) en observant son comportement, c'est-à-dire sa réaction (sa sortie) à certains stimuli (son entrée)? La sortie dépend bien sûr de la structure interne du transducteur, mais elle dépend également de l'entrée. Si l'on veut pouvoir interpréter des régularités détectées dans la sortie en termes de caractéristiques structurelles du transducteur, il faut être certain que ces régularités ne peuvent pas être une conséquence directe de régularités déjà présentes dans l'entrée. Cette remarque nous a conduits à étudier le comportement des transducteurs sur des flux d'entrée particuliers, ne comportant aucune sorte de régularité. Les flux d'entrée répondant à ce critère sont les flux aléatoires, comme nous l'apprend la complexité de Kolmogorov.

Tous les transducteurs ne sont pas capables d'accepter des flux d'entrée aléatoires. Notre premier résultat (voir théorème 5.1) est une condition, nécessaire et suffisante, portant sur la structure du transducteur, pour qu'il accepte un flux aléatoire donné. Ce résultat a deux corollaires intéressants. Tout d'abord (voir théorème 5.2), il est possible de déterminer, par programme et en un temps fini, si deux transducteurs produisent une même sortie pour une même séquence aléatoire d'entrée. Ceci a une conséquence sur l'identification du système: observons un système, que nous savons se comporter comme un transducteur ayant un nombre d'états inférieur à une certaine borne, borne que nous connaissons. Nous sommes alors capables, par l'observation de la sortie qu'il produit sur un flux d'entrée aléatoire, de construire de manière effective un certain nombre de transducteurs ayant le même comportement que le système observé sur ce flux d'entrée particulier.

Toutefois ce résultat n'exploite que le fait suivant: le transducteur accepte le flux aléatoire d'entrée. Or l'observation de la forme de la sortie peut nous apporter davantage de renseignements sur la structure du système. Rappelons-nous en effet que l'idée de départ était d'utiliser les régularités présentes dans la sortie (ne pouvant trouver leur origine que dans la structure du transducteur) pour essayer d'en déduire des propriétés du système. Différents cas peuvent être envisagés. Le système peut transférer toute l'information de l'entrée vers la sortie. Comme il y a synchronisme (une sortie sortie pour une entrée survenue), cela signifie que la sortie elle-même est aléatoire. Mais le système peut également ne transférer qu'une partie de l'information de l'entrée vers la sortie, le reste de l'information étant définitivement perdu, effacé. La sortie n'est plus alors aléatoire. Selon la proportion d'information effacée, deux nouveaux cas peuvent être envisagés. Si le transfert d'information se fait en proportion suffisante, la sortie est non-récursive. Si toute l'information est perdue, la sortie est récursive. Nous avons établi (voir le théorème 5.3) que ces différents cas correspondent en fait à la présence de certains types d'états dans le transducteurs: d'une part les états qui transfèrent de l'information (que nous appelons *états discriminants*), d'autre part les états qui effacent l'information (que nous appelons *états partiellement*

indifférents). L'absence d'état partiellement indifférent garantit que la sortie est aléatoire, tandis que la présence l'en empêche. Dans ce dernier cas, la présence d'un état discriminant permet de conserver le caractère non-récursif en transférant suffisamment d'information. Enfin l'absence d'état discriminant empêche tout transfert d'information : la sortie est alors récursive, et même plus précisément ultimement périodique. Les figures 2, 3 et 4 illustrent ces différents cas.

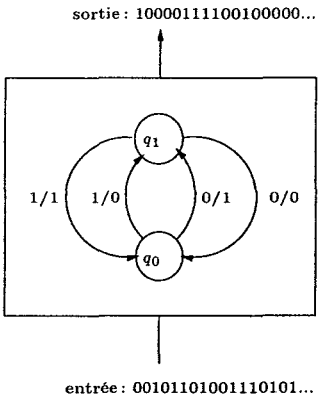


FIG. 2: *Un transducteur sans état partiellement indifférent. L'état initial est* q_0.

Dans le transducteur de la figure 2, aucun état n'est partiellement discriminant, toute l'information de l'entrée est transférer vers la sortie : une lettre sur deux est complémentée, les autres lettres restent identiques.

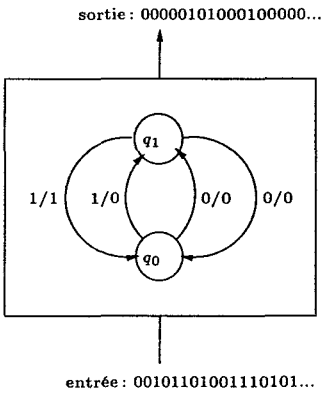


FIG. 3: *Un transducteur ayant un état partiellement indifférent* (q_0) *et un état discriminant* (q_1). *L'état initial est* q_0.

Dans le transducteur de la figure 3, une lettre sur deux est remplacée par un 0 (perte d'information) les autres lettres restent identiques (conservation

de l'information). La sortie n'est pas aléatoire mais reste non récursive.

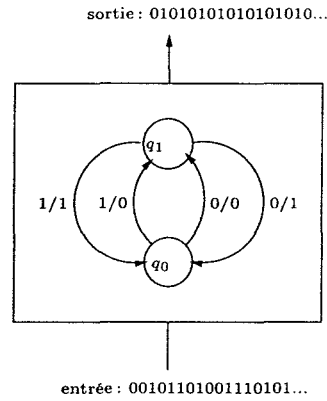


FIG. 4: *Un transducteur avec deux états partiellement indifférents et sans état discriminant. L'état initial est* q_0.

Enfin, dans le transducteur de la figure 4, toute l'information est perdue : les lettres de rang pair sont systématiquement remplacées par 1, les lettres de rang impair par 0. La sortie est périodique.

Notons enfin que dans le cas où le transducteur est connu et où l'on s'interroge sur la forme de sa sortie en présence d'un flux d'entrée aléatoire, la présence d'un état partiellement indifférent et celle d'un état discriminant sont décidables en un temps polynômial en le nombre d'états du transducteur. Nous remercions ici le Professeur G. Senizergues, à l'origine de la preuve pour le cas des états discriminants.

Il serait intéressant de prolonger cette étude par l'examen du cas plus général des transducteurs non lettre à lettre. Enfin, il nous semble que des liens avec les chaînes de Markov restent à établir.

Le cadre général établi dans la partie portant sur les graphes de fonctions récursives n'est pas effectif : le plus petit modèle n'est pas calculable. Cependant, il permet de mettre en évidence les limites de l'approche « boîte noire » des systèmes : l'observation du système ne donne pas toujours accès à toute l'information contenue dans celui-ci. L'approche purement informationnelle que nous avons choisie, fait l'impasse sur certaines notions, telles la stabilité, la continuité. C'est une approche limitée, mais qui offre la possibilité d'avoir un regard nouveau sur certains domaines. Citons l'exemple de la thermodynamique (voir à ce sujet les travaux de Zurek [Zur89]), domaine fondamental que cette théorie contribue à éclairer. Pour illustrer cet autre regard qu'offre la complexité de Kolmogorov, nous examinons dans la troisième partie, intitulée *Géométrie et complexité* un exemple de résultat qui est une interprétation, en termes purement géométriques, de résultats établis dans le cadre de cette théorie. Nous montrons également que cette approche informationnelle coïncide avec une approche plus usuelle de l'analyse de données

qu'est la régression linéaire.

Deuxième partie

Le concept d'information

Chapitre 1

Introduction

Nous présentons dans cette partie deux approches théoriques entreprises dans le but de cerner le concept intuitif d'information. Historiquement, la première tentative est due à Shannon [Sha48]. Il s'agit là d'une théorie probabiliste de l'information, réalisée dans le cadre du traitement du signal. Les bases de la seconde approche, optant pour un point de vue algorithmique, ont été établies indépendamment par R.J. Solomonoff [Sol64], A.N. Kolmogorov [Kol65] et G.J. Chaitin [Cha66] au milieu des années 60. Pour une présentation générale de la complexité de Kolmogorov on peut se reporter à [Cal94] [LV97] [Dub98].

De manière générale, l'information est échangée entre une *source* et une *destination*, par l'intermédiaire d'un *message* se propageant dans un *canal*. Nous nous limitons ici au cas d'un canal non bruité : le contenu du message n'est pas altéré par des événements extérieurs lors de son cheminement. Un observateur A situé à la source constate la survenue d'un événement, rédige un message et l'envoie au destinataire B, *via* le canal, dans le but d'amener B à une connaissance plus ou moins complète de l'événement.

L'ensemble des événements possibles $\mathcal{E} = \{e_i\}$ est un ensemble dénombrable (fini ou infini). Une probabilité $Pr(e_i)$ peut éventuellement être associée à chacun de ces événements (cas de l'approche de Shannon). Dans le cas contraire (cas de l'approche de Kolmogorov), on parlera d'objets plutôt que d'événements.

Afin de formaliser le problème, les événements (ou les objets) sont identifiés à des mots définis sur l'alphabet $\{0,1\}$, mots que nous appellerons *mots-objet*. Ainsi, e_i désigne indifféremment un événement, un objet ou un mot-objet. Un cas particulier important est celui où les objets sont des entiers. Ceux-ci sont identifiés à des mots-objet de la façon suivante : 0 est identifié au mot vide ϵ, 1 est identifié à 0, 2 à 1, 3 à 00, 4 à 01, etc. Cela revient à classer les mots d'abord par longueur puis, pour tous les mots de même longueur, par ordre lexicographique.

Codage de l'information L'observateur porte un mot-objet e_i à la connaissance du destinataire en lui transmettant un *mot-code* w_i défini sur l'alphabet $\{0, 1\}$. La fonction qui associe un mot-code à chacun des événements est appelée *fonction de codage*, tandis que l'ensemble des mots-code d'une telle fonction forme un *code*. Le destinataire procède à une opération de décodage du mot-code pour retrouver le mot-objet originel. Si l'observateur veut rendre compte d'une suite de mots-objet $e_{i_1}, e_{i_2}, \ldots$, alors il concatène tous les mots-code associés à chacun de ces mots-objet et envoie le mot résultant.

Prenons l'exemple d'un dé à six faces. Les événements possibles sont l'obtention de 1, 2, 3, 4, 5 ou 6, événements que nous codons à l'aide des mots-code 0, 1, 00, 01, 10 et 11. Pour transmettre le résultat de trois lancers de dé, par exemple 1, 4 et 2, l'observateur transmet la concaténation des mots 0, 01 et 1, c'est-à-dire le mot 0011.

Pour que le destinataire puisse retrouver le mot-objet à partir du mot-code transmis, il est nécessaire que la fonction de codage associe des mot-codes différents à des événements différents, *i.e.* soit injective. C'est bien le cas de l'exemple précédent.

Codes à décodage unique Il est naturel d'imposer que l'observateur soit capable de décoder non seulement un mot-code isolé, mais également la succession de mots-code de plusieurs événements successifs. Des codes permettant ceci sont dits *codes à décodage unique*. Ce n'est pas le cas de l'exemple que nous avons donné précédemment. En effet, le mot 0011 peut être interprété de deux façons différentes :

- comme la concaténation des mots 00 et 01, codant les résultats de deux lancers de dés, 3 et 4 ;
- comme la concaténation des mots 0, 0 et 11, codant les résultats de trois lancers de dés, 1, 1 et 6 ;
- etc.

En revanche la fonction de codage suivante possède un code à décodage unique :

$$1 \to 0, 2 \to 01, 3 \to 011, \ldots, 6 \to 011111,$$

l'apparition d'un 0 avertissant du début d'un nouveau mot-code.

Codes préfixes Un code à décodage unique peut toutefois comporter un certain défaut, à savoir la nécessité (comme dans l'exemple que nous venons de donner), d'attendre le début du mot suivant pour pouvoir décoder le mot courant. Ceci a lieu lorsque certains mots-code sont préfixes d'autres mots-code.

Un code ne souffrant pas de ce défaut, *i.e.* dont aucun mot-code n'est le préfixe d'un autre mot-code, est dit *code préfixe* ou encore *code à décodage instantané*. C'est le cas du code défini par la fonction de décodage suivante :

$$1 \to 0, 2 \to 10, 3 \to 110, \ldots, 6 \to 111110.$$

Nous présentons dans le chapitre 2 l'approche de Shannon avec la notion d'entropie puis l'approche de Kolmogorov avec la notion de complexité algorithmique. Pour chacune de ces approches, nous tentons d'expliciter les concepts intuitifs d'information sous-jacents et nous montrons comment la notion de code permet de formaliser et de quantifier l'information. Dans le chapitre 3, nous examinons le concept d'*aléatoire*, concept que la complexité de Kolmogorov a contribué à éclairer.

Chapitre 2

Les théories de l'information

2.1 L'approche de Shannon

Pour établir sa théorie de l'information, Shannon a considéré des événements e_i survenant avec une probabilité $Pr(e_i)$. L'information apportée par un événement est véhiculée par le mot-code transmis au destinataire. Mais que représente cette information ?

2.1.1 L'information selon Shannon

Imaginons le cas où les événements sont le résultat du lancer d'un dé à six faces, éventuellement pipé. La seule connaissance *a priori* que B possède est la distribution de probabilité. A lance le dé et obtient un six. Avant d'avoir connaissance du résultat, B attribue *a priori* un certain degré d'impossibilité à chacun des événements, degré d'autant plus élevé que l'événement est improbable. Ce degré quantifie la croyance que B a en la réalisation de l'événement. A transmet à B un mot-code signifiant « J'ai obtenu un six. ». Après avoir pris connaissance du mot-code, B sait quel est l'événement qui est survenu, et par conséquent lui attribue *a posterori* un degré d'impossibilité minimal. L'information véhiculée par le mot-code a provoqué une variation du degré d'impossibilité que B attribue à l'événement. Il est naturel d'identifier la quantité d'information véhiculée par le mot-code avec la variation du degré d'impossibilité qu'elle provoque chez le destinataire. À la limite, un événement de probabilité nulle qui survient apporte beaucoup d'information : on ne s'attend pas à la réalisation d'un tel événement, et par conséquent on lui attribue *a priori* un degré d'impossibilité maximal. *A contrario* un événement de probabilité 1 qui survient n'apporte pas d'information : on s'y attend de façon certaine et on lui attribue *a priori* un degré d'impossibilité minimale.

On a donc une interprétation de l'information véhiculée en terme de variation du degré d'impossibilité. Fixons le degré d'impossibilité d'un événement certain à 0. L'information est alors égale au degré d'impossibilité

attribué *a priori* avant la prise de connaissance de l'événement, degré qui est une fonction décroissante de sa probabilité. Peut-on préciser davantage cette dépendance?

Considérons le cas de deux événements indépendants. Par exemple, deux lancers successifs d'un dé. La probabilité de l'événement composé est le produit des probabilités des événements élémentaires. Chacun de ces événements élémentaires provoque une variation du degré d'impossibilité que leur attribue le destinataire. Il est naturel d'imposer que la variation du degré d'impossibilité concernant l'événement composé soit égale à la somme des variations du degré d'impossibilité pour chacun des événements élémentaires.

Ces considérations amènent à définir la quantité d'information $I(e_i)$ apportée par la réalisation d'un événement e_i de la façon suivante :

$$I(e_i) = -\log(Pr(e_i)).$$

On vérifie bien ainsi que l'information apportée est une fonction décroissante de la probabilité de l'événement, prenant une valeur nulle pour un événement certain, et respectant la propriété d'additivité.

2.1.2 Information et codes

En quel sens un code peut-il refléter l'information ainsi définie? Pour répondre à cette question, il nous faut examiner la notion d'entropie.

Définition 2.1 *L'entropie de Shannon $H(P)$ d'un ensemble d'événements se produisant selon une distribution de probabilité P est l'espérance de l'information apportée par la réalisation d'un événement :*

$$H(P) = \sum P(e_i)I(e_i).$$

Une fonction de codage f et une distribution de probabilité P étant données, l'espérance de la longueur d'un mot-code est $L(f,P) = \sum P(e_i)l(f(e_i))$. Les codes préfixes pour lesquels cette espérance est minimale sont les codes optimaux.

Définition 2.2 *Une distribution de probabilité P étant fixée, un code préfixe défini par une fonction de codage f est dit optimal si $L(f,P)$ est minimal.*

De tels codes (codes de Huffmann, code de Fano) associent simplement aux événements des mots-code d'autant plus courts que ces événements sont plus probables. Le théorème de codage en l'absence de bruit[1] de Shannon

1. Ainsi appelé car le canal de transmission transporte l'information sans la perturber.

fait le lien entre d'une part l'information moyenne transportée par le code et d'autre part la longueur moyenne des mots-code.

Théorème 2.1 (Shannon) *Pour toute fonction de codage f définissant un code préfixe optimal,*

$$H(P) \leq L(f,P) < H(P) + 1.$$

2.2 L'approche de Kolmogorov

2.2.1 La complexité de Kolmogorov

L'approche de Shannon a permis de définir une notion d'information apportée par l'occurrence d'un événement. Cette notion est toutefois davantage liée à la probabilité de l'événement qu'à l'événement lui-même. Un même événement peut apporter une quantité d'information plus ou moins grande suivant la probabilité avec laquelle il apparaît.

Il est naturel de se demander s'il n'existe pas une notion d'information qui soit intrinsèque à un événement ou, plus généralement à un objet, indépendamment de toute distribution de probabilité sur ces objets. C'est cette question qui est à l'origine de la théorie de la complexité de Kolmogorov.

Nous parlerons désormais de la complexité d'un objet, plutôt que de celle d'un événement, les objets étant identifiés à des mots de l'alphabet $\{0,1\}$. Un cas particulier important est celui où les objets considérés sont des entiers. Une première façon de les identifier à des mots binaires est de procéder comme suit : l'entier 0 est identifié au mot vide ϵ, 1 à 0, 2 à 1, 3 à 00, 4 à 01... Cela revient à classer les mots d'abord par longueur puis pour tous les mots de même longueur dans l'ordre lexicographique. On confondra par la suite l'entier x et le mot binaire qui le code de cette façon et on notera $l(x)$ le nombre de bits de ce mot. Pour tout entier x, $l(x) = E(\log_2(x+1))$, E désignant la partie entière. Une autre façon de faire consiste à utiliser des mots formant un code préfixe. On parle alors d'écriture préfixe ou autodélimitée. Par exemple, un entier x peut être représenté par un mot $\bar{x} = 1^{l(x)}0x$. Le nombre de bits nécessaires à cette écriture est $l(\bar{x}) = 2l(x) + 1$. Cela permet de représenter un n-uplet d'entiers $(x_1, \ldots, x_n)$ par un mot $\langle x_1, \ldots, x_n \rangle$, en concaténant simplement les écritures préfixes de chacun des éléments :

$$\langle x_1, \ldots, x_n \rangle = \bar{x}_1 \bar{x}_2 \ldots \bar{x}_n.$$

La notion d'information est abordée ici d'un point de vue algorithmique. La définition de la complexité se fonde sur les machines de Turing. Différentes définitions de la complexité peuvent être envisagées, selon les machines que l'on considère.

1. Les machines considérées peuvent être les machines calculant les fonctions partielles récursives. Cela permet de définir une complexité dite

complexité simple, première, historiquement à avoir été envisagée. Les programmes d'une telle machine (un programme est un mot d'entrée délimité par des blancs qui amène la machine dans un état d'arrêt), ne constituent pas nécessairement un code à décodage unique.

2. Les machines considérées peuvent être les machines calculant des fonction partielles récursives préfixes : de deux programmes différents d'une telle machine, aucun n'est préfixe de l'autre. L'ensemble des programmes d'une telle machine forme un code préfixe. Cela permet de définir une complexité dite *complexité préfixe.*

Dans tous les cas, la complexité est définie de la façon suivante.

Définition 2.3 *La complexité conditionnelle de Kolmogorov $K_\phi(x|y)$ de x sachant y, relativement à une machine ϕ, est la longueur d'un plus petit programme, s'il existe, qui, exécuté sur la machine ϕ et ayant en entrée y, calcule x. Si un tel programme n'existe pas, $K_\phi(x|y)$ vaut l'infini.*

$$K_\phi(x|y) = \min\{l(p), \phi\langle p, y\rangle = x\}$$

On se donne donc une machine ϕ. On dispose éventuellement d'une information partielle *a priori* de x, information représentée par la donnée de y (dans le cas où l'on ne dispose d'aucune connaissance *a priori*, y est le mot vide). La quantité d'information contenue dans x est alors vue comme le plus petit moyen effectif (*i.e.* le plus petit programme s'exécutant sur la machine ϕ et disposant de la connaissance *a priori* y), de reconstruire x.

Ainsi définie, la notion d'information est étroitement liée à la machine ϕ choisie. Existe-t-il une machine « meilleure » que toute les autres, en ce sens que la complexité d'un objet relativement à cette machine soit plus petite que la complexité du même objet relativement à tout autre machine? La réponse est négative. Il suffit pour s'en convaincre de remarquer que pour tout objet x, il est toujours possible de considérer la machine qui produit x quelque soit son entrée : la complexité de x relativement à cette machine est nulle. Toutefois, certaines machines permettent d'obtenir un résultat proche : il s'agit des machines *additivement optimale.*

Définition 2.4 *Une machine ϕ est additivement optimale si*

$$\forall\ \phi'\ \exists\ C_{\phi'}\ \forall\ x, y\ K_\phi(x|y) \leq K_{\phi'}(x|y) + C_{\phi'}.$$

Dans tout système de programmation raisonnable, *i.e.* pour toute famille $(\phi_i)_{i\in\mathbb{N}}$ de machines de Turing vérifiant les propriétés suivantes :

1. toute fonction partielle récursive est calculable par une machine ϕ_i,
2. il existe une machine universelle :

$$\exists\ u\ \forall\ x, y\ \phi_u\langle x, y\rangle = \phi_x(y),$$

3. il existe une fonction de composition totale récursive c :

$$\forall\, i,j\ \phi_i \circ \phi_j = \phi_{c\langle i,j\rangle},$$

on est assuré de l'existence d'une machine additivement optimale.

Définir la complexité par rapport à une machine additivement optimale assure la propriété suivante : la complexité d'un objet quelconque relativement à une autre machine donnée ne peut être arbitrairement plus petite. Les « bonnes » machines pour définir la notion de complexité sont les machines additivement optimales. Changer de machine (additivement optimale) de référence ne modifie la complexité d'un objet quelconque que d'une constante additive (indépendante de l'objet, mais dépendant des deux machines de référence). C'est ce qu'exprime le théorème suivant.

Théorème 2.2 *Pour toutes machines de Turing additivement optimale ϕ_1 et ϕ_2,*

$$\exists\, C_{\phi_1,\phi_2}\ \forall\, x,y\ |K_{\phi_1}(x|y) - K_{\phi_2}(x|y)| \leq C_{\phi_1,\phi_2}.$$

Cela tient du fait que la donnée d'un programme interpréteur, écrit dans le langage de la machine ϕ_2 et interprétant les programmes écrit dans le langage de la machine ϕ_1, ainsi que d'un programme calculant x sur ϕ_1, constitue un programme de la machine ϕ_2 calculant x. La constante intervenant dans l'inégalité précédente représente en fait la taille de cet interpréteur.

Ce résultat permet de s'affranchir de la machine de référence choisie. Par la suite, nous supposerons donc fixée une machine additivement optimale, et l'indice ϕ dans la notation de la complexité sera négligé. En outre, $K(x|\epsilon)$, à savoir la complexité de x en l'absence de toute information *a priori*, sera notée plus simplement $K(x)$.

2.2.2 Propriétés

Une majoration simple de la complexité d'un objet x est donnée par la longueur de cet objet. Ainsi,

$$\forall\, x\ K(x) \leq l(x) + \mathcal{O}(1), \text{pour la complexité simple,} \tag{2.1}$$

$$\forall\, x\ K(x) \leq 2l(x) + \mathcal{O}(1), \text{pour la complexité préfixe.} \tag{2.2}$$

Dans le cas de la complexité simple, il suffit de considérer la machine de Turing particulière qui se contente de recopier son entrée en sortie.

Pour la complexité préfixe, il est nécessaire que les programmes de la machine forment un ensemble préfixe. Considérons la machine de Turing qui lit le nombre n de 1 successifs de son entrée, ignore le 0 qui suit, puis produit en sortie le mot formé des n bits suivants. Les programmes de cette machine sont les mots $\bar{x} = 1^{l(x)}0x$, de longueur $2l(x)+1$, et qui forment un ensemble

préfixe. L'exécution du programme $\bar{x}$ sur cette machine produit en sortie x. Par conséquent $K(x) \leq 2l(x) + \mathcal{O}(1)$.

Dans le cas de la complexité préfixe, cette majoration peut être affinée en employant d'autres écritures préfixes. Considérons par exemple la machine de Turing effectuant les opérations suivantes : elle compte le nombre n de 1 successifs apparaissant au début de son entrée. Elle ignore le 0 qui suit, puis lit le mot m formé des n bits suivants. Enfin, elle produit en sortie le mot formé des m bits suivants. Les programmes de cette machine sont les mots $w(x) = 1^{l(l(x))}0l(x)x$, de longueur $l(x) + 2l(l(x)) + 1$, et qui forment un ensemble préfixe. Le résultat de l'exécution du programme $w(x)$ sur cette machine est x. Par conséquent, $K(x) \leq l(x) + \mathcal{O}(l(l(x))$.

La complexité de Kolmogorov n'est pas calculable. Toutefois, la simulation en parallèle de tous les programmes de la machine de référence choisie permet de découvrir des programmes de plus en plus courts calculant un x donné. Cette simulation se fait de la façon suivante. x est donné en entrée. Le premier pas du premier programme est simulé, puis le premier pas du second programme et le second pas du premier, puis le premier pas du troisième programme, le second pas du second et le troisième pas du premier, et ainsi de suite. Quand un des programmes simulés s'arrête, sa sortie est comparée à x. S'il y a égalité, on produit en sortie le programme qui s'est arrêté, puis on reprend la simulation, mais en se limitant aux programmes de taille inférieure au programme trouvé. On énumère ainsi des programmes de plus en plus courts calculant x.

Chapitre 3

Le concept d'aléatoire

Nous abordons dans ce chapitre le concept de *séquence aléatoire*, le terme séquence désignant toute suite infinie de lettres de l'alphabet $\{0, 1\}$.

La formalisation de ce concept a pris beaucoup de temps pour se stabiliser. Le premier à avoir proposé une définition a été Von Mises, en 1919. Pour lui, une séquence aléatoire est une séquence telle que dans toute suite extraite par un « procédé raisonnable », les proportions de 0 et de 1 sont, à la limite, les mêmes. Cette approche recèlait toutefois une difficulté, à savoir le sens précis donné à l'expression « procédé raisonnable ». Il s'est avéré au cours du temps qu'aucune définition ne pouvait conduire à l'obtention d'une notion satisfaisante de séquence aléatoire.

Les statisticiens étudient le caractère aléatoire de séquences en leur faisant passer des tests statistiques. L'application d'un test statistique consiste à vérifier si une séquence satisfait ou non une loi statistique, toute « bonne » séquence aléatoire se devant de satisfaire cette loi. On peut exiger, par exemple, que dans toute séquence aléatoire il y ait équiproportion de 0 et de 1, ou de 00, 01, 10, 11, etc. Une séquence échouant à un test est déclarée de façon certaine non aléatoire. Martin-Löf a développé cette idée dans [ML66], et a formalisé le concept de test, celui-ci désignant tout test statistique envisageable et applicable de manière effective.

Une autre approche, envisagée dans un premier temps par Kolmogorov avec la complexité simple, puis par Chaitin avec la complexité préfixe, a consisté à définir une séquence aléatoire comme une séquence incompressible, c'est-à-dire une séquence dont aucun mot préfixe ne peut être calculé par un programme significativement plus petit.

Intuitivement, une séquence est aléatoire s'il nous apparaît que la cause la plus probable de cette séquence est le lancer d'une pièce équilibrée. Observons les séquences suivantes.

- $s_1 = 00000000000000000000...$ séquence ne comportant que des 0 ;
- $s_2 = 00100100001111110110...$ début du développement binaire de π ;

- s_3 = 01001101110010001011... codant la succession des pixels blancs ou noirs d'une image de téléviseur déréglé (en supposant qu'il y ait autant de pixels blancs que de pixels noirs);
- s_4 est la séquence obtenue à partir de s_3 en dupliquant chaque bit.

Il ne viendrait à l'idée de personne de déclarer que s_1 est une séquence aléatoire. Il paraît en effet déraisonnable de penser que cette séquence ait été obtenue par tirages successifs de pile ou face, plutôt que par un processus recopiant inlassablement la lettre 0.

En ce qui concerne la séquence s_2, la situation est un peu moins claire. À première vue, il ne semble pas choquant de qualifier cette séquence d'aléatoire. Rien dans l'observation de cette séquence ne semble suggérer l'idée d'une quelconque régularité. Cependant, si l'on déclare à l'observateur qu'il s'agit là du début du développement binaire du nombre π, il lui deviendra tout à coup beaucoup plus naturel de supposer que cette séquence a été obtenue par un processus en lien étroit avec le nombre π plutôt que par le lancer d'une pièce.

Ces deux séquences, pouvant être engendrées par des processus de calculs sans entrées fonctionnant indéfiniment, sont des séquences récursives.

Les deux dernières séquences représentatives d'un phénomène de bruit ne sont probablement pas récursives. Sont-elles aléatoires?

En ce qui concerne s_3, il paraît raisonnable de supposer que chaque pixel est tiré au hasard, indépendamment de tous les autres. Nous sommes donc plutôt enclin à la déclarer aléatoire.

Pour s_4, ce n'est pas le cas. Bien sûr un bit sur deux paraît tiré au hasard. Mais il nous paraît plus raisonnable de penser que l'origine de cette séquence est un processus de calcul, ayant cette fois une entrée, dans notre cas constituée de la séquence s_3, qui duplique chaque bit. Bien que cette séquence soit non-récursive, elle ne nous apparaît pas aléatoire.

Il semble donc bien que le caractère aléatoire d'une séquence dépende étroitement de l'existence d'une description « courte ».

La validité des deux approches de Martin-Löf et de Chaitin a été considérablement renforcée quand Levin [Lev73] et Schnorr [Sch73] ont démontré qu'elles conduisaient à définir des classes identiques de séquences aléatoires [1].

3.1 L'aléatoire selon Martin-Löf

Martin-Löf a fondé sa définition de séquence aléatoire sur l'idée de *test*. Un test est un procédé (devant satisfaire certaines propriétés de calculabilité)

1. Plus exactement, Levin et Schnorr ont démontré que les séquences aléatoires au sens de Martin-Löf et que les séquences incompressibles au sens d'autres versions de la complexité (différentes de la complexité simple et de la complexité préfixe) sont les mêmes, mais leur résultat peut s'étendre au cas de la complexité préfixe. Pour ce qui concerne les relations entre ces différentes variantes de la complexité, se reporter à [US96] [USS90].

qui élimine toute séquence possédant une régularité que l'on refuse de trouver dans une séquence aléatoire. Une séquence échouant au test contient cette régularité et ne peut être considérée comme aléatoire. Une séquence passant le test ne contient pas cette régularité. Toutefois, cela n'assure pas qu'elle soit aléatoire : il est toujours possible qu'elle contienne un autre type de régularité. Martin-Löf a montré qu'il existe un test universel, capable de détecter toute forme de régularité, qui élimine toute séquence non aléatoire et ne conserve que les séquences aléatoires.

Une séquence est aléatoire, au sens de Martin-Löf, si et seulement si elle n'appartient à aucun ensemble constructivement de mesure nulle. Plus précisément, nous avons la définition suivante.

Définition 2.5 *Une séquence s est non aléatoire si et seulement s'il existe une fonction récursive $u(i,j)$, qui associe un mot à tout couple d'entiers, telle que :*

1. $\forall\ i\ \mu(\Omega_i) \leq 2^{-i}$;
2. $\forall\ i\ s \in \Omega_i$.

$\Omega_i = \bigcup_j \Gamma_{u(i,j)}$, $\Gamma_{u(i,j)}$ *désignant l'ensemble de toutes les séquences ayant* $u(i,j)$ *comme préfixe, et* μ *désignant la mesure uniforme sur l'ensemble des séquences : pour tout mot* w, $\mu(\Gamma_w) = 2^{-l(w)}$.

Par exemple une séquence s dont les lettres de rang pair sont 0 est éliminée par le test suivant. Nous prenons comme fonction $u(i,j)$ la fonction qui à tout couple d'entiers (i,j) associe le mot de longueur i dont toutes les lettres de rang pair sont 0 et dont les lettres de rang impair sont les $\frac{i}{2}$ premiers bits du j-ième mot de longueur au moins $\frac{i}{2}$. Cette fonction est bien récursive. Pour tout i, Ω_i est l'ensemble des mots de longueur i dont toutes les lettres de rang pair sont 0 : $\mu(\Omega_i) = 2^{-\frac{i}{2}}$. Pour tout i, $s \in \Omega_i$. Par conséquent s est éliminé par ce test et est jugé non aléatoire.

3.2 L'aléatoire selon la théorie de la complexité

L'approche de Kolmogorov et de Chaitin repose sur l'idée que toute régularité présente dans une séquence peut être exploitée pour comprimer cette séquence. Par exemple, une séquence s dont les lettres de rang pair sont 0 possède des préfixes très compressibles. Pour retrouver les n premières lettres d'une telle séquence, il suffit de donner les $\frac{n}{2}$ premières lettres de rang impair (sous forme préfixe, ce qui demande au plus $\frac{n}{2} + \log(\frac{n}{2})$ bits) et un programme, de longueur constante, qui intercale un 0 entre chacune de ces lettres. Les séquences aléatoires, qui ne comportent aucune régularité,

apparaissent comme les séquences incompressibles. C'est ce qu'établit plus précisément le théorème suivant, dû à Schnorr [Sch73].

Théorème 2.3 *Une séquence s est aléatoire si et seulement si tous ses préfixes sont compressibles de moins d'une constante,* i.e.

$$\exists\, c\, \forall\, n\ K(s_{1:n}) \geq n - c,$$

K désignant ici la complexité préfixe.

Dans la cinquième partie, nous étudions des transducteurs dont les entrées sont des séquences aléatoires. Dans cette partie, certaines preuves font usage du résultat suivant : une séquence s n'est pas aléatoire s'il existe un programme (auto-délimité) lisant en entrée, lettre à lettre, une autre séquence s', et n'utilisant au plus que les λn premières lettres de s' pour calculer les n premières lettres de s, avec $\lambda < 1$. C'est ce résultat important que nous établissons maintenant.

Lemme 2.1 *Soit s et s' deux séquences. S'il existe un programme P, une suite $(u_n)_{n\in\mathbb{N}}$, une suite $(v_n)_{n\in\mathbb{N}}$ strictement croissante et $\lambda < 1$ tels que*

$$\forall\, n\ P(s'_{1:u_n}) = s_{1:v_n} \text{ et } u_n \leq \lambda v_n,$$

alors s n'est pas aléatoire.

Preuve. Nous allons donner une majoration de la complexité préfixe des mots $s_{1:v_n}$. Pour tout n, le mot $w_n = P1^{l(l(s'_{1:u_n}))}0l(s'_{1:u_n})s'_{1:u_n}$ est un programme auto-délimité qui permet de calculer $s_{1:v_n}$. Par conséquent, nous avons :

$$\begin{aligned}
\forall\, n\ K(s_{1:v_n}) &\leq l(w_n) + \mathcal{O}(1) \\
&= l(s'_{1:u_n}) + 2l(l(s'_{1:u_n})) + \mathcal{O}(1) \\
&= u_n + 2l(u_n) + \mathcal{O}(1) \\
&= u_n + 2\log(u_n) + \mathcal{O}(1) \\
&\leq \lambda v_n + 2\log(v_n) + \mathcal{O}(1) \text{ car } u_n \leq \lambda v_n.
\end{aligned}$$

Soit $c > 0$ fixé. Comme $\lambda < 1$, il existe n_c tel que

$$\forall\, n \geq n_c,\ \lambda v_n + 2\log(v_n) + \mathcal{O}(1) \leq v_n - c.$$

Finalement, nous obtenons :

$$\forall\, c\, \exists\, n_c\, \forall\, n \geq n_c,\ K(s_{1:v_n}) \leq v_n - c.$$

Comme la suite (v_n) est strictement croissante, cela implique que s n'est pas aléatoire. □

Troisième partie

Géométrie et complexité

Chapitre 1

Introduction

À première vue, on pourrait penser qu'il n'existe aucun rapport entre une approche en termes géométriques et une approche en terme de complexité de l'espace ou du plan.

Il y a pourtant là d'intéressantes études à mener, dans ce que l'on pourrait nommer approche *informationnelle* de l'espace. Physiciens, logiciens et philosophes travaillent depuis longtemps en ce sens, et on peut y raccrocher les considérations thermodynamiques de Bennet et Zürek.

Nous nous contentons ici de deux illustrations. La première est un travail récent de [HS] qui permet de retrouver, à partir de résultats sur l'entropie, un résultat géométrique.

Nous précisons ensuite la convergence de point de vue entre complexité et régression linéaire en analyse de données.

Ce dernier résultat ne prétend pas à l'originalité, il ne fait que reformuler des résultats connus, mais nous pensons qu'il donne un éclairage original.

Chapitre 2

Une approche non classique de la géométrie euclidienne

Dans de récents travaux [HRSV97], Hammer, Romashchenko, Shen et Vereshchagin ont étudié les inégalités linéaires vérifiées par l'entropie de Shannon et la complexité de Kolmogorov.

$X_1,\ldots,X_n$ sont soit n variables dont les valeurs sont des mots binaires de longueur finie (dans le cas de la complexité de Kolmogorov), soit n variables aléatoires (dans le cas de l'entropie de Shannon). $\mathcal{A}$ désigne l'ensemble des sous-ensembles non vides de $\{1,\ldots,n\}$. Soit $\alpha = \{\alpha_1,\ldots,\alpha_m\}$ un élément de $\mathcal{A}$. X_α désigne le m-uplet de variables $(X_{\alpha_1},\ldots,X_{\alpha_m})$.

Théorème 3.1 (Hammer, Romashchenko, Shen, Vereshchagin) *Toute inégalité linéaire vraie pour la complexité de Kolmogorov est également vraie pour l'entropie de Shannon et réciproquement. Plus précisément, un ensemble de coefficients $\{\lambda_\alpha | \alpha \in \mathcal{A}\}$ étant donnés, l'inégalité*

$$\sum_{\alpha\in\mathcal{A}} \lambda_\alpha H_P(X_\alpha) \geq 0$$

est vraie pour toute distribution de probabilité P sur n variables aléatoires $X_1,\ldots,X_n$ si et seulement si l'inégalité

$$\left(\sum_{\alpha\in\mathcal{A}} \lambda_\alpha K(X_\alpha)\right) + \mathcal{O}(l(l(X_1)+\ldots+l(X_n))) \geq 0$$

est vraie pour tous mots binaires de longueur finie $X_1,\ldots,X_n$.

Ces inégalités permettent de retrouver, de façon élégante, un résultat de géométrie à partir de considérations informationnelles sur les points de l'espace [HS]. Ce résultat est le suivant.

Théorème 3.2 *Soit V un volume de $\mathbb{R}^3$ de mesure v. Soit S_x, S_y et S_z les surfaces projetées de ce volume sur les trois plans perpendiculaires aux trois axes de coordonnées x, y et z. Soient s_x, s_y et s_z les mesures respectives de ces surfaces. On a alors l'inégalité :*

$$v \leq \sqrt{s_x s_y s_z}.$$

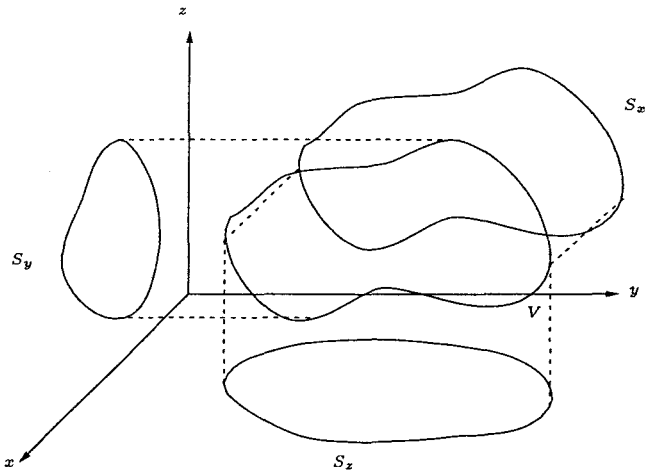


FIG. 2.1: *Volume V et ses projections S_x, S_y, S_z*

Nous montrons maintenant comment les inégalités linéaires vérifiées par l'entropie permettent de retrouver ce résultat. Soient trois variables aléatoires X, Y, Z données. Classiquement, on a les relations suivantes :

$$H(X,Y|Z) \leq H(X|Z) + H(Y|Z), \tag{2.1}$$
$$H(X,Y) = H(X) + H(Y|Z). \tag{2.2}$$

Le lemme suivant généralise l'inégalité 2.1 au cas de l'entropie d'un triplet de variable.

Lemme 3.1 *L'entropie d'un triplet de variables X, Y, Z est majorée par la demi-somme des entropies des variables prises deux à deux :*

$$H(X,Y,Z) \leq \frac{1}{2}\left[H(X,Y) + H(Y,Z) + H(X,Z)\right]. \tag{2.3}$$

Preuve.

Étape 1 : Nous commençons par établir l'inégalité préliminaire suivante :

$$H(X,Y,Z) + H(Z) \leq H(X,Z) + H(Y,Z).$$

D'après l'équation 2.2, on a $H(X,Y,Z) = H(Z) + H(X,Y|Z)$. En tenant compte de l'équation 2.1 il vient :

$$H(X,Y,Z) \leq H(Z) + H(X|Z) + H(Y|Z).$$

En ajoutant $H(Z)$ à chaque membre et en regroupant, nous obtenons :

$$H(X,Y,Z) + H(Z) \leq (H(Z) + H(X|Z)) + (H(Z) + H(Y|Z)).$$

Finalement, en tenant compte de l'inégalité 2.2 dans le membre de gauche, on obtient l'inégalité recherchée.

Étape 2 : En particularisant à chaque fois une des variables X, Y, Z, on obtient à partir de l'inégalité 2 trois inégalités :

$$\begin{aligned} H(X,Y,Z) + H(X) &\leq H(Y,X) + H(Z,X), && (2.4)\\ H(X,Y,Z) + H(Y) &\leq H(X,Y) + H(Z,Y), && (2.5)\\ H(X,Y,Z) + H(Z) &\leq H(X,Z) + H(Y,Z). && (2.6) \end{aligned}$$

En sommant membre à membre ces trois inégalités, nous obtenons :

$$3H(X,Y,Z) + H(X) + H(Y) + H(Z) \leq 2\left[H(X,Y) + H(Y,Z) + H(X,Z)\right].$$

Or $H(X,Y,Z) \leq H(X) + H(Y) + H(Z)$. En sommant ces deux inégalités et en simplifiant par deux il vient finalement :

$$2H(X,Y,Z) \leq H(X,Y) + H(Y,Z) + H(X,Z).$$

□

Preuve du théorème 3.2. Discrétisons l'espace en cubes unités. Le volume V contient v de ces cubes, les surfaces S_x, S_y et S_z en contiennent s_x, s_y et s_z carrés unités.

Un petit cube est désigné par un triplet de coordonnées (x, y, z). Supposons que l'on choisisse au hasard et de manière équiprobable les petits cubes du volume V. Nous avons donc un triplet de variables aléatoires (X, Y, Z). Le choix étant fait de manière équitable, son entropie $H(X, Y, Z)$ est maximale et vaut $\log(v)$. D'autre part, nous pouvons majorer l'entropie $H(X, Y)$ par sa valeur maximale. Celle-ci est atteinte lorsque les s_x carrés unités recouvrant S_x sont choisis de manière équiprobable. La même majoration peut-être obtenue pour les entropies $H(Y, Z)$ et $H(X, Z)$ et l'on obtient :

$$\begin{aligned} H(X,Y) &\leq \log(s_z),\\ H(X,Z) &\leq \log(s_y),\\ H(Y,Z) &\leq \log(s_x). \end{aligned}$$

L'équation 2 devient alors :

$$\begin{aligned} 2\log(v) &\leq \log(s_x) + \log(s_y) + \log(s_z)\\ v &\leq \sqrt{s_x s_y s_z}. \end{aligned}$$

□

Chapitre 3

Régression linéaire et principe MDL

Nous examinons dans ce chapitre les liens entre deux méthodes d'inférence, à savoir d'une part *la régression linéaire* ou *méthode des moindres carrés*, méthode fondamentale de l'analyse de données, et d'autre part le principe *Minimum Description Length*.

D'une manière générale, on considère d variables d'un système liées par une relation linéaire inconnue. Elles décrivent donc un hyperplan H_0. Pour connaître cet hyperplan, on effectue des mesures, une mesure étant constituée d'un d-uplet dont chaque valeur est associée à une variable. Dans l'espace, chaque mesure correspond à un point qui, en raison d'erreurs de mesure inévitables, s'écarte plus ou moins de l'hyperplan H_0.

Comment peut-on utiliser ces mesures pour inférer une approximation de H_0 aussi bonne que possible? La régression linéaire et le principe MDL sont deux solutions possibles à cette question. Sous certaines hypothèses, elles conduisent asymptotiquement aux mêmes résultats.

Tout au long de ce chapitre, nous donnons des preuves dans le cas particulier de la dimension 1, preuves qui s'étendent au cas des dimensions supérieures. Le cas de la dimension 1 est par exemple celui d'une grandeur physique possédant une valeur x_0 que l'on cherche à déterminer. Pour cela, on réalise une série de mesures $x_1, \ldots, x_n$, que nous désignerons par X dans la suite.

3.1 Régression linéaire

De façon générale, la régression linéaire propose de choisir l'hyperplan qui minimise la somme des carrés de ses distances aux points de mesure. Dans notre exemple, la valeur $\bar{x}_0$ à choisir est celle qui minimise l'expression

$$\sum_{x_i \in X} (x - x_i)^2,$$

ce qui revient à prendre la moyenne : $\bar{x}_0 = \frac{1}{n} \sum_{x_i \in X} x_i$.

D'un point de vue statistique, dans l'hypothèse où les explications que l'on s'autorise sont des hyperplans H auxquels se superposent des bruits de mesures gaussiens, de moyenne nulle et d'écart-type σ, cette méthode se justifie de différentes façons.

- D'une part par le principe du maximum de vraisemblance : l'hyperplan obtenu par régression linéaire est celui pour lequel la probabilité d'observation du nuage de points expérimental est maximale. Dans notre exemple, la probabilité de faire l'observation X en prenant comme hypothèse que la valeur réelle est x est :

$$\begin{aligned} Pr(X|x) &= \Pi_i \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x_i-x)^2}{2\sigma^2}} \\ &= \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2\sigma^2}\sum_i (x_i-x)^2}. \end{aligned}$$

  Cette probabilité est maximale quand $\sum_i (x_i - x)^2$ est minimale, *i.e.* quand $x = \bar{x}_0$.

- D'autre part, l'hyperplan trouvé par régression est l'estimateur sans biais le plus efficace : sa valeur moyenne, quand on réalise des observations de n points, est H_0 (estimateur sans biais) ; c'est également l'hyperplan qui, toujours en moyenne sur des observations de n points, s'en écarte le moins (estimateur le plus efficace).

Dans le cas plus général d'un bruit non gaussien, mais toujours en forme de cloche (la densité de probabilité associée à ce bruit est paire et décroît quand l'amplitude du bruit croît), quand le nombre de points de mesures tend vers l'infini, l'hyperplan obtenu tend vers H_0 avec une probabilité qui tend vers 1. Dans notre exemple, c'est simplement l'expression de la loi des grands nombres :

$$\forall\, \delta\ \forall\, \epsilon\ \exists\, \nu\ \forall\, n \geq \nu\ Pr(\frac{1}{n} \sum_{x_i \in X} |x_i - x_0| < \delta) > 1 - \epsilon.$$

3.2 Le principe MDL

La démarche scientifique classique consiste à effectuer des observations du monde réel et à proposer des théories qui rendent compte au mieux de ces observations. Bien sûr, plusieurs théories peuvent être en compétition, ce qui pose le problème du choix de la meilleure d'entre elles. Différents principes proposent un critère de choix. Nous pouvons par exemple appliquer un

principe d'économie, connu sous le nom de *Rasoir d'Occam*, et qui s'exprime comme suit.

> *Parmi toutes les théories cohérentes avec les observations, choisir la plus simple d'entre elles.*

C'est un principe d'économie dans le sens où l'on ne doit mettre dans la théorie que le strict nécessaire pour rendre compte des observations.

Ce principe trouve une justification mathématique grâce à la *Complexité de Kolmogorov*. Dans ce cadre, une *explication simple* d'une observation est *un programme court* calculant cette observation. Les programmes courts étant les plus probables, nous pouvons exprimer plus rigoureusement le *Rasoir d'Occam* :

> *La théorie la plus courte consistante avec les observations est la plus probable.*

Notons qu'en un certain sens, ce principe est absolu, car il désigne la « meilleure théorie » [1]. La contrepartie à ce caractère absolu est évidemment la non calculabilité.

Le principe MDL (Minimum Description Length), dû à Rissanen [Ris78], relativise à une famille F de théories récursivement énumérables l'approche par la complexité de Kolmogorov, en proposant le critère de choix suivant.

> *Parmi toutes les théories d'une famille F, choisir celle qui minimise la somme de :*
>
> - *la longueur, exprimée en bits, de la description de la théorie ;*
> - *la longueur, exprimée en bits, de la description des observations avec l'aide de la théorie.*

En pratique, on s'arrange pour que le critère de choix soit calculable ou, tout au moins, pour disposer de bonnes approximations calculables.

Ce principe repose sur l'idée qu'il faut réaliser un compromis entre, d'une part, la taille de la théorie et, d'autre part, la taille des observations quand celles-ci sont codées à l'aide de la théorie. La théorie doit synthétiser toutes les régularités communes aux observations. Ce principe trouve également de solides justifications, moyennant certaines hypothèses sur les observations réalisées, dans le cadre de la complexité de Kolmogorov [2].

Reprenons l'exemple du début de ce chapitre. Nous réalisons n mesures $x_1, \ldots, x_n$ d'une grandeur x_0. Le bruit de mesure est de moyenne nulle. Nous

1. Il peut y en avoir plusieurs également courtes. Toutefois, la théorie de Kolmogorov montre qu'il y en a très peu. Nous utilisons le singulier pour simplifier.

2. Pour plus de précision, on pourra se reporter au chapitre 5.5 de [LV97], intitulé *Hypothesis Identification by Minimum Description Length.*

supposons que la précision de mesure est finie, ce qui revient à travailler sur des entiers. La famille de théories envisagée est l'ensemble des entiers x. La description d'une mesure x_i à l'aide de la théorie x est l'écart algébrique $x - x_i$ de x_i par rapport à x. Les valeurs algébriques sont codées par des mots binaires, de la façon suivante : le premier bit est 0 ou 1 selon que la valeur est positive ou négative ; les bits suivants sont l'écriture préfixe de la valeur absolue. Nous notons $l(x)$ la longueur du codage binaire d'une valeur algébrique x. Nous avons alors :

$$\forall\, x,\ l(x) = 2E(\log_2(|x|+1)) + 2, \tag{3.1}$$

$$\forall\, x, y, z,\ l(x-y) \leq l(x-z) + l(z-y). \tag{3.2}$$

E désigne ici la fonction partie entière. La première relation se déduit directement de la façon de coder une valeur algébrique. Montrons la seconde relation. Pour toutes valeurs algébriques x, y et z, nous avons, d'après l'inégalité triangulaire,

$$|x-y| + 1 \leq (|x-z|+1) + (|z-y|+1).$$

La croissance et la convexité de la fonction $\log_2$ entraînent que

$$\log_2(|x-y|+1) \leq \log_2(|x-z|+1) + \log_2(|z-y|+1).$$

La croissance de la fonction E et le fait que pour tout x et y nous ayons $E(x+y) \leq E(x) + E(y) + 1$ entraînent que

$$E(\log_2(|x-y|+1)) \leq E(\log_2(|x-z|+1)) + E(\log_2(|z-y|+1)) + 1,$$

ce qui montre bien la relation attendue.

Le principe MDL nous propose de choisir la valeur x_{MDL} qui minimise l'expression :

$$l(x) + l(x-x_1) + \ldots + l(x-x_n).$$

Pour un bruit en forme de cloche, quand le nombre de mesures tend vers l'infini, la valeur x_{MDL} ainsi trouvée tend vers x_0 avec une probabilité qui tend vers 1, *i.e.*

$$\forall\, \delta > 0\ \forall\, \epsilon > 0\ \exists\, \nu\ \forall\, n \geq \nu\ Pr(|x_{\text{MDL}} - x_0| < \delta) > 1 - \epsilon.$$

Nous le montrons dans le cas de notre exemple. Supposons que ce ne soit pas le cas, nous avons alors :

$$\exists\, \delta > 0\ \exists\, \epsilon > 0\ \forall\, \nu\ \exists\, n > \nu\ Pr(|x_{\text{MDL}} - x_0| > \delta) > \epsilon.$$

Soit $\delta' = l(\delta)$. Comme la fonction l est croissante sur $\mathbb{N}$, nous avons

$$\exists\, \delta' > 0\ \exists\, \epsilon > 0\ \forall\, \nu\ \exists\, n > \nu\ Pr(l(x_{\text{MDL}} - x_0) \geq \delta') > \epsilon.$$

Notons $X(n)$ l'ensemble des ensembles de n points de mesures et $X(n,\delta)$ l'ensemble $\{X \in X(n) \text{ tq } |x_{\text{MDL}} - x_0| > \delta\}$. Pour tout $X = \{x_1, \ldots, x_n\}$ de $X(n,\delta)$ nous avons $l(x_{\text{MDL}} - x_0) \geq \delta'$ et d'après l'inégalité 3.2 :

$$\forall\, i\ l(x_{\text{MDL}} - x_i) + l(x_i - x_0) \geq \delta'.$$

En sommant sur l'indice i, il vient :

$$\sum_i l(x_{\text{MDL}} - x_i) + \sum_i l(x_i - x_0) \geq n\delta'. \tag{3.3}$$

Or, par définition de x_{MDL} nous avons

$$\begin{aligned} \sum_i l(x_{\text{MDL}} - x_i) &\leq \sum_i l(x_0 - x_i) + l(x_0) - l(x_{\text{MDL}}) \\ &\leq \sum_i l(x_0 - x_i) + l(x_0). \end{aligned} \tag{3.4}$$

En remplaçant, dans l'équation 3.4, $\sum l(x_{\text{MDL}} - x_i)$ par sa minoration donnée dans l'équation 3.3, nous obtenons

$$\begin{aligned} \frac{1}{n}\sum_i l(x_0 - x_i) &\geq \frac{\delta'}{2} - \frac{l(x_0)}{2n} \\ &> \frac{\delta'}{4} \text{ pour } n \text{ assez grand.} \end{aligned}$$

et *a fortiori*

$$\frac{1}{n}\sum_i |x_0 - x_i| > \frac{\delta'}{4} \text{ pour } n \text{ assez grand.}$$

Ceci a lieu pour tout X de $X(n,\delta)$, *i.e.* avec une probabilité supérieure à ϵ, ce qui est en contradiction avec la loi des grands nombres.

3.3 Conclusion

Dans le cas d'un bruit en forme de cloche, la régression linéaire et le principe de Rissanen sont, asymptotiquement, équivalents. Ce n'est pas nécessairement le cas si le bruit a une forme différente. Par exemple si la densité de probabilité qui lui est associée est paire mais avec deux maxima pour des amplitudes a et $-a$, le principe de Rissanen donnera asymptotiquement deux solutions, à savoir les hyperplans situés parallèlement de part et d'autre de H_0 à une distance a, tandis que la régression linéaire donnera, asymptotiquement, l'hyperplan H_0.

Quatrième partie

Observation d'un flot de données : complexité d'une fonction et complexité de son graphe

Chapitre 1

Introduction

L'objectif de cette partie est d'expliciter les relations entre la *complexité globale* d'un système qui produit un flot de données, et la complexité observée dans le flot de données produit. Par *complexité globale*, nous entendons *quantité minimale d'information* décrivant *exactement* et *complètement* le système. Nous supposons que cette complexité est inaccessible directement à l'observateur pour qui le système n'est qu'une boîte noire dont le fonctionnement interne est inconnu. L'observateur n'a connaissance du système qu'à travers l'observation de son comportement sur des fenêtres ayant des tailles finies. C'est à partir de ces observations que l'observateur peut envisager de construire un modèle du système, modèle qu'il espère être une approximation aussi bonne que possible de la réalité.

La fenêtre d'observation peut par exemple être une fenêtre temporelle. Dans ce cas, il faut envisager des modèles de processus qui effectuent le calcul d'une nouvelle donnée en un temps borné. Ces modèles, des machines déterministes à états finis sans entrées, ont un comportement pauvre : leurs sorties sont ultimement périodiques (tic-tac d'une horloge par exemple). Nous précisons et adaptons ce point de vue temporel dans la partie suivante, dans laquelle nous considérons des processus avec paramètres. Dans le cas présent, nous envisageons un système produisant par exemple les décimales de π. Notre système est alors représentable par une fonction de $\mathbb{N}$ dans $\mathbb{N}$, ou de $\mathbb{N}$ dans $\{0,1\}$.

Pour mener cette étude, nous nous plaçons dans le cadre de la théorie de la complexité de Kolmogorov, théorie qui donne une consistance mathématique rigoureuse à la notion intuitive et vague d'information. Nous nous limitons à des systèmes *déterministes* et *non bruités* (les grandeurs observées sont les grandeurs réellement produite par le système), c'est-à-dire que nous voyons un tel système comme une fonction f de $\mathbb{N}$ dans $\mathbb{N}$. Nous formalisons la notion de complexité globale d'un système en la définissant comme la taille d'un plus petit programme calculant la fonction représentant ce système. L'observation du système sur une fenêtre conduit à la connaissance

partielle du graphe de cette fonction, restreint à cette fenêtre. L'observateur infère un modèle local, qui est un programme calculant une fonction coïncidant avec la fonction f sur la fenêtre d'observation. La complexité observée est bornée par la taille maximale d'un modèle local, que nous définissons comme la *complexité faible.*

Il semble raisonnable de penser que la complexité globale et la complexité faible d'une fonction sont les mêmes: en faisant des observations sur des fenêtres de plus en plus grandes, on peut penser capturer toute l'information caractérisant la fonction, ainsi que paraît le montrer l'exemple suivant.

Considérons une fonction f polynômiale de degré n. Nous faisons des observations de f sur des fenêtres $0, \ldots, k$ croissantes et à chaque fois nous inférons un polynôme, de plus faible degré possible, rendant parfaitement compte des observations. Pour $k < n$, nous inférerons un polynôme de degré k, et pour $t \geq n$, nous inférerons toujours le même polynôme de degré n, à savoir la fonction f elle-même. Pour $k \geq n$, nous avons en quelque sorte capturé toute l'information sur f, par une simple observation de son comportement sur une fenêtre de taille finie, égale au degré du polynôme.

Le principal résultat de cette partie, à savoir le théorème 4.4 du chapitre 3, établit que, de façon surprenante, cette intuition est fausse. Parmi une famille de fonctions donnée (famille devant respecter certaines hypothèses raisonnables) il est toujours possible de trouver une fonction pour laquelle la différence entre la complexité globale et la complexité faible peut être aussi grande que l'on veut. Si nous en revenons aux processus, cela signifie que, pour des familles de processus raisonnables, il est possible de trouver un processus pour lequel la quantité d'information sortie est toujours beaucoup plus petite que l'information globale du processus.

L'exemple précédent de la fonction polynômiale est faussé par le fait que nous n'inférons que des polynômes. Dans le cadre de la complexité de Kolmogorov, les modèles que nous inférons peuvent être quelconques. Il existe des polynômes pour lesquels la donnée de la fenêtre, c'est-à-dire de k, donne, pour tout k suffisamment grand, beaucoup d'information. Le plus petit modèle (parmi tous les modèles possibles et non seulement parmi les modèles de polynômes) n'est plus alors un programme calculant un polynôme, mais un programme beaucoup plus court calculant une fonction qui n'est pas un polynôme et impossible à expliciter en général. L'exemple 4.1 du théorème 4.4 est une illustration plus directe du même fait, dont l'intuition reste toutefois difficile à saisir.

Ce résultat peut sembler rendre vain l'inférence d'un bon modèle d'un système à partir de l'observation de son comportement. Cependant, un second résultat, à savoir le théorème 4.5, vient tempérer le premier. Pour les mêmes familles de fonctions, la majorité des fonctions d'une famille ont des complexités globale et faible qui sont proches. Les cas où ces complexités sont très différentes sont rares.

Le chapitre 2 introduit les définitions formelles de modèle global et de

modèle faible, ainsi que celles de complexité globale et de complexité faible associées, puis présente une étude dont l'objectif est de justifier la définition de la complexité faible.

Dans le chapitre 3 nous rappelons tout d'abord un résultat attribué à Meyer par Loveland [Lov69], et dont la preuve apparaît également dans l'article fondamentale de Zvonkin et Levin [ZL70]. Ce résultat assure l'existence d'un modèle global lorsque la complexité faible est finie. Nous définissons ensuite la notion de *famille raisonnable de fonctions* pour établir nos principaux résultats, à savoir les théorèmes 4.4 et 4.5. Le premier affirme que la complexité faible ne peut toujours constituer une bonne approximation de la complexité globale, le second nuance cette affirmation en montrant que l'approximation est tout de même valable dans la plupart des cas.

Chapitre 2

Étude préliminaire

Nous donnons dans ce chapitre les définitions formelles pour les notions de *modèle faible* et *modèle global* et les complexités associées. Plusieurs définitions étant possibles *a priori* pour la complexité faible, nous menons une étude afin de justifier le choix fait pour la définition de cette notion.

2.1 Définitions

Nous considérons des fonctions de $\mathbb{N}$ dans $\mathbb{N}$ et leurs graphes $\mathcal{G}_f = \{(x,y), y = f(x)\}$. Nous notons $\mathcal{G}_f^n$ la restriction du graphe de f aux abscisses inférieures ou égales à n :

$$\mathcal{G}_f^n = \{(x,y), x \leq n, y = f(x)\}.$$

Nous codons un ensemble $\mathcal{G}_f^n$ par le mot $\langle f(0), f(1), \ldots, f(n)\rangle$ qui est la concaténation des écritures auto-délimitées de $f(0), f(1), \ldots, f(n)$.

Une fonction récursive est une fonction définie partout, calculable par programme[1]. Pour formaliser la notion d'information caractérisant une fonction récursive, nous définissons les concepts de *modèle global* et *complexité globale.*

Définition 4.1 *Un* modèle global *d'une fonction récursive f est un programme P acceptant n en entrée et qui, pour tout n, s'arrête et produit $\mathcal{G}_f^n$,* i.e.

$$\forall\, n \in \mathbb{N}\ P(n) = \mathcal{G}_f^n.$$

La complexité globale $K_g(f)$ *d'une fonction f est la longueur d'un plus petit modèle global de f, s'il existe, l'infini sinon.*

1. L'ensemble des fonctions récursives n'est pas récursivement énumérables. Mais nous ne considérons ici que des fonctions prises une à une.

Une définition différente peut être envisagée, par exemple en requérant que, pour tout n, le programme $P(n)$ calcule $f(n)$ et non $\mathcal{G}_f^n$. À une constante additive près, cela donne la même notion de complexité globale. Les résultats que nous présentons peuvent s'exprimer avec l'une ou l'autre des définitions. Notons enfin qu'une fonction f est récursive si et seulement si sa complexité globale est finie.

Nous prenons connaissance d'une fonction f par l'observation des restrictions de son graphe. Pour n fixé, cela nous conduit à l'idée de *modèle local* de f sur l'intervalle $0, \ldots, n$: nous le définissons comme un programme calculant $\mathcal{G}_f^n$ à partir de n. Si tous les plus petits modèles locaux de f, pour n décrivant $\mathbb{N}$, ont des tailles bornées, alors ils sont en nombre fini et l'un d'entre eux au moins calcule des restrictions $\mathcal{G}_f^n$ du graphe de f pour une infinité de n. Cela nous amène maintenant à introduire la notion de *modèle faible.*

Définition 4.2 *Un* modèle faible *d'une fonction f est un programme P acceptant en entrée n et qui, pour une infinité de n, s'arrête et produit $\mathcal{G}_f^n$. La* complexité faible $K_w(f)$ *d'une fonction f est :*

$$K_w(f) = \limsup_{n \to \infty} K(\mathcal{G}_f^n | n).$$

Chaque valeur d'adhérence finie de la suite $K(\mathcal{G}_f^n|n)$ est la taille d'un plus petit programme calculant $\mathcal{G}_f^n$ pour un ensemble infini de n où cette valeur d'adhérence est atteinte. Si la complexité faible de f est finie, alors c'est la taille maximale d'un plus petit modèle faible de f.

2.2 Étude des modèles faibles

Proposition 4.1 *Si une fonction f est récursive, alors elle possède un modèle global et l'on a $K_w(f) \leq K_g(f)$.*

En général, les résultats de la complexité de Kolmogorov sont donnés à un terme résiduel additif près. Ce n'est pas le cas ici, tout modèle global étant un modèle faible pour tous les n. Toutefois, si l'on définit la complexité globale d'une fonction f comme la taille d'un plus petit programme qui pour tout n produit $f(n)$, alors il devient nécessaire d'ajouter une constante additive dans l'inégalité : $K_w(f) \leq K_g(f) + C$.

Nous présentons maintenant quelques propriétés de la suite $K(\mathcal{G}_f^n|n)$ qui nous permettent de justifier la définition de la complexité faible. En effet :

1. la suite $K(\mathcal{G}_f^n|n)$ converge pour au plus un nombre fini de fonctions f (voir Théorème 4.1) ;

2. la limite inférieure de $K(\mathcal{G}_f^n|n)$ est majorée par une constante indépendante de f (voir Lemme 4.1), ceci étant essentiellement dû au fait que l'information caractérisant f peut être encodée infiniment souvent dans l'entrée n ;

3. la limite supérieure de $K(\mathcal{G}_f^n|n)$ est la taille d'un plus petit modèle faible pour lequel l'entrée n fournit un minimum d'information sur f.

Lemme 4.1 *Il existe un modèle faible universel pour les fonctions récursives* i.e. *il existe un programme $P_u(n)$ qui est un modèle faible de toute fonction récursive.*

Preuve. Considérons le programme $P_u(n)$ suivant :

Programme $P_u(n)$
    Si `il existe un entier` k `et un mot` x `tels que` $n = 1^k0x$
    Alors
        `Simuler le programme` x `sur l'entrée` n
    Sinon
        `Boucler indéfiniment`
    Fin Si
Fin

Soit f une fonction récursive possédant un modèle global P_g. P_u est un modèle faible de f car pour tout n appartenant à l'ensemble $\{1^k0P_g, k \in \mathbb{N}\}$, $P_u(n)$ calcule $\mathcal{G}_f^n$. □

Lemme 4.2 *Étant donné $A > 0$, l'ensemble des fonctions f dont la complexité faible est majorée par A est fini.*

Preuve. Soit $A > 0$ fixé. Supposons qu'il existe une infinité de fonctions f telles que $K_w(f) \leq A$. Soient $f_1, \ldots, f_{2^{2A}}$ 2^{2A} telles fonctions distinctes. Nous avons :

1. par définition de la complexité faible, $\forall\ k = 1, \ldots, 2^{2A}\ \exists\ n_k\ \forall\ n \geq n_k,\ K(\mathcal{G}_{f_k}^n|n) \leq A$;

2. les fonctions considérées étant distinctes, $\exists\ n_0\ \forall\ n \geq n_0\ \forall\ i, j = 1, \ldots, 2^{2A},\ i \neq j \Rightarrow \mathcal{G}_{f_i}^n \neq \mathcal{G}_{f_j}^n$.

Soit $n = max\{n_0, n_1, \ldots, n_{2^{2A}}\}$. Alors nous avons 2^{2A} restrictions de graphes distinctes, ayant une complexité inférieure ou égale à A, ce qui est évidemment impossible, le nombre de programmes disponibles étant borné par 2^A. □

Les lemmes 4.1 et 4.2 impliquent clairement le théorème suivant.

Théorème 4.1 *Les fonctions récursives pour lesquelles* $\lim_{n\to\infty} K(\mathcal{G}_f^n|n)$ *existe sont en nombre fini.*

Dans ce théorème, le nombre fini de fonctions pour lesquelles la limite existe dépend du système de programmation.

Voyons d'abord un système pour lequel ce nombre est nul. Considérons l'énumération standard des fonctions partielles récursives ϕ_i. Définissons maintenant le système de programmation suivant : ψ_0 est la fonction récursive calculée par le modèle faible universel P_u proposé dans la preuve du lemme 4.1. $\psi_1, \ldots, \psi_{1998}$ sont des fonctions dont les index sont des programmes qui divergent toujours, et $\psi_i = \phi_i$ pour tout $i > 1998$. Soit f une fonction récursive. Nous pouvons faire les remarques suivantes.

1. $\liminf K(\mathcal{G}_f^n|n) = 0$ car $\psi_0(n) = \mathcal{G}_f^n$ pour une infinité de n.
2. $\limsup K(\mathcal{G}_f^n|n) \geq |1998|$ pour les raisons suivantes.
   (a) D'une part, nous ne pouvons avoir $\limsup K(\mathcal{G}_f^n|n) = 0$. Si c'était le cas, nous aurions $\psi_0(n) = \mathcal{G}_f^n$ pour n suffisamment grand. Or ψ_0 diverge pour une infinité de n.
   (b) D'autre part, nous ne pouvons avoir $0 < \limsup K(\mathcal{G}_f^n|n) < |1998|$. Si c'était le cas, il existerait un programme P tel que $1 \leq |P| \leq 1997$ calculant $\mathcal{G}_f^n$ pour une infinité de n. Or un tel programme diverge toujours dans le système de programmation choisi.

Par conséquent, la limite de $K(\mathcal{G}_f^n|n)$ n'existe pour aucune fonction récursive dans ce système de programmation.

Le nombre de fonctions récursives pour lesquelles $\lim K(\mathcal{G}_f^n|n)$ existe peut être non nul, par exemple dans le système de programmation suivant. $\gamma_0, \ldots, \gamma_{1998}$ sont des fonctions dont les index sont des modèles globaux de fonctions récursives $f_1, \ldots, f_{1998}$ distinctes :

$$\forall\, i \leq 1998,\ \forall\, n,\ \gamma_i(n) = \mathcal{G}_{f_i}^n.$$

Considérons une fonction f_i, $i \leq 1998$. Comme le programme calculant γ_i est un modèle global de f_i, nous avons :

$$\limsup K(\mathcal{G}_{f_i}^n|n) \leq |i|. \tag{2.1}$$

Soit j tel que $|j| < |i|$. Comme les fonctions f_i sont deux à deux distinctes, pour n suffisamment grand $\mathcal{G}_{f_j}^n \neq \mathcal{G}_{f_i}^n$ *i.e.* $\gamma_j(n) \neq \gamma_i(n)$.

Par conséquent, pour n suffisamment grand et pour tout j tel que $|j| < |i|$, nous avons $\gamma_j(n) \neq \gamma_i(n)$. Ceci implique que :

$$\liminf K(\mathcal{G}_{f_i}^n|n) \geq |i|. \tag{2.2}$$

Finalement, pour tout $i \leq 1998$, les équations 2.1 et 2.2 impliquent que $\lim K(\mathcal{G}_{f_i}^n|n)$ existe et vaut $|i|$.

Chapitre 3

Comparaison des modèles faibles et globaux

Nous allons maintenant nous intéresser de plus près à l'inégalité entre la complexité faible et la complexité globale présentée dans la proposition 4.1, au chapitre précédent.

3.1 Existence d'un modèle global

Un théorème (ici Théorème 4.2), que Loveland attribue à Meyer [Lov69], établit que si $K(\mathcal{G}_f^n|n)$ est bornée sur $\mathbb{N}$, *i.e.* si la complexité faible de f est finie, alors f est une fonction récursive. En d'autres termes, l'existence d'un nombre fini de modèles faibles calculant toutes les restrictions de graphe $\mathcal{G}_f^n$ assure la récursivité de la fonction f.

Ce résultat est un bon exemple d'une situation fréquemment rencontrée dans la théorie de la complexité de Kolmogorov, à savoir qu'un résultat paraissant intuitif et évident s'avère en réalité difficile à établir. C'est pourquoi nous en donnons ici la preuve. La difficulté de ce résultat provient de ce que nous ne savons pas *a priori* quel modèle faible calcule $\mathcal{G}_f^n$ pour un n donné.

Théorème 4.2 (Meyer) *Soit f une fonction définit partout. Si $K(\mathcal{G}_f^n|n)$ est borné sur $\mathbb{N}$ alors f est récursive.*

Preuve. Supposons que pour tout n nous ayons $K(\mathcal{G}_f^n|n) \leq A$. Il existe donc un ensemble fini de programmes $\mathcal{P} = \{P_1, \ldots, P_N\}$ tels que, pour tout n, l'un au moins de ces programmes calcule $\mathcal{G}_f^n$.

Soit b_n le nombre de programmes de $\mathcal{P}$ convergeant sur l'entrée n. Cette suite étant bornée, $b = \limsup b_n$ est fini. Il existe donc n_0 tel que, pour tout $n \geq n_0$, nous ayons $b_n \leq b$.

Soit D le sous-ensemble de $\mathbb{N}$ tel que pour tout entier de D, le nombre de programmes de $\mathcal{P}$ qui convergent avec cet entier en entrée est exactement b.

Cet ensemble est infini et est récursivement énumérable de la façon suivante. Pour tout entier $n \geq n_0$, nous simulons en parallèle tous les programmes de $\mathcal{P}$ en leur donnant en entrée cet entier. Quand b programmes exactement s'arrêtent sur un entier, alors cet entier appartient à D. D étant infini et récursivement énumérable, il en existe un sous-ensemble D' infini et récursivement énumérable dans l'ordre croissant. Notons d_n le n-ième élément de D' ainsi énuméré.

Nous définissons maintenant, de façon récurrente, un arbre dont les noeuds sont des couples $(\Pi, \mathcal{G})$, Π étant un sous-ensemble de programmes de $\mathcal{P}$, et $\mathcal{G}$ un sous-ensemble de $\mathbb{N}^2$ tel que :

$$\exists\, l\ \forall\, \lambda \leq l\ \exists \text{ un unique } \mu,\ (\lambda, \mu) \in \mathcal{G}.$$

La racine de cet arbre est le couple $(\emptyset, \emptyset)$. Le couple $(\Pi', \mathcal{G}')$ est un noeud de profondeur k ayant pour père le noeud $(\Pi, \mathcal{G})$ de profondeur $k-1$ si et seulement si :

1. $\mathcal{G}'$ est un sous-ensemble de $\mathbb{N}^2$ tel que :

$$\forall\, \lambda \leq d_k\ \exists \text{ un unique } \mu,\ (\lambda, \mu) \in \mathcal{G};$$

2. Π' est l'ensemble des programmes de $\mathcal{P}$ convergeant sur l'entrée d_k et calculant $\mathcal{G}'$;

3. $\mathcal{G} \subset \mathcal{G}'$.

Considérons par exemple le cas d'une fonction pour laquelle nous ayons $\mathcal{P} = \{P_1, P_2, P_3, P_4\}$. La figure 3.1 montre ce que peut être le début d'un tel arbre pour cette fonction. À la profondeur 1 nous voyons que P_1 et P_2 calculent $\mathcal{G}_1$ sur l'entrée d_1 et que P_3 calcule $\mathcal{G}_2$ sur l'entrée d_1. À la profondeur 2, nous voyons que P_4 calcule $\mathcal{G}_4$ sur l'entrée d_2, avec $\mathcal{G}_1 \subset \mathcal{G}_4$.

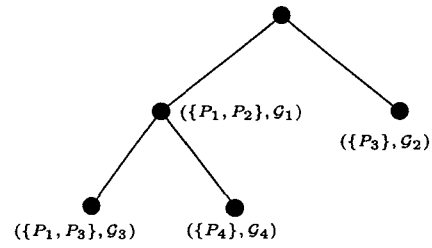


FIG. 3.1: *Exemple d'arbre*

Un chemin de longueur infinie commençant à la racine est une suite de couples $(\Pi_k, \mathcal{G}_k)$ tels que pour tout k :

- $\mathcal{G}_k \subset \mathcal{G}_{k+1}$;

– il existe un programme de $\mathcal{P}$ calculant $\mathcal{G}_k$ sur l'entrée d_k.

Un chemin de longueur infinie commençant à la racine représente donc une fonction g telle que pour tout k il existe un programme de $\mathcal{P}$ calculant $\mathcal{G}_k$ sur l'entrée d_k. Réciproquement, toute fonction satisfaisant cette propriété est représentée par un unique chemin infini commençant à la racine. Par conséquent, l'arbre contient au moins un chemin infini (celui qui représente f).

Le nombre de chemins infinis commençant à la racine est fini. Supposons que cela ne soit pas le cas. Considérons alors $N+1$ de ces chemins. Il existe une profondeur à laquelle ils sont tous séparés, donc à laquelle se trouvent $N+1$ noeuds. Or à chacun de ces noeuds est associé, en particulier, un sous-ensemble non vide de $\mathcal{P}$ disjoints de ceux des autres noeuds se trouvant à la même profondeur. Il ne peut donc y avoir au plus que N noeuds distincts, d'où une contradiction.

Les chemins infinis depuis la racine étant en nombre fini, il existe une profondeur ν à partir de laquelle tous ces chemins sont séparés. Parmi tous les noeuds de profondeur ν se trouve un noeud à partir duquel commence un unique chemin infini et dont chacun des noeuds est associé aux graphes $\mathcal{G}_f^{d_\nu}$, $\mathcal{G}_f^{d_{\nu+1}}$, etc. Appelons α le sous-arbre commençant à partir de ce noeud et α_k cet arbre tronqué à la profondeur k. La figure 3.2 montre ce que peut être le début de α.

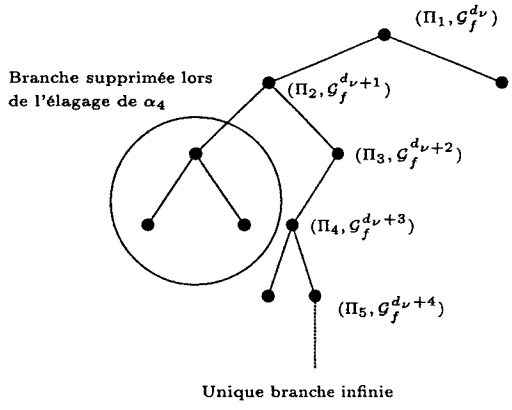


FIG. 3.2: *Un exemple d'arbre α*

La fonction CONSTRUIREARBRE(k) construit α_k pour tout k.

Fonction CONSTRUIREARBRE(k)
  Pour i de 1 à k **Faire**
    Calculer $d_{\nu+i}$ par énumération de D'
    Simuler en parallèle tous les programmes de $\mathcal{P}$ sur $d_{\nu+i}$
    Attendre que b d'entre eux exactement s'arrêtent
    Construire les noeuds de profondeur i de α
  Fin Pour
  Retourner l'arbre construit
Fin

Appelons ÉLAGUER(α_k) une procédure qui coupe toutes les branches de α_k n'atteignant pas la profondeur k.

α contient un seul chemin infini depuis sa racine, chemin qui représente la fonction f. Pour calculer un graphe $\mathcal{G}_f^n$, il suffit de calculer un graphe $\mathcal{G}_f^{d_k}$ avec $d_k \geq n$, *i.e.* de trouver un noeud $(\Pi, \mathcal{G}_f^{d_k})$ avec $d_k \geq n$, noeud situé à la profondeur $\nu - k$ sur le seul chemin infini de l'arbre α.

Cependant, il est possible qu'à cette profondeur se trouvent d'autres noeuds. Mais alors ils sont nécessairement sur un chemin de longueur finie. Pour k suffisamment grand, l'élagage de α_k élimine tous ces noeuds, et il ne reste alors, à la profondeur $\nu - k$, que le noeud $(\Pi, \mathcal{G}_f^{d_n})$.

En conséquence, le programme $P_f(n)$ suivant est un programme qui, pour tout n, calcule $\mathcal{G}_f^n$.

Programme $P_f(n)$
  Si $n \leq \nu$ **Alors**
    Sortir $\mathcal{G}_f^\nu$ tronqué aux abcisses inférieures ou égales à n
  Sinon
    $i = 0$
    Répéter
      $\beta =$ CONSTRUIREARBRE(i)
      ÉLAGUER(β)
      $i = i + 1$
    Jusqu'à obtenir un seul noeud à la profondeur $n - \nu$
    Sortir le graphe associé au seul noeud de profondeur $n - \nu$
  Fin Si
Fin

Remarquons que ce programme fait intervenir des constantes ν et b qui ne sont pas calculables. Nous avons donc prouvé l'existence d'un modèle global que l'on ne sait pas construire effectivement. □

3.2 Comparaison des complexités faible et globale

Nous venons de voir qu'une complexité faible finie implique l'existence d'un modèle global. Sommes-nous alors en droit d'approximer la complexité globale par la complexité faible? La preuve du théorème 4.2, non constructive, ne fournit aucune réponse à cette question. En réalité, et de façon surprenante, la réponse est non, comme le montre le théorème 4.4. Ceci implique qu'il n'existe pas de preuve constructive du théorème de Meyer.

La complexité de Kolmogorov étant définie à une constante additive près, il nous faut envisager une famille de fonctions pour exprimer ce résultat.

Nous énonçons d'abord un premier théorème, dont nous devons la démonstration au Professeur Shen, et qui établit l'existence d'une famille de fonctions récursives pour laquelle la complexité faible n'est pas une approximation de la complexité globale.

Théorème 4.3 *Il existe une famille de fonctions récursives* $\mathcal{F} = \{f_i\}_{i\in\mathbb{N}}$ *telle que*

$$\forall\ A\ \exists\ i \in \mathbb{N}\ K_g(f_i) - K_w(f_i) > A.$$

La preuve de ce théorème est basée sur la notion de machine de Turing avec oracle et sur la complexité K' qui lui est associée. L'oracle est l'ensemble $\mathbb{K}$ des programmes s'arrêtant sur notre machine de référence, ensemble qui est récursivement énumérable.

Preuve. Soit $\mathcal{F} = \{f_i\}_{i\in\mathbb{N}}$ la famille de fonctions suivante :

1. $\forall\ n \leq i, f_i(n) = 1$;
2. $\forall\ n > i, f_i(n) = 0$.

L'idée de cette preuve est de majorer $K_w(f_i)$ et de minorer $K_g(f_i)$ par deux fonctions de i pouvant s'écarter l'une de l'autre autant que l'on veut. Plus précisément, nous allons démontrer les trois inégalités suivantes :

1. $K_w(f) \leq K'(i) + \mathcal{O}(1)$
2. $\forall\ A\ \exists\ i\ K(i) - K'(i) > A$
3. $K(i) \leq K_g(f_i) + \mathcal{O}(1)$

Première inégalité : i étant fixé, désignons par P_i le plus petit programme utilisant l'oracle $\mathbb{K}$ et calculant i. $\mathbb{K}$ étant récursivement énumérable, fixons un procédé d'énumération de cet ensemble, et notons $\mathbb{K}_n$ l'ensemble des n premiers énumérés de $\mathbb{K}$ par ce procédé.

Envisageons maintenant le programme Q_i suivant, acceptant en entrée n et calculant $\mathcal{G}^n_{f_i}$ pour tout n suffisamment grand.

Programme $Q_i(n)$
Calculer $\mathbb{K}_n$.
Simuler P_i en utilisant $\mathbb{K}_n$ à la place de l'oracle $\mathbb{K}$.
Soit k le résultat de cette simulation.
Produire en sortie l'ensemble $\{(x,1) \mid x \leq k \text{ et } x \leq n\} \cup \{(x,0) \mid x > k \text{ et } x \leq n\}$.
Fin

Quelque soit i, le programme P_i effectue son calcul en posant un nombre fini de questions à l'oracle. Donc il existe n_i tel que, pour tout $n \geq n_i$, $\mathbb{K}_n$ contienne toutes les réponses aux questions posées par ce programme. Par conséquent,

$$\forall\, i\ \exists\, n_i\ \forall\, n \geq n_i\ Q_i(n) \text{ calcule } \mathcal{G}^n_{f_i},$$

ce qui nous permet de majorer $K(\mathcal{G}^n_{f_i}|n)$:

$$\begin{aligned} \forall\, i\ \exists\, n_i\ \forall\, n \geq n_i\ K(\mathcal{G}^n_{f_i}|n) &\leq |Q_i| + \mathcal{O}(1) \\ &\leq |P_i| + \mathcal{O}(1) \text{ par construction de } Q_i \\ &= K'(i) + \mathcal{O}(1) \text{ par définition de } P_i. \end{aligned}$$

Finalement, cela prouve bien l'inégalité attendue.

Seconde inégalité : Nous allons maintenant montrer que

$$\forall\, A\ \exists\, i\ K(i) - K'(i) > A.$$

Pour cela, envisageons la suite u_n définie par $u_n = \min\{x | K(x) > n\}$. En énumérant dans l'ordre croissant tous les x, et pour chacun d'eux en calculant $K(x)$ grâce à l'oracle $\mathbb{K}$, nous pouvons observer le premier d'entre eux vérifiant $K(x) > n$, et ainsi obtenir u_n. Par conséquent nous avons $K'(u_n|n) \leq \mathcal{O}_n(1)$, ce qui implique que $K'(u_n) \leq \log(n) + \mathcal{O}(1)$. Comme par définition de u_n nous avons aussi $K(u_n) > n$, nous obtenons l'inégalité attendue.

Troisième inégalité : Il s'agit de montrer que

$$K(i) \leq K_g(f_i) + \mathcal{O}(1).$$

À l'aide d'un modèle global de f_i, nous pouvons retrouver i en calculant les premières valeurs $f(0), f(1), \ldots$ de f_i jusqu'à obtenir une valeur nulle.

□

Nous élargissons le résultat précédent en montrant qu'il est valable pour toutes les familles de fonctions *raisonnables*, familles que nous définissons maintenant.

Définition 4.3 (famille raisonnable de fonctions) *Une* famille raisonnable de fonctions $\mathcal{F} = \{f_i\}_{i\in\mathbb{N}}$ *est un ensemble récursif de fonctions récursives deux à deux distinctes,* i.e. *vérifiant :*

1. $\forall\, i,j\ (i \neq j) \Rightarrow (f_i \neq f_j)$;

2. $\exists$ *un programme* $\mathcal{P}$ $\forall$ i, x $\mathcal{P}(i, x)$ *s'arrête et produit* $f_i(x)$.

Théorème 4.4 *Soit* $\mathcal{F} = \{f_i\}_{i \in \mathbb{N}}$ *une famille raisonnable de fonctions. Alors nous avons :*

$$\forall\ A\ \exists\ i \in \mathbb{N}\ K_g(f_i) - K_w(f_i) > A.$$

En fait, nous allons montrer un résultat plus fort : K_w est infiniment souvent de l'ordre de $\log(k)$ quand K_g est de l'ordre de k.

Exemple 4.1 Considérons la famille $\mathcal{F}$ de fonctions définie de la façon suivante :

1. $\forall\ n\ f_0(n) = 0$;
2. $\forall\ i > 0\ f_i(i) = 1$ et $\forall\ n\ \neq i\ f_i(n) = 0$.

Il s'agit d'une famille raisonnable. Par conséquent, pour toute constante donnée A, il est possible de trouver une fonction de cette famille pour laquelle la différence entre la complexité locale et la complexité faible est supérieure à A.

Le lemme 4.1 et le théorème 4.4 nous permettent d'illustrer (voir Figure 3.3) le comportement général de la suite $K(\mathcal{G}_f^n|n)$.

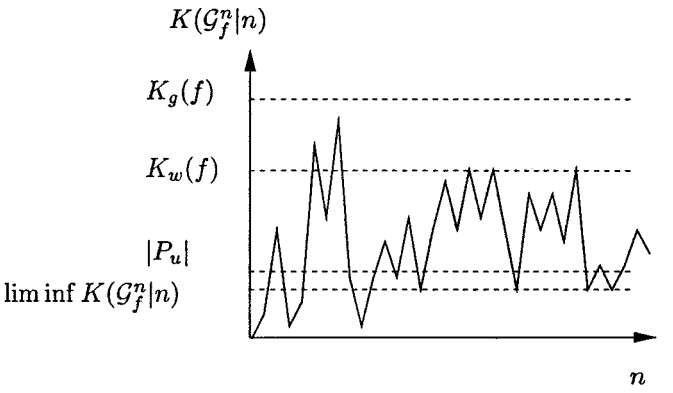


FIG. 3.3: *Complexité de* $\mathcal{G}_f^n$ *sachant* n

Afin de démontrer le théorème 4.4, nous avons besoin de quelques résultats préliminaires. Dans ce qui suit, la notation $\text{Step}(P, x, t)$ désigne le programme qui simule t pas du programme P avec x en entrée, et produit $P(x) + 1$ s'il y a convergence du programme P durant la simulation, 0 sinon.

Définition 4.4 *Soit* P *un programme et* f *une fonction. Nous définissons les relations d'extension* $P \subset f$ *et, pour tout* n, $P \subset_n f$ *par :*

$$(P \subset f) \Leftrightarrow (si\ P(x)\ converge,\ alors\ P(x) = f(x)),$$

$$(P \subset_n f) \Leftrightarrow (\forall\ x \leq n\ Step(P,x,n) \neq 0 \Rightarrow Step(P,x,n) = f(x) + 1).$$

$P \subset_n f$ représente l'observation de la relation $P \subset f$ après n étapes de calcul de P pour les x inférieurs ou égaux à n.

Définition 4.5 *Nous disons que i est* bon *pour k, $i \leq 2^k$, si*

$$\exists\ un\ programme\ P, |P| < k, (P \subset f_i\ et\ \forall\ i' \leq 2^k, i' \neq i, P \not\subset f_{i'}).$$

i est bon pour k signifie qu'il existe un programme de longueur strictement inférieure à k définissant une fonction partielle récursive qui, d'une part, coïncide sur son ensemble de définition avec la fonction f_i, et qui, d'autre part, ne coïncide sur son ensemble de définition avec aucune autre fonction $f_{i'}$, $i' \leq 2^k$ et $i' \neq i$.

Définition 4.6 *Nous disons que i est n-*bon *pour k, $i \leq 2^k$, si*

$$\exists\ un\ programme\ P, |P| < k, (P \subset_n f_i\ et\ \forall\ i' \leq 2^k, i' \neq i, P \not\subset_n f_{i'}).$$

C'est une définition similaire à la précédente. Mais ici, on ne considère que les n premiers pas de calcul des programmes, et on ne compare les fonctions que sur les x inférieurs à n.

Définition 4.7 *Nous disons que i est* mauvais *pour k (respectivement n-*mauvais *pour k), $i \leq 2^k$, s'il n'est pas bon pour k (respectivement s'il n'est pas n-bon pour k)* i.e. *si, respectivement,*

$$\forall\ P, |P| < k, (P \subset f_i) \Rightarrow (\exists\ i' \leq 2^k, i' \neq i\ P \subset f_{i'}),$$
$$\forall\ P, |P| < k, (P \subset_n f_i) \Rightarrow (\exists\ i' \leq 2^k, i' \neq i\ P \subset_n f_{i'}).$$

Nous désignons par $i(k)$ (respectivement par $i(n,k)$) le plus petit i qui soit mauvais pour k (respectivement le plus petit i qui soit n-mauvais pour k).

i est mauvais pour k signifie soit qu'il n'existe aucun programme de longueur strictement inférieure à k définissant une fonction partielle récursive coïncidant avec f_i sur son domaine de définition, soit qu'il existe un tel programme, mais qu'alors la fonction qu'il définit coïncide avec une autre fonction de la famille.

i est n-mauvais pour k a un sens similaire, mais en ne considérant que les n premiers pas de calcul des programmes, et en ne comparant les fonctions que sur les x inférieurs à n.

Lemme 4.3 *Étant donné k, il existe $i \leq 2^k$ mauvais pour k.*

Preuve. Soit k fixé. Supposons que pour tout $i \leq 2^k$, i soit bon pour k. Alors pour tout $i \leq 2^k$ il existe un programme P_i de longueur strictement inférieure à k tel que $P_i \subset f_i$. Les fonctions f_i étant étant deux à deux distinctes, nous avons $2^k + 1$ fonctions f_i, $i \leq 2^k$, distinctes, mais seulement $2^k - 1$ programmes de longueur strictement inférieure à k. Par conséquent, il existe $i_0 \leq 2^k$ et $i'_0 \leq 2^k$, $i_0 \neq i'_0$, tels que $P_i i_0 = P_{i'_0}$. Donc $P_{i_0} \subset f_{i_0}$ et $P_{i_0} \subset f_{i'_0}$, ce qui est en contradiction avec le fait que i_0 soit bon pour k. □

La relation $\subset_n$ est une approximation de la relation $\subset$, comme le prouve le lemme suivant. Il faut noter que la vitesse de convergence de $\subset_n$ vers $\subset$ est non calculable.

Lemme 4.4 *Étant donné k, pour n suffisamment grand, $i(n,k)$ est stationnaire et égal à $i(k)$.*

Preuve. k étant fixé, nous allons montrer que pour n suffisamment grand :

1. tous les entiers de 0 à $i(k) - 1$ sont n-bons pour k ;
2. $i(k)$ est n-mauvais pour k.

Étape 1 : $i(k)$ étant le plus petit indice mauvais pour k, tous les entiers de 0 à $i(k) - 1$ sont bons pour k. Considérons d'abord l'entier 0. Comme il est bon pour k, nous sommes sûrs qu'il existe un programme P, $|P| < k$, tel que $P \subset f_0$. Clairement, $P \subset_n f_0$. Nous sommes certains de ne pas avoir $P \subset f_i$ pour tout $i \neq 0$ et $i \leq 2^k$. Par conséquent, il existe n_0 tel que pour tout $n \geq n_0$ il est impossible d'avoir $P \subset_n f_i$, $i \neq 0$ et $i \leq 2^k$. En d'autres termes, pour tout $n \geq n_0$, 0 est n-bon pour k.

Nous pouvons itérer le processus et trouver $n_1 \geq n_0$ tel que pour tout $n \geq n_1$, 1 est n-bon pour k, et ainsi de suite jusqu'à $i(k) - 1$.

Étape 2 : Examinons maintenant l'entier $i(k)$. $i(k)$ est mauvais pour k. Cela signifie soit qu'il n'existe pas de programme de taille inférieure à k qui calcule une restriction de $f_{i(k)}$, soit qu'il existe un tel programme, mais alors ce programme calcule une restriction d'une autre fonction f_i, $i \leq 2^k$. Dans le premier cas, il nous suffit d'attendre que tous les programmes de longueur inférieure à k donnent une sortie non compatible avec $f_{i(k)}$. Alors, pour tout n suffisamment grand, $i(k)$ est n-mauvais pour k. Dans le second cas, si P calcule la restriction de deux fonctions f_i, alors sa simulation sur n pas le fait également, et pour tout n nous avons $i(k)$ n-mauvais pour k.

En conclusion, pour n suffisamment grand, le plus petit indice n-mauvais pour k est $i(k)$. □

Preuve du théorème 4.4. Nous prouvons les inégalités suivantes :

1. $\forall\, k\ K_g(f_{i(k)}) \geq k$,

2. $\forall\ k\ \limsup_{n\to\infty} K(\mathcal{G}^n_{f_{i(k)}}|n) \leq \log(k) + \mathcal{O}(1)$.

Étape 1 : Supposons qu'il existe k tel que $K_g(f_{i(k)}) < k$. Alors il existe un programme $P(x)$, $|P| < k$, calculant $f(x)$ pour tout x[1]. Donc $P \subset f_{i(k)}$. Comme $i(k)$ est mauvais pour k, alors il existe $i' \neq i(k)$ tel que $P \subset f_{i'}$, *i.e.*tel que $f_{i(k)} \subset f_{i'}$, donc $f_{i(k)} = f_{i'}$ ce qui est faux d'après l'hypothèse faite sur la famille $\mathcal{F}$.

Étape 2 : Comme $\mathcal{F}$ est une famille raisonnable de fonctions, il existe un programme $\mathcal{P}$ tel que :

$$\forall\ i, x,\ \mathcal{P}(i,x) \text{ s'arrête et produit } f_i(x).$$

Le programme $P(n,k)$ suivant accepte deux entrées n et k et calcule $i(n,k)$ en utilisant le programme $\mathcal{P}$:

Programme $P(n,k)$
Simuler n pas de tous les programmes de longueur inférieure à k sur toutes les entrées inférieures ou égales à n.
Calculer les 2^k premières fonctions f_i sur $0,\ldots,n$ en utilisant le programme $\mathcal{P}$.
Comparer les résultats et sortir le plus petit i qui soit mauvais pour k.
Fin

D'après le lemme 4.4, $P(n,k)$ calcule $i(n,k)$ pour tout n et tout k.

Comme nous avons supposé que la famille $\mathcal{F}$ est récursive, nous pouvons calculer $\mathcal{G}^n_{f_i}$ en utilisant le programme $\mathcal{P}$ sur les entrées i et n. Par conséquent :

$$\forall\ i\ K(\mathcal{G}^n_{f_i}|n) \leq K(i|n) + \mathcal{O}(1).$$

De plus, puisque pour tout n,k, $P(n,k)$ calcule $i(n,k)$, nous avons :

$$\begin{aligned} \forall\ n,k\ K(\mathcal{G}^n_{f_{i(n,k)}}|n) &\leq |P(.,k)| + \mathcal{O}(1) \\ &\leq \log(k) + \mathcal{O}(1). \end{aligned}$$

D'après le lemme 4.4, k étant donné, il existe n_k tel que :

$$\forall\ n \geq n_k\ i(n,k) = i(k).$$

1. Remarquons que nous utilisons ici une définition alternative de modèle global.

En conséquence, nous avons :

$$\forall\ k\ \exists\ n_k\ \forall\ n \geq n_k\ K(\mathcal{G}^n_{f_{i(k)}}|n) \leq \log(k) + \mathcal{O}(1).$$

Finalement, nous obtenons :

$$\limsup_{n\to\infty} K(\mathcal{G}^n_{f_{i(k)}}|n) \leq \log(k) + \mathcal{O}(1).$$

□

Ainsi, pour des familles raisonnables de fonctions, la complexité du graphe d'une fonction n'est pas une bonne approximation de la complexité de la fonction. Cependant, ainsi que l'exprime le théorème 4.5, cette approximation est correcte pour la plupart des fonctions de la famille.

Théorème 4.5 *Soit $\mathcal{F} = \{f_i\}_{i\in\mathbb{N}}$ une famille raisonnable de fonctions. Pour tout ν et tout n, la proportion de fonctions, parmi les k premières fonctions f_i, pour lesquelles la différence entre la complexité globale et la complexité faible est supérieure à ν est au plus de $\frac{A}{2^\nu}$, A étant une constante dépendant uniquement de la famille $\mathcal{F}$. De façon plus formelle :*

$$\exists\ A\ \forall\ \nu\ \forall\ k\ \frac{Card\{i \leq k, K_g(f_i) - K_w(f_i) \geq \nu\}}{k} \leq \frac{A}{2^\nu}.$$

Preuve. Comme $\mathcal{F}$ est une famille récursive de fonctions récursives, il existe un programme calculant f_i pour tout i, et par conséquent :

$$\exists\ A\ \forall\ i\ K_g(f_i) \leq log(i) + A. \tag{3.1}$$

Puisque $K_w(f_i) = \limsup_{n\to\infty} K(\mathcal{G}^n_{f_i}|n)$ nous avons :

$$\forall\ i\ \exists\ n'_i\ \forall\ n \geq n'_i\ K_w(f_i) \geq K(\mathcal{G}^n_{f_i}|n). \tag{3.2}$$

D'après les équations 3.1 et 3.2 nous avons :

$$\exists\ A\ \forall\ i\ \exists\ n'_i\ \forall\ n \geq n'_i\ K_g(f_i) - K_w(f_i) \leq \log(i) + A - K(\mathcal{G}^n_{f_i}|n). \tag{3.3}$$

Soient k et ν fixés. Comme toutes les fonctions de la famille $\mathcal{F}$ sont distinctes, il existe n_k tel que, pour tout $n \geq n_k$, toutes les restrictions de graphes $(\mathcal{G}^n_{f_i})_{i<k}$ sont différentes.

Désignons par $S(\nu, k)$ l'ensemble $\{i \leq k, K_g(f_i) - K_w(f_i) \geq \nu\}$. Pour $n \geq \max\{n_k, (n'_i)_{i\leq k}\}$, d'après l'équation 3.3 nous avons :

$$\begin{aligned} S(\nu,k) &\subset \{i \leq k, \log(i) + A - K(\mathcal{G}^n_{f_i}|n) \geq \nu\} \\ &= \{i \leq k, K(\mathcal{G}^n_{f_i}|n) \leq \log(i) + A - \nu\} \\ &\subset \{i \leq k, K(\mathcal{G}^n_{f_i}|n) \leq \log(k) + A - \nu\}. \end{aligned}$$

Comme $n \geq n_k$, tous les graphes restreints $(\mathcal{G}^n_{f_i})_{i \leq k}$ sont distincts. Par conséquent, tous les plus petits programmes calculant ces graphes restreints sont différents. Finalement, nous avons donc :

$$Card(S(\nu, k)) \leq 2^{\log(k)+A-\nu}.$$

□

Cinquième partie

Séquences aléatoires et transducteurs déterministes

Chapitre 1

Introduction

Dans la partie précédente, nous avons modélisés les processus à l'origine de flots de données par des fonctions récursives ou, en d'autres termes, par des programmes toujours convergents, sans paramètres d'entrée et disposant *a priori* d'autant d'espace mémoire et de temps que nécessaire pour effectuer leurs calculs. Nous introduisons dans cette partie des contraintes de temps ainsi que des paramètres d'entrées. Nous avons donc un flot d'entrée e_1,...,e_t... e_t représentant l'entrée à l'instant t et un flot de sortie s_1,...,s_t... s_t étant calculé « instantanément », c'est-à-dire en un temps borné. Sous ces nouvelles contraintes, les processus se comportent comme des transducteurs rationnels, c'est-à-dire des automates possédant un nombre d'états fini, passant d'un état à l'autre suivant les événements survenant en entrée et produisant, lors de ces changements d'états, des événements en sortie qui constituent le flot de données observé.

En quoi l'observation du flot de sortie peut-il nous renseigner sur la structure interne du processus, *i.e.* du transducteur sous-jacent ? Il est clair que le flot de sortie dépend d'une part du transducteur, d'autre part du flot d'entrée. Pour espérer obtenir le maximum d'information sur le transducteur, il faut provoquer toutes les situations possibles en entrée et observer les réactions du transducteurs. Ce sera par exemple le cas si le flot d'entrée est un flot *aléatoire*. Dans un flot aléatoire, tout événement finit par se produire et par conséquent le transducteur est soumis à toutes les situations possibles.

Les processus étant déterministes et étudiés sur un seul flot (infini) d'entrée, nous consacrons cette partie à l'étude du comportement des transducteurs rationnels déterministes sur des séquences d'entrées aléatoires, en nous limitant aux transducteurs lettre à lettre, c'est-à-dire produisant exactement une lettre en sortie pour chaque lettre consommée en entrée, ce qui revient à envisager un synchronisme total entre le flot d'entrée et le flot de sortie.

Dans le chapitre 2 nous introduisons une classification des états des transducteurs, avec notamment les notions d'*états complets*, acceptant n'importe quelle entrée, d'*états récurrents* (atteints infiniment souvent) et d'*états fré-*

quents (états récurrents atteints avec une fréquence strictement positive), notions inspirées de la théorie des processus de Markov [MT96].

Dans le chapitre 3 nous établissons des propriétés liées à la classification précédente, ce qui nous permet d'obtenir une condition nécessaire et suffisante sur la structure d'un transducteur pour qu'il accepte une séquence aléatoire donnée en entrée. Puis nous montrons qu'il est possible de déterminer à l'aide d'un programme et en un temps fini si deux transducteurs produisent la même sortie. Enfin, a étant une séquence d'entrée aléatoire et s son image par un transducteur T dont on connaît seulement un majorant du nombre d'états, nous montrons qu'il est possible d'inférer une classe de transducteurs ayant le même comportement que T sur a, c'est-à-dire transformant a en s.

Dans le dernier chapitre, nous montrons (voir le théorème 5.3) que des propriétés structurelles du transducteur déterminent le type de sortie obtenue lorsque l'entrée est une séquence aléatoire: la séquence de sortie peut être aléatoire, non-aléatoire et non-récursive, ou récursive (et dans ce cas ultimement périodique) selon que le transducteur ne possède pas d'état partiellement indifférent, possède un état partiellement indifférent et un état discriminant ou possède un état partiellement indifférent mais pas d'état discriminant. Nous montrons en outre que le problème de décision de la présence de tels états dans un transducteur se résout en un temps polynômial en le nombre d'états du transducteur (voir les théorèmes 5.4 et 5.5).

Chapitre 2

Classification des états d'un transducteur

Nous considérons des transducteurs classiques, lettre à lettre, c'est-à-dire produisant en sortie une lettre et une seule à chaque fois qu'une lettre est lue en entrée, et ne comportant qu'un nombre Q fini d'états. La figure 2.1 montre un exemple d'un tel transducteur. L'état q_0 est l'état initial. Si la première lettre d'entrée est 0, alors le transducteur passe dans l'état q_1 en émettant la lettre 1 en sortie.

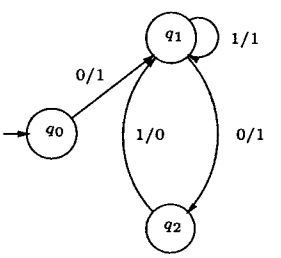


FIG. 2.1: *Un exemple de transducteur rationnel lettre à lettre*

Nous mettons en entrée de ces transducteurs des séquences binaires infinies. Si lors de la lecture d'une séquence le transducteur atteint un état d'où ne sort aucune transition étiquetée par la lettre d'entrée qui suit, alors le transducteur s'arrête et nous dirons qu'il rejette la séquence d'entrée, sinon nous dirons qu'il l'accepte. Ainsi le transducteur de la figure 2.1 accepte la séquence 01^∞ mais rejette toute séquence commençant par 1. Les transducteurs ne comportent aucun état final.

Nous allons maintenant donner quelques définitions et résultats préliminaires nécessaires à l'établissement des principaux résultats.

2.1 États complets

Définition 5.1 (état complet) *Un état* complet *est un état duquel sortent deux transitions étiquetées par* 0 *et* 1. *Un ensemble d'états est complet si tous les états de cet ensemble sont complets.*

Considérons le pré-ordre sur l'ensemble des états défini par :

$$q < q' \Leftrightarrow q' \text{ est accessible à partir de } q,$$

et les classes d'équivalence de ces états ($q \sim q' \Leftrightarrow q < q'$ et $q' < q$). Les classes maximales pour l'ordre quotient $^{<}/_{\sim}$ sont *absorbantes*. En effet, dès lors que l'on atteint un état d'une classe maximale il est impossible de ressortir de cette classe. Il est clair que si une classe absorbante est complète, alors n'importe quelle séquence est acceptée à partir d'un état quelconque de cette classe.

Il est possible de partitionner, par programme, un transducteur en sous-automates $T_1, \ldots, T_N$ et T_P (voir Figure 2.2), où $T_1, \ldots, T_N$ sont les classes absorbantes complètes de T et T_P est l'union des autres classes.

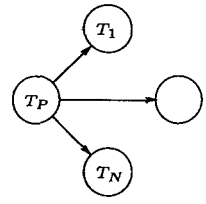


FIG. 2.2: *Classes absorbantes complètes*

2.2 États récurrents et fréquents

Définition 5.2 (état récurrent) *Soit a une séquence acceptée par un transducteur T. Un état de T est* récurrent *pour a s'il est atteint une infinité de fois lorsque a est lue en entrée. Nous désignons par T_a l'ensemble des états de T qui sont récurrents pour a.*

Nous allons maintenant quantifier, par la notion de fréquence, les apparitions d'un chemin d'états ou d'une transition lors de la lecture d'une séquence. Pour cela, nous utilisons la notion d'occurrence d'un chemin d'états $q_1 \ldots q_l$ de longueur finie. Une apparition d'un chemin lors de la lecture d'une séquence est appelée occurrence s'il s'agit de la première apparition de ce chemin ou, dans le cas contraire, si elle ne chevauche pas une occurrence

précédente deu ce chemin. Voyons cela sur l'exemple suivant. Supposons que la suite des états empruntés lors de la lecture d'une séquence soit la suivante :

$$\overbrace{q_1q_2q_3}\overbrace{q_1q_2q_3}q_1q_4q_5\overbrace{q_1q_2q_3}q_1\dots$$

Pour les 13 premières lettres lues de la séquence d'entrée, le chemin $q_1q_2q_3$ apparaît trois fois (apparitions délimitées par les accolades). Aucune apparition ne chevauchant une autre, chaque apparition constitue une occurrence. Nous mettons maintenant en valeur les apparitions du chemin $q_1q_2q_3q_1$ sur la même suite d'états.

$$\overbrace{q_1q_2q_3\underbrace{q_1q_2q_3q_1}}q_4q_5\overbrace{q_1q_2q_3q_1}\dots$$

La première apparition de $q_1q_2q_3q_1$ est une occurrence. La seconde apparition chevauche la première, qui est une occurrence, ce n'est donc pas une occurrence. Enfin, la troisième apparition est une occurrence.

Soit S un ensemble de chemin de longueur finie. Nous définissons la notion d'occurrence d'un chemin de S de la façon suivante. La première apparition de l'un quelconque des chemins de S constitue la première occurrence. La première apparition de l'un quelconque des chemins de S qui survient après la première occurrence sans la chevaucher constitue la seconde occurrence. Et ainsi de suite.

Définition 5.3 (chemin fréquent) *Soit a une séquence acceptée par un transducteur T. Soit $q_1 \dots q_l$ un chemin de T de longueur finie. Nous notons $occ_n^a(q_1 \dots q_l)$ le nombre d'occurrences de ce chemin lors de la lecture des n premières lettres de a. La fréquence d'apparition $f^a(q_1 \dots q_l)$ de ce chemin pour a est définie par :*

$$f^a(q_1 \dots q_l) = \limsup_{n\to\infty} \frac{occ_n^a(q_1 \dots q_l)}{n}.$$

Si $f^a(q_1 \dots q_l) > 0$, ce chemin est dit fréquent *pour a. Dans le cas où ce chemin est constitué d'un seul état q, nous dirons que l'état q est fréquent pour a.*

Définition 5.4 *Soit a une séquence acceptée par un transducteur T. Soit S un ensemble d'états de T. Soient c_1, c_2, c_3, ... des éléments de S. Nous notons $occ_n^{a,S}(c_1$ ou c_2 ou $c_3 \dots)$ le nombre d'apparitions de c_1, c_2, $c_3 \dots$ qui sont des occurrences de chemins de S lors de la lecture des n premières lettres de a. Nous définissons également*

$$f^{a,S}(c_1 \text{ ou } c_2) = \limsup_{n\to\infty} \frac{occ_n^{a,S}(c_1 \text{ ou } c_2)}{n}.$$

Définition 5.5 (transition fréquente) *Soit a une séquence acceptée par un transducteur T. Soit $q_1 \dots q_l$ un chemin de T de longueur finie et t une transition sortant de l'état q_l. L'événement « occurrence du chemin $q_1 \dots q_l$ suivi d'une occurrence de la transition t » est appelé plus simplement « occurrence de t après $q_1 \dots q_l$ ».*

Lors de la lecture des n premières lettres de a, le nombre d'occurrences de t après $q_1 \dots q_l$ est noté $occ_n^a(t|q_1 \dots q_l)$. La fréquence $f^a(t|q_1 \dots q_l)$ de t après $q_1 \dots q_l$ pour a est définie par :

$$f^a(t|q_1 \dots q_l) = \limsup_{n \to \infty} \frac{occ_n^a(t|q_1 \dots q_l)}{n}.$$

Si $f^a(t|q_1 \dots q_l) > 0$, la transition t est dite fréquente après $q_1 \dots q_l$ pour a.

Nous employons l'expression « occurrence de t après $q_1 \dots q_l$ » pour ne pas alourdir le texte. Il faut toutefois bien garder à l'esprit que deux occurrences successives de t après des apparitions de $q_1 \dots q_l$ ne se chevauchant pas ne constituent pas nécessairement deux occurrences de t après $q_1 \dots q_l$ comme le montre l'exemple suivant. Supposons que la suite des états empruntés lors de la lecture d'une séquence soit

$$\overbrace{q_1 q_2 q_3 q_1} q_4 \underbrace{q_1 q_2 q_3 \overbrace{q_1 q_2 q_3 q_1} q_4} \dots$$

Soit t la transition menant de q_1 à q_4. Ce qui est sous l'accolade la plus à droite ne constitue pas une occurrence de t après $q_1 q_2 q_3 q_1$. En effet, l'apparition de $q_1 q_2 q_3 q_1$ correspondante n'est pas une occurrence : elle chevauche une occurrence précédente.

Lemme 5.1 *Soit a une séquence acceptée par un transducteur T, $q_1 \dots q_l q_{l+1}$ un chemin de T de longueur finie et t la transition menant de q_l à q_{l+1}. Si t après $q_1 \dots q_l$ est fréquente pour a, alors le chemin $q_1 \dots q_{l+1}$ est fréquent pour a,* i.e.

$$f^a(t|q_1 \dots q_l) > 0 \Rightarrow f^a(q_1 \dots q_l q_{l+1}) > 0.$$

Preuve. Nous allons établir l'inégalité suivante, qui implique clairement le résultat que nous cherchons à montrer.

$$\forall\, n,\ occ_n^a(t|q_1 \dots q_l) \leq 2 occ_n^a(q_1 \dots q_l q_{l+1}).$$

Notons d'abord qu'une occurrence de $q_1 \dots q_l$ suivie de t commence nécessairement à partir d'un état appartenant à une occurrence de $q_1 \dots q_l q_{l+1}$. Comme deux occurrences de $q_1 \dots q_l$ suivies de t ne peuvent se chevaucher, il y a au plus deux occurrences de $q_1 \dots q_l$ suivies de t commençant par un état appartenant à une occurrence de $q_1 \dots q_l q_{l+1}$. □

Lemme 5.2 *Soit a une séquence acceptée par un transducteur T. T contient au moins un état fréquent pour a.*

Preuve. Raisonnons par l'absurde et supposons qu'aucun état de T ne soit fréquent pour a, *i.e.* que pour tout $i = 1, \ldots, Q$, $f^a(q_i) = 0$. Soit $\epsilon = \frac{1}{2Q}$. Nous avons alors :

$$\begin{aligned} \exists\, n_0\ \forall\, n \geq n_0\ \forall\, q_i \in\ T \quad \frac{occ_n^a(q_i)}{n} &\leq \epsilon \\ \sum_{i=1}^{Q} \frac{occ_n^a(q_i)}{n} &\leq \epsilon Q. \end{aligned}$$

Comme $\sum_{i=1}^{Q} \frac{occ_n^a(q_i)}{n} = 1$ et $\epsilon = \frac{1}{2Q}$, cela nous mène à la contradiction $1 \leq \frac{1}{2}$. □

Chapitre 3

Inférence de transducteurs

3.1 Acceptation d'une séquence aléatoire

3.1.1 Transducteurs acceptant une séquence aléatoire

Nous nous intéressons aux conditions que doivent nécessairement vérifier les transducteurs lettre à lettre pour accepter une séquence aléatoire. Nous montrons en particulier que les états récurrents pour une séquence aléatoire donnée forment une classe d'équivalence absorbante et complète (lemme 5.7).

Lemme 5.3 *Soit a une séquence aléatoire acceptée par un transducteur T. Tout état de T fréquent pour a est complet.*

Preuve. Soit q un état fréquent de T. Supposons qu'il ne soit pas complet. Il existe donc une seule transition t sortant de cet état, étiquetée par exemple par 0 en entrée, et menant à un état q'. Montrons que cela implique que a n'est pas aléatoire. Considérons la séquence a' définie de la façon suivante : c'est la séquence a dans laquelle est supprimée toute lettre lue lors du passage par la transition t. Le programme suivant, qui utilise le transducteur T, produit a en sortie quand on lui fournit a' en entrée.

```
Programme P(a')
    Répéter
        Si T est dans l'état q_l Alors
            Sortir 0
            Placer T dans l'état q'
        Sinon
            Lire une lettre b de a' et sortir b
            Simuler T sur l'entrée b
        Fin Si
    Fin Répéter
Fin
```

Ce programme recopie en sortie les lettres de a'. Il détecte les endroits de a où une lettre a été effacée en effectuant une simulation de T sur l'entrée a'. Quand T atteint l'état q, cela signifie qu'une lettre a été effacée. Cette lettre étant nécessairement 0, P produit un 0 en sortie. Comme de q on ne peut atteindre que l'état q', P place T dans l'état q' avant de continuer de la même manière.

Comme q est fréquent pour a nous avons $\limsup_{n\to\infty} \frac{occ_n^a(q)}{n} = \alpha$ avec $\alpha > 0$. Donc il existe une suite $(v_n)_{n\in\mathbb{N}}$ et n_0 tels que pour tout $n > n_0$ nous ayons $\frac{occ_{v_n}^a(q)}{v_n} \geq \frac{\alpha}{2}$. Soit $u_n = v_n - occ_{v_n}^a(q)$. Pour tout $n > n_0$ nous avons $u_n \leq \lambda v_n$ avec $\lambda = 1 - \frac{\alpha}{2} < 1$. Le programme P est tel que pour tout n nous avons $P(a'_{1:u_n}) = a_{1:v_n}$. Les hypothèses du lemme 2.1 sont satisfaites et par conséquent nous concluons que a n'est pas aléatoire. □

Lemme 5.4 *Soit a une séquence aléatoire acceptée par un transducteur T. Pour tout chemin $q_1 \dots q_l$ de longueur finie et toute transition t sortant de q_l, t après $q_1 \dots q_l$ est fréquente pour a si $q_1 \dots q_l$ est fréquent pour a,* i.e.

$$f^a(q_1 \dots q_l) > 0 \Rightarrow f^a(t|q_1 \dots q_l) > 0.$$

Preuve. Soit $q_1 \dots q_l$ un chemin de longueur finie fréquent pour a. Nous avons $occ_n^a(q_l) \geq occ_n^a(q_1 \dots q_l)$ pour tout n, donc $f^a(q_l) \geq f^a(q_1 \dots q_l)$: q_l est fréquent pour a. D'après le lemme 5.3, q_l est donc complet. Soit t_0 (respectivement t_1) la transition acceptant 0 (respectivement 1) à partir de q_l et soient q^0 et q^1 les deux états atteints respectivement par ces deux transitions.

Supposons qu'une de ces transitions, par exemple t_0, ne soit pas fréquente après $q_1 \dots q_l$ pour a : $f^a(t_0|q_1 \dots q_l) = 0$. Considérons maintenant la séquence a' définie de la façon suivante : c'est la séquence a dans laquelle :

1. toute lettre 1 lue lors d'une occurrence de t_1 après $q_1 \dots q_l$ est supprimée ;
2. toute lettre 0 lue lors d'une occurrence de t_0 après $q_1 \dots q_l$ est remplacée par un entier, codée de façon préfixe : cet entier est le nombre d'occurrences successives de t_1 après $q_1 \dots q_l$ avant la prochaine occurrence de t_0 après $q_1 \dots q_l$.

Le programme suivant, qui utilise T, produit a en sortie quand on lui fournit a' en entrée.

```
Programme P(a')
    Initialiser n avec le nombre d'occurrences de t_1 après
    q_1 ... q_l ayant lieu avant la première occurrence de t_0
    après q_1 ... q_l pour a
    Répéter
        Si une occurrence de q_1 ... q_l vient d'avoir lieu
        Alors
            Si n = 0 Alors
                Réinitialiser n avec l'entier codé de façon
                préfixe à la position courante dans a'
                Remplacer dans a' les lettres codant cet
                entier par un 0
            Sinon
                n = n - 1
                Insérer à la position courante dans a' un 1
            Fin Si
        Fin Si
        b = prochaine lettre de a'
        Sortir b
        Simuler T sur b
    Fin Répéter
Fin
```

Ce programme recopie en sortie les lettres de a' tant que celles-ci correspondent aux lettres de a quand il ne détecte pas un endroit où les séquences a et a' diffèrent. Pour faire cette détection, il maintient un jour un compteur n contenant le nombre d'occurrences successives de t_1 après $q_1 \dots q_l$ restant à venir avant la prochaine occurrence de t_0 après $q_1 \dots q_l$. Après qu'une occurrence de $q_1 \dots q_l$ a eu lieu, le programme teste la valeur du compteur n pour déterminer ce qu'il doit faire. Si cette valeur est strictement positive (respectivement nulle), cela signifie que c'est la transition t_1 (respectivement t_0) qui doit être empruntée et que la lettre manquante de a est 1 (respectivement 0). Avant de poursuivre, le programme doit mettre à jour son compteur n. Dans le premier cas, il se contente de le décrémenter d'une unité. Dans le second cas, il l'initialise avec la valeur codée de façon préfixe à la position courante dans a' et qui, par construction de a', indique bien le nombre d'occurrences de t_1 après $q_1 \dots q_l$ qui doivent avoir lieu avant la prochaine occurrence de t_0 après $q_1 \dots q_l$.

L'existence de ce programme implique que a n'est pas aléatoire. Pour arriver à cette conclusion, nous allons maintenant montrer que les hypothèses du lemme 2.1 sont satisfaites. Soit u_n la longueur du préfixe de a' ayant servi au calcul des n premières lettres de a par le programme P. Soit ν_i le nombre d'occurrences de la transition t_1 après $q_1 \dots q_l$ entre les $(i-1)$-ième et i-

ième occurrences de la transition t_0 après $q_1 \dots q_l$. Chaque mot $a'_{1:u_n}$ est construit à partir du mot $a_{1:n}$ par suppression de la lettre 1 apparaissant à chaque occurrence de la transition t_1 après $q_1 \dots q_l$ et par remplacement de la lettre 0 apparaissant à chaque occurrence de la transition t_0 après $q_1 \dots q_l$ par l'écriture auto-délimitée d'un ν_i qui annonce le nombre d'occurrences t_1 après $q_1 \dots q_l$ qui vont suivre. Par conséquent,

$$u_n = n - occ_n^a(q_1 \dots q_l) + \sum_{i=1}^{z_n} l(\overline{\nu_i}),$$

$z_n = occ_n^a(t_0|q_1 \dots q_l)$ désignant le nombre d'occurrences de la transition t_0 après $q_1 \dots q_l$ sur les n premières lettres de a.

La longueur de l'écriture auto-délimitée de ν_i étant $2\log(\nu_i) + 1$, nous avons :

$$\sum_{i=1}^{z_n} l(\overline{\nu_i}) \leq 2\log\left(\prod_{i=1}^{z_n} \nu_i\right) + z_n. \tag{3.1}$$

Le produit de k nombres dont la somme est une constante S est maximal quand ces k nombres sont tous égaux à $\frac{S}{k}$. Dans notre cas, cette somme est le nombre d'occurrences de t_1 après $q_1 \dots q_l$ pour les n premières lettres de a, soit $occ_n^a(t_1|q_1 \dots q_l)$. Le produit apparaissant dans l'équation 3.1 est donc majoré par $\left(\frac{occ_n^a(t_1|q_1 \dots q_l)}{z_n}\right)^{z_n}$, lui-même majoré par $\left(\frac{n}{z_n}\right)^{z_n}$. On obtient par conséquent la majoration suivante de u_n :

$$u_n \leq n - occ_n^a(q_1 \dots q_l) + 2z_n \log\left(\frac{n}{z_n}\right) + z_n.$$

En divisant chaque membre par n il vient :

$$\frac{u_n}{n} \leq 1 - \frac{occ_n^a(q_1 \dots q_l)}{n} + 2\frac{\log\left(\frac{n}{z_n}\right)}{\frac{n}{z_n}} + \frac{z_n}{n}.$$

Comme la transition t_0 n'est pas fréquente après $q_1 \dots q_l$ pour a, $\frac{z_n}{n}$ tend vers 0 et par conséquent le terme $2\frac{\log\left(\frac{n}{z_n}\right)}{\frac{n}{z_n}} + \frac{z_n}{n}$ également. Comme $q_1 \dots q_l$ est fréquent pour a, nous avons $\limsup \frac{occ_n^a(q_1 \dots q_l)}{n} > 0$. Donc il existe $\beta > 0$ tel que, pour une infinité de n, nous ayons $\frac{u_n}{n} \leq 1 - \beta$. Les hypothèses du lemme 2.1 sont vérifiées, ce qui nous permet de conclure que a n'est pas aléatoire.

□

Lemme 5.5 *Soit a une séquence aléatoire acceptée par un transducteur T. Tout état accessible à partir d'un état fréquent pour a est fréquent pour a.*

Preuve. Soit q un état fréquent pour a. D'après le lemme 5.3, q est complet. Soient t_0 et t_1 les deux transitions sortant de q. Soient q_0 et q_1 les états atteints respectivement par ces deux transitions. Examinons le cas de q_0. D'après le lemme 5.4, q étant fréquent, $f^a(t_0|q) > 0$. Or

$$occ_n^a(q_0) \geq occ_n^a(t_0|q),$$

et par conséquent $f^a(q_0) \geq f^a(t_0|q)$. Donc q_0 est fréquent pour a. Le même raisonnement montre que q_1 est également fréquent pour a.

De proche en proche, nous montrons ainsi que tous les états accessibles à partir de q sont fréquents pour a. □

Lemme 5.6 *Soit a une séquence aléatoire acceptée par un transducteur T. Un état de T est récurrent pour a si et seulement s'il est fréquent pour a.*

Preuve. La condition suffisante est évidente. Montrons la condition nécessaire. Soit q un état récurrent pour a. D'après le lemme 5.2, il existe un état q' fréquent pour a. q et q' étant tous deux récurrents pour a, q est accessible à partir de q'. q' étant fréquent, nous pouvons affirmer grâce au lemme 5.5 que q est fréquent pour a. □

Lemme 5.7 *Soit a une séquence aléatoire acceptée par un transducteur T. Les états de T récurrents pour a forment une classe absorbante complète.*

Preuve. D'après le lemme 5.6 cela revient à montrer que les états de T qui sont fréquents pour a forment une classe absorbante complète.

1. Les états fréquents pour a forment une classe absorbante. En effet,
   (a) deux états fréquents quelconques sont nécessairement accessibles l'un à partir de l'autre, donc appartiennent à la même classe ;
   (b) d'après le lemme 5.5 tout état accessible à partir d'un état fréquent pour a est fréquent pour a.
2. D'après le lemme 5.3, cette classe est complète.

□

Lemme 5.8 *Soit a une séquence aléatoire acceptée par un transducteur T. Soit $q_1 \dots q_l$ un chemin de longueur finie. Si q_1 est récurrent pour a, alors $q_1 \dots q_l$ est fréquent pour a.*

Preuve. Nous faisons un raisonnement par récurrence sur la longueur l du chemin.

Etape 1 : Pour $l = 1$, cela revient à montrer que si q est récurrent pour a, alors q est fréquent pour a. D'après le lemme 5.6, c'est vrai.

Etape 2 : Supposons que pour tout chemin de longueur inférieure ou égale à l, ce soit vrai. Soit $q_1 \dots q_l q_{l+1}$ un chemin de longueur $l+1$, q_1 étant récurrent pour a. Soit t la transition menant de q_l à q_{l+1}. $q_1 \dots q_l$ est un chemin de longueur l commençant par un état récurrent, donc par hypothèse de récurrence, il est fréquent. D'après le lemme 5.4, t après $q_1 \dots q_l$ est fréquente pour a. Nous en concluons, grâce au lemme 5.1, que $q_1 \dots q_l q_{l+1}$ est fréquent pour a. □

3.1.2 Acceptation d'une séquence aléatoire

Théorème 5.1 *Soit a une séquence aléatoire et T un transducteur lettre à lettre. T accepte a si et seulement si T atteint une classe absorbante complète lors de la lecture de a.*

Preuve. Soit a une séquence aléatoire et T un transducteur lettre à lettre.

Condition nécessaire : Supposons que T accepte une séquence a aléatoire. T comporte au moins un état récurrent pour a. Or d'après le lemme 5.7, l'ensemble des états récurrents forme une classe absorbante complète.

Condition suffisante : Supposons maintenant que T atteint un classe absorbante complète lors de la lecture de a. Comme cette classe est absorbante, le transducteur ne peut en sortir. Comme tous les états de cette classe sont complets, à partir du moment où le transducteur atteint cette classe, il accepte n'importe quelles lettres d'entrée. □

Une conséquence de ce théorème est qu'il est possible de déterminer, par programme, si un transducteur rationnel lettre à lettre accepte ou non une séquence aléatoire.

Corollaire 5.1 *Il existe un programme* ACCEPT *qui, pour tout transducteur rationnel lettre à lettre T et toute séquence aléatoire a, prend T en entrée, utilise a comme oracle, s'arrête, et sort oui si et seulement si T accepte a.*

$$\exists \text{ un programme ACCEPT } \forall\, T\ \forall\, a \text{ aléatoire} \begin{cases} \text{ACCEPT}^a(T) \text{ s'arrête} \\ \text{ACCEPT}^a(T) = \text{oui} \Leftrightarrow T \text{ accepte } a \end{cases}$$

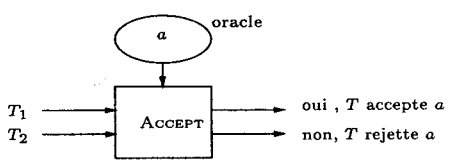


FIG. 3.1: *Un programme* ACCEPT *déterminant si T accepte a*

Preuve. Le programme suivant détermine si un transducteur T accepte une séquence aléatoire a. En effet, un transducteur accepte une séquence aléatoire si et seulement s'il atteint une classe absorbante complète.

```
Programme ACCEPT^a(T)
    Décomposer T en classes absorbantes complètes
    Pour chaque lettre de a Faire
        Simuler T
        Si T n'accepte pas la lettre d'entrée Alors
            Sortir « non »
            Arrêter
        Sinon Si T a atteint une classe absorbante complète
        Alors
            Sortir « oui »
            Arrêter
        Fin Si
    Fin Pour
Fin
```

□

3.2 Problème de l'égalité sur les séquences aléatoires

Nous disons que deux transducteurs T_1 et T_2 sont égaux sur une séquence s (de longueur infinie) si $T_1(s) = T_2(s)$. Notons la différence de cette définition avec la définition de l'équivalence des transducteurs : T_1 et T_2 sont équivalents si $T_1(w) = T_2(w)$ pour tout mot w (de longueur finie).

Le théorème suivant exprime qu'il est possible de déterminer, par programme, si deux transducteurs sont égaux sur une séquence aléatoire.

Théorème 5.2 *Il existe un programme* ÉGALITÉ *qui, pour tous transducteurs rationnels lettre à lettre T_1 et T_2 et toute séquence aléatoire a, prend T_1 et T_2 en entrée, utilise a comme oracle, s'arrête, et sort oui si et seulement si T_1 et T_2 sont égaux sur a.*

$$\exists\ \text{ÉGALITÉ}\ \forall\ T_1, T_2\ \forall\ a \text{ aléatoire} \begin{cases} \text{ÉGALITÉ}^a(T_1, T_2) \text{ s'arrête} \\ \text{ÉGALITÉ}^a(T_1, T_2) = \text{oui} \Leftrightarrow T_1(a) = T_2(a) \end{cases}$$

Le théorème est faux si l'on supprime la contrainte que la séquence a est aléatoire. Considérons par exemple les transducteurs T_1 et T_2 définis sur la figure 3.3.

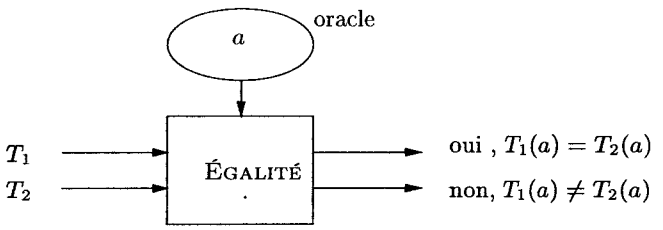


FIG. 3.2: *Un programme* ÉGALITÉ *déterminant si* $T_1(a) = T_2(a)$

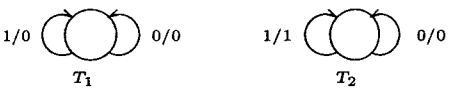


FIG. 3.3: *Deux transducteurs égaux sur* 0^∞ *et non égaux sur* 0^n10^∞

Supposons qu'il existe un programme P qui, pour tous transducteurs T_1 et T_2 et toute séquence s, prend en entrée T_1 et T_2, utilise s comme oracle, s'arrête, et sort oui si et seulement si $T_1(s) = T_2(s)$. Soit $s_1 = 0^\infty$. Comme $T_1(s_1) = T_2(s_1)$, $P^{s_1}(T_1, T_2)$ s'arrête et sort oui en ayant utilisé un nombre fini de lettres de s, toutes contenues dans un préfixe de longueur n de s. Considérons maintenant la séquence $s_2 = 0^n10^\infty$. Puisque les n premières lettres de s_1 et s_2 sont les mêmes, $P^{s_2}(T_1, T_2)$ s'arrête en produisant la même sortie que $P^{s_1}(T_1, T_2)$, *i.e.* oui, bien que, clairement, nous ayons $T_1(s_2) \neq T_2(s_2)$.

Pour prouver le théorème 5.2, nous utilisons la notion de transducteur produit que nous définissons maintenant.

Définition 5.6 *Soient T_1 et T_2 deux transducteurs. Leur transducteur produit $T = T_1 \times T_2$ est défini comme suit : si, dans le transducteur T_1, un état q_1' est accessible à partir d'un état q_1, via une transition t_1, et si, dans le transducteur T_2, un état q_2' est accessible à partir d'un état q_2, via une transition t_2, les transitions t_1 et t_2 acceptant la même lettre d'entrée b_{in} et produisant la même lettre de sortie b_{out}, alors T contient deux états $q = (q_1, q_2)$ et $q' = (q_1', q_2')$, reliés par une transition acceptant b_{in} en entrée et produisant b_{out} en sortie.*

La figure 3.4 illustre ce qu'est le transducteur produit de deux transducteurs T_1 et T_2 très simples.

Ainsi défini, le transducteur produit $T = T_1 \times T_2$ accepte une séquence a si et seulement si les deux transducteurs T_1 et T_2 acceptent a et produisent la même séquence en sortie.

Nous donnons maintenant la preuve du théorème 5.2.

Preuve du théorème 5.2. Soit a une séquence aléatoire. T_1 et T_2 acceptent a

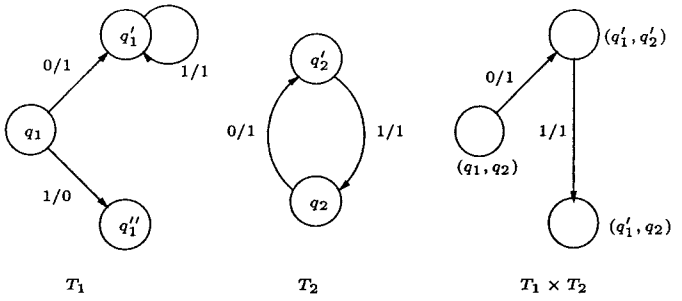


FIG. 3.4: *Un exemple de transducteur produit*

en produisant la même séquence de sortie si et seulement si leur transducteur produit T accepte a, ou encore, d'après le théorème 5.1, si et seulement si T atteint une de ses classes absorbantes complètes lors de la lecture de a. Si T s'arrête (ce qui ne peut avoir lieu que s'il ne se trouve pas dans une de ses classes absorbantes complètes), nous distinguons plusieurs cas :

1. T_1 et T_2 s'arrêtent en même temps que T : comme les deux mots produits jusqu'alors par T_1 et T_2 sont identiques, T_1 et T_2 sont égaux sur a ;

2. l'un des transducteurs T_1 et T_2 s'arrête en même temps que T, l'autre pas : $T_1(a)$ et $T_2(a)$ ont des longueurs différentes, par conséquent T_1 et T_2 ne sont pas égaux sur a ;

3. T_1 et T_2 ne s'arrêtent pas à ce moment, mais produisent en sortie des lettres différentes : par conséquent, ils ne sont pas égaux sur a.

Finalement, T_1 et T_2 sont égaux sur a si et seulement si T atteint l'une de ses classes maximales complètes lors de la lecture de a, ou si T, T_1, T_2 s'arrêtent simultanément. Ces observations nous amènent à proposer le programme ÉGALITÉ suivant.

```
Programme ÉGALITÉ^a(T1, T2)
    Construire le transducteur produit T = T1 × T2
    Partitionner T en classes absorbantes complètes
    Pour chaque lettre de a Faire
        Simuler T, T1, T2
        Si T atteint l'une de ses classes absorbantes
        complètes Alors
            T1 et T2 sont égaux sur la séquence d'entrée
            Arrêter
        Sinon Si T s'arrête Alors
            Si T1 et T2 s'arrêtent Alors
                T1 et T2 sont égaux sur la séquence d'entrée
                Arrêter
            Sinon
                T1 et T2 ne sont pas égaux sur la séquence
                d'entrée
                Arrêter
            Fin Si
        Fin Si
    Fin Pour
Fin
```

□

3.3 Inférence de transducteurs

Considérons un processus, dont le modèle sous-jacent est un transducteur lettre à lettre T. Nous lui fournissons en entrée une séquence aléatoire a, et nous observons la séquence de sortie s qu'il produit. Est-il possible par cette observation de retrouver le transducteur? Moyennant l'hypothèse que T comporte au plus Q états, nous pouvons inférer une classe de transducteurs à laquelle appartient T et dont tous les éléments se comportent de la même façon que T (*i.e.* produisent la même séquence de sortie) sur la séquence d'entrée a. L'algorithme qui suit infère cette classe. Considérons l'ensemble $\mathcal{T}_Q$ des transducteurs lettre à lettre comportant au plus Q états. À l'aide du programme ÉGALITÉ, nous pouvons partitionner $\mathcal{T}_Q$ en classes de transducteurs égaux sur la séquence a. Considérons maintenant deux transducteurs T_1 et T_2 appartenant à deux classes différentes. L'un au moins de ces transducteurs produit une sortie différente de s. Simulons les deux transducteurs sur l'entrée a et comparons leurs sorties à celle de T. Au bout d'un temps fini, nous nous apercevons que l'un des deux transducteurs (disons T_1) produit une sortie qui diffère de s. Nous pouvons par conséquent éliminer T_1 ainsi que la classe à laquelle il appartient. En revanche, nous ne

pouvons rien conclure pour T_2. Choisissons maintenant un transducteur T_3 d'une troisième classe. De nouveau, l'un au moins des deux transducteurs T_2 et T_3 produit une sortie différente de s. En simulant le fonctionnement de ces deux transducteurs et en comparant leur sortie à s, nous pouvons encore une fois éliminer l'un d'eux. Nous procédons ainsi (choix d'un transducteur d'une nouvelle classe, puis élimination du transducteur (et *a fortiori* de la classe à laquelle il appartient) produisant une sortie différente de s) pour toutes les classes. À une certaine étape, nous sommes nécessairement amenés à choisir un transducteur appartenant à la même classe que T. Comme il produit la même sortie que T sur l'entrée a, il n'est jamais éliminé. La dernière classe qui n'a pas été éliminée est donc nécessairement celle de T.

Cet algorithme est toutefois irréaliste en temps de calcul. Des heuristiques seraient à étudier, inférant le transducteur au fur et à mesure de la lecture.

Chapitre 4

Classification des images de séquences aléatoires

Nous nous intéressons maintenant aux images des séquences aléatoires par les transducteurs rationnels lettre à lettre. Le théorème 5.3 montre comment la structure du transducteur détermine le type de séquence obtenue en sortie.

4.1 États partiellement indifférents et états discriminants

Définition 5.7 (état partiellement indifférent) *Un état q est* partiellement indifférent *s'il existe deux mots d'entrée différents de même longueur conduisant à un même état à partir de q en produisant la même sortie.*

Remarquons que si un état q est partiellement indifférent, alors il est clair que tout état $q' < q$ l'est également. Par conséquent, un état d'une classe d'équivalence est partiellement indifférent si et seulement si tous les états de cette classe le sont.

Définition 5.8 (hypothèse de discrimination) *Soit q un état et l un entier positif. Nous appelons hypothèse de discrimination, notée* DISCRIMINATION(q, l)*, l'équivalence suivante : pour toute paire de mots de longueur l, les deux mots commencent par la même lettre si et seulement si leurs images à partir de q se terminent par la même lettre.*

Définition 5.9 (état discriminant) *Un état q est* discriminant *s'il existe $l > 0$ pour lequel l'hypothèse de discrimination* DISCRIMINATION(q, l) *est vraie.*

Intuitivement, un état partiellement indifférent est un état qui *efface* de l'information. Si cet état survient suffisamment souvent lors de la lecture de

a (plus précisément, s'il est fréquent pour a), alors $T(a)$ est compressible. *A contrario*, un état discriminant est un état qui *conserve* l'information. Si cet état survient suffisamment souvent lors de la lecture de a (plus précisément s'il est fréquent pour a), alors $T(a)$ est non récursive. C'est ce que nous établissons, de façon plus détaillée, dans le théorème suivant.

4.2 Classification des images de séquences aléatoires

Théorème 5.3 *Soit T un transducteur rationnel lettre à lettre et soit a une séquence aléatoire acceptée par T.*

1. *si T_a ne contient aucun état partiellement indifférent alors $T(a)$ est aléatoire ;*
2. *(a) si T_a contient au moins un état partiellement indifférent, alors $T(a)$ est non-aléatoire. De plus :*
   (b) si T_a contient au moins un état discriminant, alors $T(a)$ est non-récursive ;
   (c) si T_a ne contient aucun état discriminant alors $T(a)$ est récursive et ultimement périodique : il existe deux mots u et v tels que $T(a) = uv^\infty$.

Preuve de 1. et 2.(a). Soit T un transducteur rationnel lettre à lettre et a une séquence aléatoire acceptée par T. Nous montrons que $T(a)$ est aléatoire si et seulement si T_a ne contient aucun état partiellement indifférent.

Condition nécessaire : Nous montrons que si T_a contient au moins un état q partiellement indifférent, alors $T(a)$ n'est pas aléatoire. Comme q est un état partiellement indifférent, il existe deux mots différents, w_1 et w_2, de même longueur l, conduisant à un même état q' à partir de q, tout en produisant le même mot u en sortie. Soient c_1 et c_2 les chemins d'états partant de q et arrivant à q' correspondant à la lecture de w_1 et w_2. Soit S l'ensemble des chemins de longueur l partant de q.

A priori, deux situations sont envisageables, selon que $f^{a,S}(c_1 \text{ ou } c_2) = 0$ ou que $f^{a,S}(c_1 \text{ ou } c_2) > 0$.

Première situation : $f^{a,S}(c_1 \text{ ou } c_2) = 0$, ce qui ne peut avoir lieu que si $l \geq 2$. Il existe nécessairement un chemin c_3 de S, différent de c_1 et c_2, tel que $f^{a,S}(c_3) > 0$. Soit c_4 un quatrième chemin de S, différent de c_1, c_2 et c_3. Clairement, $f^{a,S}(c_3 \text{ ou } c_4) > 0$. Soient w_3 et w_4 les mots de longueur l dont la lecture permet de suivre les chemins c_3 et c_4. Soit v un mot quelconque de $l-2$ lettres. Considérons une fonction de codage f de $\{0,1\}^l$ dans $\{0,1\}^l$ telle que :

$$\begin{cases} f(w_1) = v00 \text{ et } f(w_2) = v10 \\ f(w_3) = v01 \text{ et } f(w_4) = v11 \end{cases}$$

Les images de w_3 et w_4 diffèrent uniquement par leur avant-dernière lettre. Il en est de même des images de w_1 et de w_2.

Considérons tout d'abord la séquence a', définie à partir de a où tout mot w_2 lu lors d'une occurrence d'un chemin de S est remplacé par w_1. Clairement, $T(a') = T(a)$.

Considérons maintenant la séquence a'', définie à partir de a' où tout mot lu lors d'une occurrence d'un chemin de S est remplacé par un autre mot.

- Si le mot lu est w_1, le mot de remplacement est l'écriture préfixe d'un entier qui est le nombre d'occurrences successives de chemins de $S\backslash\{c_1\}$ qui vont suivre.
- Si le mot lu est w_3 (respectivement w_4), le mot de remplacement est $v0$ (respectivement $v1$). Notons que ce mot de remplacement est de $l-1$ lettres.
- Dans tous les autres cas, le mot de remplacement est l'image par f du mot à remplacer.

Le programme suivant calcule $T(a)$ quand on lui fournit a'' en entrée.

```
Programme P(a'')
    Initialiser n.
    Simuler T sur a''.
    Si une occurrence d'un chemin de S commence Alors
        Si n = 0 Alors
            Réinitialiser n avec l'entier codé de façon
            préfixe à la position courante dans a''.
            Dans a'', remplacer cet entier par w1.
        Sinon
            n = n − 1
            Soit w le mot formé par les l − 1 lettres de a''
            qui suivent.
            Si w = v0 Alors
                Remplacer dans a'' le mot w lu par w3.
            Sinon Si w = v1 Alors
                Remplacer dans a'' le mot w lu par w4.
            Sinon
                Lire encore une lettre b dans a''.
                Remplacer dans a'' le mot wb par f^{-1}(wb).
            Fin Si
        Fin Si
    Fin Si
    Reprendre la simulation de T.
Fin
```

Ce programme utilise un compteur n initialisé avec le nombre d'occurrences successives de chemins de $S \setminus \{c_1\}$ dans a' avant la première occurrence d'un chemin de S qui soit c_1.

Il reconstruit a' à partir de a'' et simule T sur a', ce qui produit $T(a)$. Pour reconstruire a', il simule T sur a''. Quand le transducteur atteint l'état q et que cela débute une nouvelle occurrence d'un chemin de S, cela signifie que l'on a atteint un endroit ou a'' et a' diffèrent. Le programme P vérifie d'abord la valeur de son compteur n. Si celui-ci est à 0, cela signifie qu'on a atteint le début d'une occurrence de c_1 dans a'. Le compteur est alors réinitialisé avec l'entier codé de façon préfixe dans a'', entier qui est remplacé, dans a'' par le mot w_1. La simulation de T peut alors reprendre. Si le compteur n'est pas nul, les $l-1$ lettres suivantes de a'' sont lues. Si elles forment le mot $v0$ (respectivement $v1$) le début d'une occurrence de w_3 (respectivement w_4) a été atteint. Dans a'', le mot $v0$ (respectivement $v1$) est alors remplacé par le mot w_3 (respectivement w_4). La simulation de T peut alors reprendre. Si les $l-1$ lettres lues ne forment ni le mot w_3 ni le mot w_4, une lettre supplémentaire est lue pour former un mot w de l lettres. Ce mot w est alors remplacé dans a'' par $f^{-1}(w)$.

Notons que dans a'', on gagne une lettre à chaque fois qu'il y a une occurrence de c_3 ou de c_4 dans S, ce qui survient avec une fréquence strictement positive. On perd éventuellement quelques lettres à chaque fois qu'une occurrence de c_1 dans S se produit, mais ceci ne survient qu'avec une fréquence nulle.

Par une technique de preuve analogue à celle utilisée dans la preuve du lemme 5.4, nous montrerions que les hypothèses du lemme 2.1 sont satisafites : a n'est pas aléatoire.

Seconde situation : $f^{a,S}(c_1 \text{ ou } c_2) > 0$. Soit f une fonction de codage de $\{0,1\}^l$ dans $\{0,1\}^l$ telle que $f(w_1)$ et $f(w_2)$ diffèrent seulement par leur dernière lettre. Par exemple $f(w_1) = v0$ et $f(w_2) = v1$, où v est un mot de longueur $l-1$. Soit a' la séquence définie à partir de a de la façon suivante : dans la séquence a, nous remplaçons tout mot w de longueur l lu lors d'une occurrence d'un chemin de S par :

- v si $w = w_1$ ou $w = w_2$;
- $f(w)$ sinon.

Le programme suivant, qui utilise le transducteur T, produit la séquence $T(a)$ en sortie quand on lui fournit a' en entrée.

```
Programme P(a')
    Simuler T avec en entrée la séquence a'
    Si une occurrence d'un chemin de S commence Alors
        Soit w le mot de l-1 lettres à partir de la
        position courante dans a'
        Si w = v Alors
            Remplacer dans a' le mot w lu par w1
        Sinon
            Lire encore une lettre b dans a'
            Soit w' le mot tel que f(w') = wb
            Dans a' remplacer le mot wb lu par w'
        Fin Si
    Fin Si
    Reprendre la simulation de T
Fin
```

Ce programme simule T sur l'entrée a'. Sa sortie est celle de T tant que les lettres de a et de a' correspondent. Il détecte les endroits où les deux séquences diffèrent en surveillant l'état atteint par T. Quand T atteint l'état q et que cela correspond au début d'une occurrence d'un chemin de S, cela signifie que les lettres de a et de a' ne correspondent plus. Le programme P lit alors les $l-1$ lettres suivantes de a' et les compare au mot v. S'il y a égalité, cela signifie que les l prochaines lettres de a forment soit le mot w_1 soit le mot w_2. Comme ces deux mots produisent la même sortie et permettent d'atteindre le même état à partir de q, P peut remplacer dans a' le mot v lu par l'un quelconque de ces deux mots. Ici, c'est w_1 qui est choisit. S'il n'y a pas égalité, P lit une encore une lettre afin d'obtenir un mot de l lettre. Le décodage de ce mot par f^{-1} permet de retrouver les l lettres de a. La simulation de T peut alors reprendre là où elle s'était arrêtée.

Comme $f^{a,S}(c_1 \text{ ou } c_2) > 0$, nous avons $\limsup \frac{occ_n^a(c_1 \text{ ou } c_2)}{n} = \alpha$ avec $\alpha > 0$. Donc il existe une suite $(v_n)_{n\in\mathbb{N}}$ et n_0 tels que pour tout $n > n_0$ nous ayons $\frac{occ_{v_n}^a(c_1 \text{ ou } c_2)}{v_n} \geq \frac{\alpha}{2}$. Le nombre u_n de lettres de a' nécessaires au programme P pour calculer les v_n premières lettres de a est majoré par $v_n - occ_{v_n}^a(c_1 \text{ ou } c_2)$. Pour tout $n > n_0$ nous avons $u_n \leq \lambda v_n$ avec $\lambda = 1 - \frac{\alpha}{2} < 1$. Les hypothèses du lemme 2.1 sont satisfaites et par conséquent nous concluons que a n'est pas aléatoire.

Condition suffisante : a étant une séquence aléatoire acceptée par T, nous montrons que si T_a ne contient aucun état partiellement indifférent, alors $T(a)$ est aléatoire. Raisonnons par l'absurde et supposons que $T(a)$ n'est pas aléatoire. Montrons que ceci implique que la séquence a n'est pas aléatoire non plus.

Dans la suite, $u(i,j)$ représente une fonction récursive associant des mots à des couples d'entiers, $\Gamma_{u(i,j)}$ représente l'ensemble des séquences admet-

tant comme préfixe le mot $u(i,j)$, Ω_i représente l'union sur j de tous ces ensembles : $\Omega_i = \bigcup_j \Gamma_{u(i,j)}$. Notons μ la mesure uniforme sur l'ensemble des séquences. Comme $T(a)$ n'est pas aléatoire, $T(a)$ appartient à un ensemble constructivement de mesure nulle (voir définition 2.5). En d'autres termes, il existe une fonction récursive $u(i,j)$ telle que :

1. $\forall\, i\ \mu(\Omega_i) \leq 2^{-i}$
2. $\forall\, i\ s \in \Omega_i$

Remarquons que i étant fixé, nous pouvons supposer que tous les ensembles $\Gamma_{u(i,j)}, j \in \mathbb{N}$, sont disjoints, *i.e.* que l'ensemble $\{u(i,j) \mid j \in \mathbb{N}\}$ est un ensemble préfixe. Ainsi, $\mu(\Omega_i) = \sum_j \mu(\Gamma_{u(i,j)})$, égalité qui nous sera nécessaire dans la suite de la preuve.

Nous allons maintenant définir une fonction $\hat{u}(i,j)$, associant un mot à chaque couple d'entier, telle que :

$$\forall\, i\ \{T(\hat{u}(i,j)) \mid j \in \mathbb{N}\} = \{u(i,j) \mid j \in \mathbb{N}\}.$$

Le programme suivant $P(i,j)$ calcule $\hat{u}(i,j)$ en utilisant la fonction récursive u et le transducteur T. Il énumère l'ensemble des mots ayant comme images par T les mots $u(i,k)$, $k \in \mathbb{N}$, et choisit le $j^{\text{ème}}$ élément.

```
Programme P(i,j)
    compteur = 0
    k = 0
    Répéter
        k = k + 1
        w = u(i,k)
        Pour tous les mots ŵ de longueur |w| Faire
            Si T(ŵ) = w Alors
                compteur = compteur + 1
                Si compteur = j Alors
                    Sortir ŵ
                    Arrêter
                Fin Si
            Fin Si
        Fin Pour
    Fin Répéter
Fin
```

Comme u est une fonction récursive, $\hat{u}$ est également récursive. Soit $\hat{\Omega}_i = \bigcup_j \Gamma_{\hat{u}(i,j)}$. Comme $\hat{u}(i,j)$ est l'antécédent d'un $u(i,k)$, a appartient à $\hat{\Omega}_i$ pour tout i. De plus nous avons :

$$\forall\, i\ \mu(\hat{\Omega}_i) \leq \sum_j \mu(\Gamma_{\hat{u}(i,j)}).$$

Soit Q le nombre d'états de T. Comme T_a ne contient aucun état partiellement indifférent, il y a au plus Q mots différents ayant la même image. Par conséquent nous avons :

$$\begin{aligned} \forall\, i\ \mu(\hat{\Omega}_i) &\leq \sum_j Q\mu(\Gamma_{u(i,j)}) \\ &= Q\mu(\Omega_i) \\ &\leq Q2^{-i} \end{aligned}$$

a appartient donc à l'ensemble constructivement de mesure nulle défini par $\hat{u}(i,j)$, ce qui signifie que a n'est pas aléatoire. □

Preuve du point 2.(b). Soit q un état discriminant. Alors il existe $l > 0$ tel que l'observation de la dernière lettre de l'image d'un mot de longueur l à partir de l'état q permette de retrouver la première lettre de ce mot. Supposons par exemple que tous les mots de longueur l commençant par 0 (respectivement par 1) aient leurs images, calculées à partir de l'état q, se terminant par 0 (respectivement par 1).

Raisonnons par l'absurde et supposons que $T(a)$ soit récursive. Il existe alors un programme $P_{T(a)}$ qui calcule $T(a)$. Définissons une séquence a' de la façon suivante : dans la séquence a, nous effaçons toutes les lettres lues lors de l'occurrence de l'état q. Nous pouvons maintenant écrire un programme calculant a en utilisant le programme $P_{T(a)}$, le transducteur T et la séquence a' en entrée. Dans ce programme, q_0 (respectivement q_1) désigne l'état atteint à partir de q lors de la lecture d'un 0 (respectivement 1) en entrée.

```
Programme P(a')
    Répéter
        Si T est dans l'état q Alors
            Calculer la l^ème lettre suivante de T(a) à
            l'aide P_T(a).
            Si cette lettre est 0 Alors
                Sortir 0
                Placer T dans l'état q0
            Sinon
                Sortir 1
                Placer T dans l'état q1
            Fin Si
        Sinon
            Lire une lettre b de a' et sortir b
            Simuler T sur la lettre b
        Fin Si
    Fin Répéter
Fin
```

Comme q est fréquent pour a nous avons $\limsup_{n\to\infty} \frac{occ_n^a(q)}{n} = \alpha$ avec $\alpha > 0$. Donc il existe une suite $(v_n)_{n\in\mathbb{N}}$ et n_0 tels que pour tout $n > n_0$ nous ayons $\frac{occ_{v_n}^a(q)}{v_n} \geq \frac{\alpha}{2}$. Soit $u_n = v_n - occ_{v_n}^a(q)$. Pour tout $n > n_0$ nous avons $u_n \leq \lambda v_n$ avec $\lambda = 1 - \frac{\alpha}{2} < 1$. Le programme P est tel que pour tout n nous avons $P(a'_{1:u_n}) = a_{1:v_n}$. Les hypothèses du lemme 2.1 sont satisfaites et par conséquent nous concluons que a n'est pas aléatoire. □

Preuve de 2.(c). Nous allons d'abord démontrer la propriété suivante : pour tout $l > 0$, pour tout état q de T_a, pour tous mots w_1 et w_2 de longueur l, les images de w_1 et de w_2 calculées à partir de q sont les mêmes. Nous allons pour cela suivre un raisonnement par récurrence sur la longueur l. Nous désignons par $W(q,l)$ l'ensemble des mots images de longueur l à partir de q.

Étape 1 : Cette propriété est vraie pour $l = 1$. En effet chaque état de T_a étant non discriminant, il produit la même lettre, que l'entrée soit 0 ou 1.

Étape 2 : Supposons maintenant que cette propriété soit vraie pour tout $l \leq L$. Soit q un état de T_a. Nous désignons par q_0 (respectivement par q_1) l'état atteint à partir de q lors de la lecture d'un 0 (respectivement d'un 1) en entrée. L'hypothèse de récurrence nous dit que tous les mots de $W(q_0, L)$ (respectivement de $W(q_1, L)$) se terminent par une même lettre b_0 (respectivement b_1). Comme q n'est pas discriminant, nous avons nécessairement $b_0 = b_1$. En d'autres termes, tous les mots de longueur $L+1$ ont des images calculées à partir de q qui se terminent par la même lettre.

Étape 3 : Nous pouvons maintenant montrer que $T(a)$ est ultimement périodique. Soit q_0 l'état initial de T. Soit q_1 un état récurrent pour a. En lisant la séquence a à partir de q_0, T atteint une première fois q_1 en ayant produit un mot u_1. À partir de q_1, d'après la propriété précédente, T produit la même séquence que s'il lisait une suite infinie de 0 en entrée. Intéressons nous donc à l'image de 0^∞ à partir de q_1. Comme le nombre d'états de T est fini, il existe un état q_2 atteint une infinité de fois. De q_1 à la première occurrence de q_2, T produit un mot u_2. De la première à la seconde occurrence de q_2, il produit un mot u_3 en ayant suivi un chemin C. De la seconde à la troisième occurrence de q_2, comme il est déterministe et comme l'entrée ne comporte que des 0, il emprunte le même chemin C et par conséquent produit à nouveau le mot u_3. En réitérant le raisonnement, il devient clair que l'image de a est la séquence $u_1u_2(u_3)^\infty$ qui est ultimement périodique. □

4.3 Exemples

Exemple 5.1 Qualifions d'injectif un transducteur définissant une fonction injective de l'ensemble des séquences (infinies) dans l'ensemble des séquences (infinies). Un transducteur injectif ne peut comporter aucun état partiellement indifférent, sinon deux séquences différentes auraient une même image.

Par conséquent il transforme les séquences aléatoires en séquences aléatoires. Le transducteur de la figure 4.1 est un exemple non-trivial de transducteur injectif (non-trivial car n'étant pas simplement un morphisme injectif de mots) qui transforme une séquence s en la séquence $0s$.

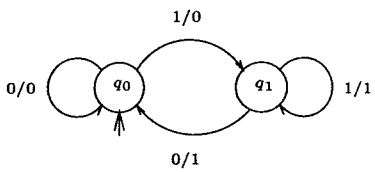


FIG. 4.1: *Exemple de transducteur injectif non-trivial*

Exemple 5.2 Dans l'exemple précédent, nous avons vu que l'injectivité du transducteur est une condition suffisante qui assure que la transformée d'une séquence aléatoire est une séquence aléatoire. Toutefois, ce n'est pas une condition nécessaire. Considérons en effet le transducteur de la figure 4.2. Comme les séquences 0^∞ et 10^∞ ont la même image 01^∞, T n'est pas un transducteur injectif. Cependant, aucun de ses états n'étant accessible par deux transitions produisant la même lettre, il ne comporte aucun état partiellement indifférent.

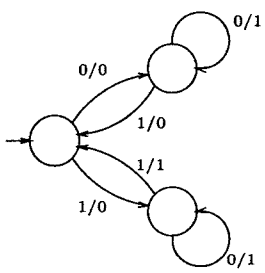


FIG. 4.2: *Un transducteur non-injectif sans aucun état partiellement indifférent*

Interprétation arithmétique Nous donnons maintenant une interprétation du théorème 5.3 en nous plaçant dans le cadre de l'arithmétique. Nous appelons « réel aléatoire » un réel dont le développement binaire est une séquence aléatoire. Rappelons qu'un nombre *algébrique* est un nombre qui est solution d'une équation polynômiale à coefficients entiers et qu'un nombre *transcendant* est un nombre qui n'est pas algébrique.

Corollaire 5.2 *L'image d'un réel aléatoire par un transducteur rationnel lettre à lettre est soit rationnel soit non récursif donc transcendant.*

Preuve. Soit a la séquence image par un transducteur T du développement binaire d'un réel aléatoire. Soit r le réel que représente a. Soit a est périodique, et dans ce cas r est rationnel ; soit a est non récursive, et dans ce cas r est transcendant. En effet, un nombre algébrique est solution d'une équation polynômiale à coefficients entiers et est donc récursif : il est possible, par programme, d'avoir des approximations aussi précises que l'on veut de ce nombre. □

4.4 Existence d'un état partiellement indifférent

Nous avons établi dans le théorème 5.3 que, lorsque l'entrée est une séquence aléatoire, la présence d'un état partiellement indifférent dans la classe absorbante complète atteinte détermine si la séquence de sortie est aléatoire. Nous nous intéressons ici au temps de calcul nécessaire à la détection de ce type d'état. Nous montrons que ce temps de calcul est polynômial en le nombre Q d'états de la classe absorbante complète. Plus précisément, il est de l'ordre de Q^2.

Théorème 5.4 *Soit Q le nombre d'états d'une classe absorbante complète T d'un transducteur. L'existence d'un état partiellement indifférent dans cette classe est décidable en un temps $\mathcal{O}(Q^2)$.*

4.4.1 Algorithme de décision

Le programme ETATINDIFFÉRENT(T, q_0) que nous allons décrire décide si q_0 est un état partiellement indifférent dans la classe absorbante complète T en un temps $\mathcal{O}(Q^2)$, Q étant le nombre d'états de la classe T.

Afin de prendre sa décision, ce programme construit un graphe orienté $\mathcal{G}$, dont les sommets sont soit des singletons soit des couples d'états de T. Les arcs sont construits en respectant trois règles. Avant de donner explicitement ces règles, nous introduisons quelques notations.

G et G' étant deux sommets du graphe construit, nous notons $G \xrightarrow{n} G'$ s'il existe dans ce graphe un chemin de longueur n reliant G à G'. Dans le cas où $n = 1$, nous disons que G est un prédécesseur de G' et G' un successeur de G.

Les trois règles de construction des arcs dans $\mathcal{G}$ sont les suivantes.

1. Pour tous états q, q', $q \xrightarrow{1} q'$ si et seulement si q' vérifie l'hypothèse $\mathcal{H}_1(q)$, c'est-à-dire s'il existe dans T une transition menant de q à q'.

2. Pour tous états q, q_1, q_2, $q \xrightarrow{1} (q_1, q_2)$ si et seulement si (q_1, q_2) vérifie l'hypothèse $\mathcal{H}_2(q)$, c'est-à-dire s'il existe dans T deux transitions différentes menant de q à q_1 et de q à q_2 et produisant la même sortie.

3. Pour tous états q_1, q_2, q'_1, q'_2, $(q_1, q_2) \xrightarrow{1} (q'_1, q'_2)$ si et seulement si (q'_1, q'_2) vérifie l'hypothèse $\mathcal{H}_3(q_1, q_2)$, c'est-à-dire si, dans T, une transition mène de q_1 à q'_1, une transition mène de q_2 à q'_2, la sortie produite étant la même si les deux transitions sont différentes.

Le graphe étant construit, le programme vérifie s'il existe un sommet (q, q) accessible à partir de q_0. Si c'est le cas, il décide que q_0 est partiellement indifférent. Dans le cas contraire, il décide que q_0 n'est pas partiellement indifférent.

```
Programme EtatIndifférent(T, q0)
    Pour tout sommet q Faire
        Pour tout sommet q' vérifiant H1(q) Faire
            Relier(q, q')
        Fin Pour
        Si il existe (q1, q2) vérifiant H2(q) Alors
            Relier(q, (q1, q2))
        Fin Si
    Fin Pour
    Pour tout sommet (q1, q2) Faire
        Pour tout sommet (q'1, q'2) vérifiant H3(q1, q2) Faire
            Relier((q1, q2), (q'1, q'2))
        Fin Pour
    Fin Pour
    Si un sommet (q, q) est accessible à partir de q0 Alors
        Sortir « q0 est partiellement indifférent »
    Sinon
        Sortir « q0 n'est pas partiellement indifférent »
    Fin Si
Fin
```

La procédure Relier(G_1, G_2) construit simplement un arc de G_1 à G_2. Nous allons montrer maintenant que le programme EtatIndifférent permet de décider de l'existence d'un état partiellement indifférent dans une classe T absorbante et complète en un temps quadratique en le nombre d'états de cette classe.

Un état de T est partiellement indifférent si et seulement si tous les états de T le sont. Pour décider de l'existence d'un état partiellement indifférent dans T, il suffit donc de décider si l'un quelconque des états de T possède cette propriété. Soit q_0 un état quelconque de T. Nous allons montrer que le programme EtatIndifférent décide si cet état est partiellement indifférent ou non en un temps quadratique en le nombre d'états de T. Soit Q le nombre d'états de T. Nous allons prouver successivement que :

1. le programme s'arrête en un temps $\mathcal{O}(Q^2)$;

2. le programme s'arrête et répond « q_0 est partiellement indifférent » si et seulement si q_0 est partiellement indifférent ;

Montrons d'abord que le programme s'arrête en un temps $\mathcal{O}(Q^2)$.

Preuve du point 1. Le temps de construction du graphe est proportionnel au nombre d'arêtes construites. Comme d'un couple (q_1, q_2) partent au plus 6 arêtes (nombre de choix de deux états parmi quatre) et d'un singleton au plus 3, le nombre d'arêtes construites est majoré par $6Q^2 + 3Q$. En outre, vérifier si le graphe $\mathcal{G}$ comporte un sommet (q, q) accessible à partir de q_0 se fait en un temps $\mathcal{O}(Q^2)$. □

Afin de montrer le second point, nous allons établir le lemme suivant.

Lemme 5.9 *Pour tout entier n, le graphe construit par le programme* ETATINDIFFÉRENT *vérifie les deux propriétés suivantes.*

$\text{Prop}_1(n)$ *: pour tout état q, $q_0 \xrightarrow{n} q$ si et seulement s'il existe dans T un chemin de longueur n menant de q_0 à q.*

$\text{Prop}_2(n)$ *: pour tous états q_1, q_2, q_1', q_2', $(q_1, q_2) \xrightarrow{n} (q_1', q_2')$ si et seulement s'il existe dans T un chemin de longueur n menant de q_1 à q_1', un chemin de longueur n menant de q_2 à q_2', les sorties étant identiques si les chemins sont différents.*

Preuve. Nous faisons un raisonnement par récurrence sur n.

Étape 1 : Par définition de $\mathcal{H}_1$, la propriété Prop_1 est vraie pour $n = 1$; par définition de $\mathcal{H}_2$, la propriété Prop_2 est vraie pour $n = 1$.

Étape 2 : Supposons que les deux propriétés soient vraies jusqu'au rang n. Montrons qu'elles sont vraies au rang $n + 1$.

Étape 2.1 : Commençons par examiner le cas de la propriété Prop_1.

Condition nécessaire : Soit q un état de T tel que $q_0 \xrightarrow{n+1} q$. Nécessairement, il existe un état q' tel que $q_0 \xrightarrow{n} q'$ et $q' \xrightarrow{1} q$, ceci signifiant que q vérifie l'hypothèse $\mathcal{H}_1(q')$, c'est-à-dire qu'il existe dans T une transition menant de q' à q. L'hypothèse de récurrence nous permet d'affirmer qu'il existe dans T un chemin de longueur n menant de q_0 à q'. Par conséquent, il existe dans T un chemin de longueur $n + 1$ menant de q_0 à q.

Condition suffisante : Soit q un état tel qu'il existe dans T un chemin de longueur $n+1$ menant de q_0 à q. Soit q' le prédécesseur de q dans ce chemin. q vérifie l'hypothèse $\mathcal{H}_1(q')$, donc $q' \xrightarrow{1} q$. L'hypothèse de récurrence nous permet d'affirmer que $q_0 \xrightarrow{n} q'$. Donc finalement $q_0 \xrightarrow{n+1} q$.

Étape 2.2 : Examinons maintenant le cas de la seconde propriété, Prop_2.

Condition nécessaire : Soient q_1, q_2, q'_1, q'_2 des états de T tels que $(q_1, q_2) \xrightarrow{n+1} (q'_1, q'_2)$. Il existe donc dans $\mathcal{G}$ un chemin de longueur $n+1$ menant de (q_1, q_2) à (q'_1, q'_2). Soit (q''_1, q''_2) le prédécesseur de (q'_1, q'_2) dans ce chemin. Nous avons alors $(q_1, q_2) \xrightarrow{n} (q''_1, q''_2)$ et $(q''_1, q''_2) \xrightarrow{1} q'_1, q'_2)$. Par hypothèse de récurrence, cela signifie que :

1. il existe dans T un chemin de longueur n menant de q_1 à q''_1, un chemin de longueur n menant de q_2 à q''_2, les sorties étant identiques si les chemins sont différents ;
2. il existe dans T une transition menant de q''_1 à q'_1, une transition menant de q''_2 à q'_2, les sorties étant identiques si les transitions sont différentes.

Par conséquent, il existe dans T un chemin de longueur $n+1$ menant de q_1 à q'_1, un chemin de longueur $n+1$ menant de q_2 à q'_2, les sorties étant identiques si les chemins sont différents.

Condition suffisante : Soient q_1, q_2, q'_1, q'_2 des états de T. Supposons qu'il existe dans T un chemin C_1 de longueur $n+1$ menant de q_1 à q'_1, un chemin C_2 de longueur $n+1$ menant de q_2 à q'_2, les sorties étant identiques si les chemins sont différents. Soit q''_1 le prédécesseur de q'_1 dans le chemin C_1 et soit q''_2 le prédécesseur de q'_2 dans le chemin C_2. Nous pouvons alors faire les affirmations suivantes.

1. Il existe dans T un chemin de longueur n menant de q_1 à q''_1, un chemin de longueur n menant de q_2 à q''_2, les sorties étant identiques si les chemins sont différents. Par hypothèse de récurrence, cela implique que $(q_1, q_2) \xrightarrow{n} (q''_1, q''_2)$.
2. Il existe dans T une transition menant de q''_1 à q'_1, une transition menant de q''_2 à q'_2, les sorties étant identiques si les transitions sont différentes. Par hypothèse de récurrence, cela implique que $(q''_1, q''_2) \xrightarrow{1} (q'_1, q'_2)$.

Finalement nous avons $(q_1, q_2) \xrightarrow{n+1} (q'_1, q'_2)$.

□

Nous pouvons maintenant terminer la preuve du théorème 5.4

Preuve du point 2. Le programme ETATINDIFFÉRENT décide que q_0 est partiellement indifférent si et seulement s'il existe un état q de T et $n > 0$ tels que dans le graphe $\mathcal{G}$ nous ayons $q_0 \xrightarrow{n} (q, q)$, *i.e.* si et seulement s'il existe des états q', q_1, q_2, q, et des entiers m et m' tels que :

$$\begin{cases} q_0 \xrightarrow{m} q' \\ q' \xrightarrow{m'} (q_1, q_2) \\ (q'_1, q'_2) \xrightarrow{1} (q, q). \end{cases}$$

D'après le lemme 5.9 et la définition de l'hypothèse $\mathcal{H}_2$, cela équivaut à l'existence d'états q', q_1, q_2, q, et d'entiers m et m' tels que, dans T :

1. il existe un chemin de longueur m menant de q_0 à q' ;
2. il existe deux transition différentes menant de q' à q_1 et de q' à q_2 et produisant les mêmes sorties ;
3. il existe un chemin de longueur m' menant de q_1 à q, un chemin de longueur m' menant de q_2 à q, les deux sorties étant identiques si les chemins sont différents.

Cela équivaut à l'existence de deux chemins différents menant de q_0 à q et produisant la même sortie, *i.e.* au fait que q_0 soit partiellement indifférent. □

4.4.2 Exemples

Exemple 5.3 Reprenons l'exemple du transducteur de la figure 4.1 : il transforme toute séquence s en une séquence $0s$, par conséquent il définit une injection sur l'ensemble des séquences infinies, donc ne peut pas comporter d'état partiellement indifférent. Le graphe $\mathcal{G}$ construit par le programme ETATINDIFFÉRENT(T, q_0) est celui de la figure 4.3.

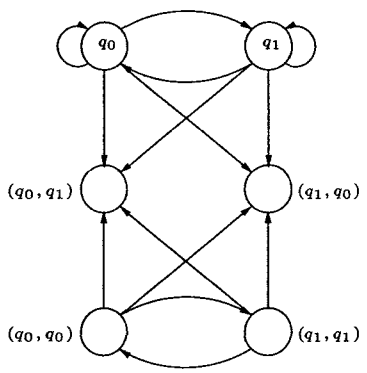


FIG. 4.3: *Graphe construit par le programme* ETATINDIFFÉRENT

Dans ce graphe, ni (q_0, q_0) ni (q_1, q_1) ne sont accessibles à partir de q_0. Par conséquent, le programme conclut que q_0 n'est pas partiellement indifférent, donc que T ne contient aucun état partiellement indifférent, ce qui est bien le cas.

Exemple 5.4 Considérons maintenant le transducteur de la figure 4.4. Dans ce transducteur, q_0 est partiellement indifférent. En effet, à partir de q_0, les entrées 00 et 11 permettent d'atteindre l'état q_1 en produisant toutes deux la même sortie 01.

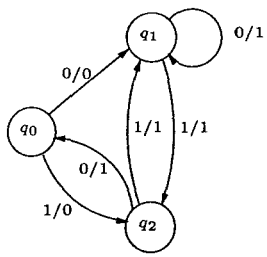


FIG. 4.4: *Un transducteur possédant un état partiellement indifférent*

Le graphe construit par le programme ETATINDIFFÉRENT(T, q_0) est celui de la figure 4.5.

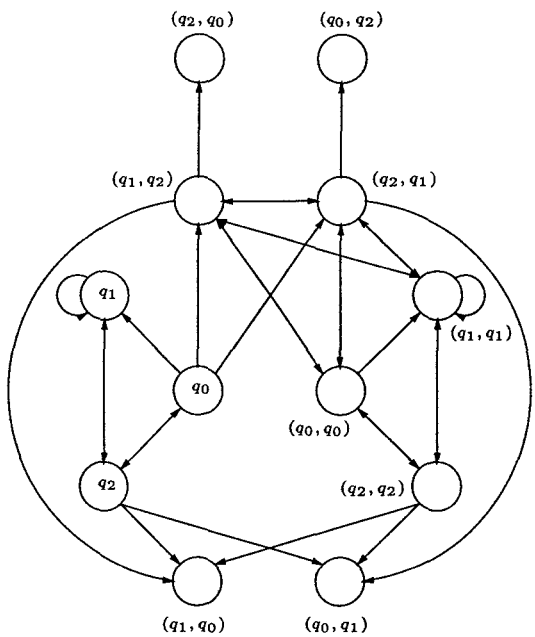


FIG. 4.5: *Graphe construit par le programme* ETATINDIFFÉRENT

Dans ce graphe, (q_1, q_1) est accessible à partir de q_0. L'algorithme s'arrête donc en concluant que q_0 est partiellement indifférent, ce qui est effectivement le cas.

4.5 Existence d'un état discriminant

Nous avons établi, dans le théorème 5.3, que, lorsque l'entrée est une séquence aléatoire, la présence d'un état discriminant dans la classe absorbante complète atteinte détermine si la séquence de sortie est récursive. Nous nous

intéressons ici au temps de calcul nécessaire à la détection de ce type d'état. Nous montrons que ce temps de calcul est polynômial en le nombre Q d'états de la classe absorbante complète. Plus précisément, il est de l'ordre de Q^9. L'idée de la preuve revient au Professeur G. Senizergues que nous remercions ici.

Théorème 5.5 *Soit Q le nombre d'états d'une classe absorbante complète T d'un transducteur. L'existence d'un état discriminant dans cette classe est décidable en un temps $\mathcal{O}(Q^9)$.*

Preuve. Soit q un état de T. Pour tout $n > 0$, nous désignons par $\mathcal{Q}_0(n)$ (respectivement par $\mathcal{Q}_1(n)$) l'ensemble des états atteints à partir de q par des mots d'entrée de longueur n et dont la première lettre est 0 (respectivement 1).

La fonction ÉTATDISCRIMINANT(T, q) retourne VRAI ou FAUX selon que l'état q est ou n'est pas discriminant, et ceci en un temps $\mathcal{O}(Q^9)$. Elle construit, pour des valeurs de n successives à partir de 1, les couples d'ensembles d'états $(\mathcal{Q}_0(n), \mathcal{Q}_1(n))$. Si un couple a déjà été construit précédemment, la fonction s'arrête en décidant que l'état q n'est pas discriminant. Dans le cas contraire, elle vérifie si l'hypothèse de discrimination DISCRIMINATION(q, n) est vraie. Si c'est le cas, elle s'arrête en décidant que q est un état discriminant, sinon elle passe au couple suivant.

Les couples d'ensembles d'états sont représentés par des couples de listes ordonnées d'états. La fonction garde en mémoire une liste ordonnée $\mathcal{L}$ des couples déjà construits.

```
Fonction ÉTATDISCRIMINANT(T, q)
    Si les deux transitions sortant de q ont des sorties
    différentes Alors
        Retourner VRAI
    Sinon
        n = 1 ; L = ∅
        Répéter
            Construire (Q_0(n), Q_1(n))
            Si ce couple est déjà présent dans L Alors
                Retourner FAUX
            Sinon
                Si tous les états de Q_0(n) produisent une
                même lettre Et tous les états de Q_1(n)
                produisent une même lettre Alors
                    Si ces deux lettres diffèrent Alors
                        Retourner VRAI
                    Sinon
                        Insérer (Q_0(n), Q_1(n)) dans L
                    Fin Si
                Sinon
                    Insérer (Q_0(n), Q_1(n)) dans L
                Fin Si
                n = n + 1
            Fin Si
        Fin Répéter
    Fin Si
Fin
```

Il est clair que si cette fonction décide en un temps $\mathcal{O}(Q^8)$ si q est un état discriminant alors en l'appliquant successivement à chacun des états de T on décide en un temps $\mathcal{O}(Q^9)$ de l'existence d'un état discriminant dans T.

Montrons maintenant que la fonction ÉTATDISCRIMINANT(T, q)

1. s'arrête en un temps $\mathcal{O}(Q^8)$;
2. est telle que si elle renvoie VRAI, alors l'état q est discriminant ;
3. est telle que si q est discriminant, alors elle renvoie VRAI.

Preuve de 1 : Le nombre d'états de T étant Q, il existe nécessairement un chemin de longueur $\mathcal{O}(Q)$ entre deux états de T. Par conséquent il existe un circuit fermé C de longueur $N = \mathcal{O}(Q^2)$ passant au moins une fois par tous les états de T.

Pour tout $k \geq 0$, notons $\hat{\mathcal{Q}}_0(k) = \mathcal{Q}_0(kN+1)$. $\hat{\mathcal{Q}}_0(k+1)$ est donc l'ensemble des états accessibles par des chemins de longueur N à partir des états de $\hat{\mathcal{Q}}_0(k)$. Comme T est absorbante, $\hat{\mathcal{Q}}_0(k) \subseteq T$. L'existence du circuit fermé C implique que tout état de $\hat{\mathcal{Q}}_0(k)$ se retrouve dans $\hat{\mathcal{Q}}_0(k+1)$. Par conséquent,

$$\hat{\mathcal{Q}}_0(k) \subseteq \hat{\mathcal{Q}}_0(k+1).$$

Comme le nombre d'états est fini, la suite $\hat{\mathcal{Q}}_0(k)$ est stationnaire à partir d'un certain rang k_0. De plus la stationnarité a lieu dès que deux ensembles successifs sont égaux. Par conséquent $k_0 \leq Q$.

Cela entraîne que pour $n \geq k_0 N+1$ la suite $\mathcal{Q}_0(n)$ est périodique et que :

$$\forall\, \alpha \geq 0\ \forall\, \beta < N\ \mathcal{Q}_0((k_0+\alpha)N+\beta) = \mathcal{Q}_0(k_0 N+\beta)$$

De la même façon, on démontre qu'il existe $k_1 \leq Q$ tel que :

$$\forall\, \alpha \geq 0\ \forall\, \beta < N\ \mathcal{Q}_1((k_1+\alpha)N+\beta) = \mathcal{Q}_1(k_1 N+\beta)$$

Soit $k = \max\{k_0, k_1\}$. À partir de $kN+1$ la suite de couples d'ensembles d'états $(\mathcal{Q}_0(n), \mathcal{Q}_1(n))$ est périodique et l'on a :

$$\begin{aligned}&\forall\, \alpha \geq 0\ \forall\, \beta < N,\\ &(\mathcal{Q}_0((k+\alpha)N+\beta), \mathcal{Q}_1((k+\alpha)N+\beta)) = (\mathcal{Q}_0(kN+\beta), \mathcal{Q}_1(kN+\beta))\end{aligned}$$

Par conséquent il existe deux rangs $n_1 < n_2 < (k+1)N+1$ pour lesquels les couples d'ensembles d'états $(\mathcal{Q}_0(n_1), \mathcal{Q}_1(n_1))$ et $(\mathcal{Q}_0(n_2), \mathcal{Q}_1(n_2))$ sont identiques. Comme la fonction s'arrête dès qu'un couple apparaît une seconde fois, on est donc sûr que l'arrêt a lieu avant la construction de $(k+1)N+1 = \mathcal{O}(Q^3)$ couples.

Nous évaluons maintenant le nombre d'opérations effectuées à chaque tour de boucle.

Lors de la construction d'un nouveau couple de listes ordonnées d'états, l'insertion d'un état (dans une liste de $\mathcal{O}(Q)$ éléments) requiert $\mathcal{O}(Q)$ opérations. Comme le nombre d'insertions à effectuer est en $\mathcal{O}(Q)$, la construction d'un nouveau couple se fait en $\mathcal{O}(Q^2)$ opérations.

Ce nouveau couple doit ensuite être insérer dans la liste des couples déjà construits. Comme cette liste comporte $\mathcal{O}(Q^3)$ éléments, cela nécessite $\mathcal{O}(Q^3)$ opérations de comparaisons. Un élément de la liste étant un couple de listes ordonnées d'états, une opération de comparaison requiert $\mathcal{O}(Q^2)$ opérations élémentaires de comparaisons d'états. L'insertion d'un nouveau couple se fait par conséquent en $\mathcal{O}(Q^5)$ opérations.

Lors de chaque tour de boucle, le nombres d'opérations élémentaires effectuées est donc en $\mathcal{O}(Q^5)$ opérations. Comme le nombre de tours est en $\mathcal{O}(Q^3)$ (nombres de couples construits), la fonction s'arrête en un temps $\mathcal{O}(Q^8)$.

Preuve de 2 : Il est clair que si la fonction décide que l'état q est discriminant celui-ci l'est effectivement.

Preuve de 3 : Supposons que l'état q soit discriminant. Soit ν le plus petit entier pour lequel l'hypothèse DISCRIMINATION(q, ν) est vraie. Raisonnons par l'absurde et supposons que la fonction s'arrête en décidant que q n'est pas discriminant. La fonction a donc trouvé deux couples d'ensembles d'états $(\mathcal{Q}_0(n_1), \mathcal{Q}_1(n_1))$ et $(\mathcal{Q}_0(n_2), \mathcal{Q}_1(n_2))$ identiques avec $n_1 < n_2 < \nu$ (sinon la fonction se serait arrêtée avant en décidant que l'état q est discriminant). Tous les couples $(\mathcal{Q}_0(n), \mathcal{Q}_1(n))$ avec $n > n_2$ sont égaux à des couples déjà construits. Comme en particulier $\nu > n_2$ il existe $\nu' \leq n_2$ tel que $(\mathcal{Q}_0(\nu'), \mathcal{Q}_1(\nu')) = (\mathcal{Q}_0(\nu), \mathcal{Q}_1(\nu))$. Mais alors l'hypothèse DISCRIMINATION(q, ν') est vraie, ce qui est en contradiction avec le fait que ν soit le plus petit entier vérifiant cette hypothèse. □

Bibliographie

[Cal94] C. Calude. *Information and Randomness, an Algorithmic Perspective.* Springer-Verlag, 1994.

[Cha66] G.J. Chaitin. On the length of programs for computing finite binary sequences. *J. ACM*, 13:547–569, 1966.

[Del94] J.-P. Delahaye. *Information, complexité et hasard.* Hermes, 1994.

[DP98] B. Durand and S. Porrot. Comparison between the complexity of a function and the complexity of its graph. In *MFCS'98*, volume to appear of *Lecture Notes in Computer Science.* Springer-Verlag, August 1998.

[Dub98] J.-C. Dubacq. Introduction à la théorie algorithmique de l'information. Technical Report 05, Laboratoire de l'Informatique du Parallélisme, 1998.

[HRSV97] D. Hammer, A. Romashchenko, A. Shen, and N. Vereshchagin. Inequalities for Shannon entropy and Kolmogorov complexity. In *Twelfth Annual IEEE Conference on Computational Conference*, pages 13–23, June 1997.

[HS] D. Hammer and A. Shen. A strange application of Kolmogorov complexity. *Mathematical Systems Theory*, to appear.

[Kol65] A.N. Kolmogorov. Three approchees for defining the concept of information quantity. *Information transmission*, 1:3–11, 1965.

[Lev73] L.A. Levin. On the notion of random sequence. *Soviet Math. Dokl.*, 14:1413–1416, 1973.

[Lov69] D. W. Loveland. A variant of the Kolmogorov concept of complexity. *Information and Control*, 15:510–526, 1969.

[LV97] M. Li and P. Vitányi. *An Introduction to Kolmogorov Complexity and its Applications.* Springer-Verlag, 1997.

[ML66] P. Martin-Löf. The definition of random sequences. *Inform. Contrib.*, 9:602–619, 1966.

[MT96] S.P. Meyn and R.L. Tweedie. *Markov Chains and Stochastic Stability.* Springer-Verlag, 1996.

[PDDV98] S. Porrot, M. Dauchet, B. Durand, and N. Vereshchagin. Deterministic rational transducers and random sequences. In *FOSSACS'98*, volume 1378 of *Lecture Notes in Computer Science*, pages 258–272. Springer-Verlag, March 1998.

[Ris78] J.J. Rissanen. Modeling by the shortest data description. *Automatica*, 14:465–471, 1978.

[Sch73] C.P. Schnorr. Process complexity and effective random tests. *J. Comput. System Sci.*, 7:376–388, 1973.

[Sha48] C.E. Shannon. The mathematical theory of communication. *Bell System Tech. J.*, pages 379–423, 623–656, 1948.

[Sol64] R.J. Solomonoff. A formal theory of inductive inference, part 1 and part 2. *Inform. and Control*, 7:1–22 and 224–254, 1964.

[US96] V. A. Uspensky and A. Shen. Relations between varieties of Kolmogorov complexities. *Math. Syst. Theory*, 29(3):270–291, 1996.

[USS90] V. A. Uspensky, A. L. Semenov, and A. Shen. Can an individual sequence of zeros and ones be random? *Russian Math. Surveys*, 45(1):121–189, 1990.

[ZL70] A.K. Zvonkin and L.A. Levin. The complexity of finite objects and the development of the concepts of information and randomness by means of theory of algorithms. *Russian Math. Surveys*, 25(6):83–124, 1970.

[Zur89] W.H. Zurek. Algorithmic randomness and physical entropy. *Physical Review, Ser. A*, 40(8):4731–4751, 1989.