Mesures de complexité pour les fonctions: circuits, différentes complexités de Kolmogorov

Adrien Nolhier, encadré par Olivier Bournez

19 août 2025

Sommaire

Table des matières

Introduction

La **machine de Turing** est le modèle de calcul canonique pour parler de complexité. Dans ce modèle, deux complexités algorithmiques sont très naturelles : la complexité **en temps** (nombre d'opérations élémentaires nécessaires pour aboutir au résultat, en fonction de la taille de l'entrée), et la complexité **en espace** (nombre de bits mémoire utilisées pour effectuer un calcul, à partir d'une entrée donnée). Ainsi, cette machine permet de caractériser naturellement de nombreuses classes de complexités : LOG, P, NP, EXP... Mais, pour caractériser d'autres classes, l'utilisation d'autres modèles peut être incontournable. Par exemple, le modèle des **circuits booléens** permet de définir des classes très fines (incluses dans LOG) telles que AC et NC, pas vraiment caractérisables en termes de machine de Turing. Pourtant, ce sont tout autant des classes naturelles, utilisées dans d'autres domaines de l'informatique (par exemple en informatique quantique). Néanmoins, un autre cas de figure est possible : une classe peut s'exprimer dans plusieurs modèles de calcul. Par exemple, la classe P / poly possède une caractérisation naturelle en termes de machines de Turing (avec conseil) ET en termes de circuits booléens. Nous reparlerons plus loin de cette classe particulièrement intéressante. Caractériser une classe de différentes manières (comme à l'aide de différents modèles) est utile car cela permet d'approfondir la connaissance de cette classe, et au final de diversifier les outils permettant d'aborder un problème.

La **complexité de Kolmogorov** (notée C) est une autre mesure de complexité, plus sophistiquée, apparaissant dans de nombreux domaines étudiant l'aléatoire, tels que la théorie de l'information ou la calculabilité. Elle est usuellement définie à l'aide de machines de Turing. Mais, en 2005, paraît un papier très riche intitulé "Power from Random Strings" ([1]) dû à cinq auteurs dont Eric Allender et établissant entre autres, après un certain travail, un **lien direct entre la complexité de Kolmogorov et la complexité en circuits booléens**. Ce travail impose de poser des définitions ingénieuses, ce qui constitue une part importante dans la réussite du papier d'Allender, nous en parlerons dans les sections 2 et 3 de ce document. La **complexité en circuits** (notée Size) est une autre mesure présentant des différences fondamentales avec la complexité de Kolmogorov en termes de calculabilité, c'est pourquoi ce résultat est surprenant. Nous l'étudierons en sections 1 et 2. Ce lien établi entre C et Size fait écho à deux autres liens étudiés dans d'autres domaines de recherche (complexité des preuves), le lien entre C ou Size avec la complexité en "taille de preuve".

Enfin, le modèle du réseau de neurones de type "récurrent" possède, lorsqu'on autorise tous les poids réels, la même puissance calculatoire que la machine de Turing ou qu'une famille de circuits. Mais, en faisant varier l'ensemble de poids autorisés (en prenant un sous-ensemble particulier de $\mathbb{R}$) et en restreignant le "temps d'exécution" (le nombre de couches appliquées aux bits d'entrée), nous pouvons caractériser des classes de complexité intéressantes. En 1997, Balcázar, Gavaldà et Siegelmann ont publié un papier, "Computational Power of Neural Networks : A Characterization in Terms of Kolmogorov Complexity"([2]), décrivant des classes à l'aide de tels réseaux de neurones prenant leurs poids dans des ensembles de réels d'une certaine complexité de Kolmogorov : par exemple, la classe P / poly. Puisque P / poly est usuellement définie à l'aide de circuits, on peut se demander si les définitions de Balcázar ne cachent pas des circuits. Or, c'est l'intuition suscitée par la lecture du papier d'Allender, qui établit huit ans plus tard un lien entre C et Size. Ainsi, en partant des résultats d'Allender, nous avons réussi à les adapter pour étendre ceux de Balcázar.

Contributions

La première partie de mon travail fut de comprendre l'article [1] d'Allender, notamment sa généralisation des circuits booléens en relativisant les définitions aux oracles. Ainsi, il est proposé dans ce rapport une adaptation des preuves des théorèmes de simulation entre machine de Turing et famille de circuits du livre de Sylvain Perifel ([3]), pour le cas où les machines et les circuits ont accès à des oracles, qui ne figuraient pas dans [1]. Les bornes trouvées ne sont pas exactement les mêmes, mais cela n'a pas d'incidence sur la suite. Puis, nous avons proposé une preuve plus détaillée de l'égalité "Size = KT" ainsi que l'égalité "KT = C" permettant de comprendre le résultat "Size = C". Enfin, nous avons avancé des arguments permettant de généraliser ce résultat.

La deuxième partie fut de concevoir avec Oliver Bournez (mon encadrant) et Rui Li (doctorant de l'équipe AlCo) des définitions telles que K' permettant d'utiliser les résultats d'Allender pour prolonger ceux de Balcázar. Ainsi, nous vérifions que la preuve 6.2 (4.9 dans ce document) fonctionne toujours pour cette nouvelle mesure K'. En réutilisant le théorème de hiérarchie infinie de Balcázar (7.2, [2]) avec cette nouvelle mesure (mettant en jeu les circuits booléens) nous avons pu exprimer les classes de la forme $\mathrm{P} / \mathrm{poly}^k$ à l'aide de réseaux de neurones avec des poids d'une certaine complexité en circuit.

Notations

Ensembles

- Le symbole n désigne un entier naturel.
- $[n] = \{0, \dots, n-1\}$.
- L'ensemble $\{0,1\}$ sera parfois noté 2.
- Pour $a, b \in \mathbb{R}$, $[a,b] = \{x \in \mathbb{R} : a \leq x \leq b\}$ est l'ensemble des réels compris entre a et b.

Suites

- 2^n est l'ensemble des chaînes binaires de longueur n.
- $2^{<\mathbb{N}} = \bigcup_{n\in\mathbb{N}} 2^n$ est l'ensemble des chaînes binaires finies.
- $2^{\mathbb{N}} = 2^{\omega}$ est l'ensemble des chaînes binaires infinies.
- $|x|$ désigne la taille de la chaîne binaire finie x.

Fonctions

- Pour une fonction partielle $f : X \subseteq Y$:

  $$\mathrm{dom}(f) = \{x \in X : \exists y \in Y, f(x) = y\}, \quad \mathrm{codom}(f) = f(\mathrm{dom}(f)) = \{y \in Y : \exists x \in X, f(x) = y\}.$$

- On dit que g domine f, noté $f \geq g$, si $\mathrm{dom}(f) = \mathrm{dom}(g)$ et $\forall x \in \mathrm{dom}(f), f(x) \geq g(x)$.
- Une *notation* pour un ensemble dénombrable X est une fonction surjective calculable $2^{<\mathbb{N}} \to X$.
- Une *représentation* est l'analogue pour un ensemble non-dénombrable (typiquement lorsque $|X| = \mathbb{R}$).
- $\delta_{\mathbb{N}} : 2^{<\mathbb{N}} \to \mathbb{N}$ associe à chaque entier son écriture binaire.
- Les fonctions `bin_of_int` et `int_of_bin` sont des fonctions calculables établissant une bijection entre les entiers et les chaînes binaires finies.
- $\delta_{\mathbb{R}} : 2^{\mathbb{N}} \to [0,1]$ est une représentation des chaînes binaires vers les réels.
- On appelle *time-out* une fonction de borne temporelle, généralement notée T et supposée calculable.
- $\delta_2 : 2^{\mathbb{N}} \to [0,1]$ est une bijection calculable entre les chaînes binaires et les réels.
- poly est l'ensemble des fonctions bornées par une fonction polynomiale (y compris certaines fonctions incalculables croissant lentement).

Calculabilité

- Une fonction calculable est supposée totale ($f : X \to Y$), sinon on précisera qu'elle est partielle ($f :\subseteq X \to Y$).
- Φ_e désigne la fonction partielle calculée par la machine de code $e \in \mathbb{N}$.
  - Si la machine s'arrête sur l'entrée $x \in \mathbb{N}$ (noté $\Phi_e(x)\downarrow$), alors la fonction est définie en x et vaut l'entier représenté sur la bande de sortie.
  - Sinon, la machine ne s'arrête pas ($\Phi_e(x)\uparrow$) et la fonction n'est pas définie en x.
- On note $\Phi_e^A(x)[t]\downarrow$ si la machine de code e, avec l'oracle A, s'arrête en t étapes de calcul.

1 Préliminaires — Machine de Turing et complexité en circuit

1.1 Modèle de la Machine de Turing

On suppose que notre lecteur a certaines connaissances sur le modèle des machines de Turing. Nous avons besoin de définir en plus, des machines de Turing avec oracle.

> **Définition 1.1 (Machine de Turing avec oracle).** *Soit $A \subseteq \mathbb{N}$ un ensemble fini ou infini appelé oracle. Intuitivement, une machine avec un oracle pour A est une machine de Turing au sens classique, mais à laquelle on ajoute la possibilité,* ***au cours de son exécution, de "questionner l'oracle"****, en posant des questions du type "x appartient-il à A ?".* ∘

Par rapport à une machine classique, une version avec oracle peut présenter :

— Un gain en **temps de calcul** : la machine s'exécute avec une meilleure complexité temporelle asymptotique.

— Un gain en **puissance calculatoire** : la machine permet de calculer une fonction normalement "incalculable" (pour laquelle il n'existe pas d'algorithme).

Pour plus de détails sur le formalisme de la machine avec oracle, voir la preuve de simulation d'une machine par une famille de circuits.

On peut voir l'oracle A comme une suite binaire finie ou infinie (suivant que l'oracle est fini ou non) où le i-ème bit code l'appartenance à l'oracle du i-ème entier. On notera d plutôt que A si l'oracle est fini.

Exercice 1. *Montrer que pour tout oracle $A \subseteq \mathbb{N}$, il existe une machine de code e calculant A à l'aide de l'oracle A. On dit aussi que A est A-calculable.*

Preuve. Soit x l'entrée, il suffit de parcourir la chaîne A et de renvoyer son x-ème bit. ∎

Remarque

Nous avons mentionné la calculabilité d'un ensemble A, ce qui se définit comme la calculabilité de sa fonction caractéristique $\chi_A : x \mapsto \begin{cases} 1 \text{ si } x \in A \\ 0 \text{ sinon} \end{cases}$.

Exercice 2. *Montrer qu'un oracle fini n'ajoute pas de puissance calculatoire (en termes de calculabilité) à une machine.*

Preuve. Soit $e \in \mathbb{N}$ un code d'une machine pouvant faire appel à l'oracle $d \in 2^{<\mathbb{N}}$. On peut construire un nouveau code e' dans lequel d est "codé en dur", puis qui copie e. On obtient $\Phi_e^d = \Phi_{e'}$. ∎

C'est l'idée du théorème SMN, par exemple présenté dans [4]. On pourrait alors se questionner sur l'utilité du formalisme utilisant des oracles finis plutôt que par exemple donner les codes en argument aux fonctions. C'est un choix pris par Allender, permettant de mesurer la complexité en temps directement par rapport à la taille de l'entrée, sans ajouter une constante $|d|$.

La même stratégie ne fonctionne pas pour les oracles infinis, car en toute généralité ils ne peuvent se représenter par un mot fini sur un alphabet fini, et que le code d'une machine est nécessairement fini.

1.2 Modèle du circuit booléen et complexité

1.2.1 Définition

Définition 1.2 (Circuit booléen). *Un circuit booléen D est un graphe connexe orienté acyclique comportant :*

- *n feuilles (nœuds de degré entrant 0) représentant un mot binaire*
- *Trois types de nœuds internes : des portes logiques OU et ET (de degré entrant 2), et la porte NON (de degré entrant 1)*
- *Un unique nœud de degré sortant 0 : la sortie du circuit* ◦

A priori, les circuits peuvent être de taille infinie, car on peut brancher la sortie d'une porte en entrée de plusieurs autres portes.

Notation

On note $\text{size}(D)$ le nombre de portes binaires du circuit D.

Intuitivement, la porte NON est nécessaire pour construire toutes les tables de vérités possibles (si $x = \top$, brancher deux fois x sur OU ou ET ne permet que d'obtenir encore $\top$, on ne peut pas obtenir $\bot$ donc simuler ET), mais le nombre de portes NON est proportionnel au nombre de feuilles plus le nombre de portes binaires car brancher une porte NON après une porte NON ne permet pas de construire plus de tables. C'est donc cohérent de compter seulement les portes binaires.

Définition 1.3 (Calculabilité par circuit). *Un circuit D calcule une fonction $f : 2^n \to 2$ si, pour toute chaîne binaire x de longueur n inscrite dans l'ordre en entrée (sur les feuilles) du circuit D, sa sortie est $f(x)$.* ◦

Une machine de Turing peut prendre des entrée de différentes taille, alors qu'un circuit booléen a une taille d'entrée fixe. Pour relier ces deux modèles, on peut plutôt étudier une machine par rapport à une **famille** de circuits booléens, c'est-à-dire une suite infinie où le i-ème circuit attend une entrée de taille i. Ainsi, les domaines de définition des fonctions calculées sont les mêmes.

En fait, nous pouvons calculer les mêmes choses avec une machine de Turing et avec une famille de circuits. Cela va même plus loin : le **temps de calcul** d'une fonction f par une machine de Turing et la **taille d'une famille** de circuit calculant f sont polynomialement liés. De la même manière, **l'espace** utilisé par une machine est l'analogue de la **profondeur** pour les circuits. Dans ce document, nous allons principalement utiliser la généralisation du lien "taille de circuit / temps de calcul" pour les circuits et machines à oracles, et en faire la preuve section 2.3.

1.2.2 Complexité en circuit

On rappelle que $\text{size}(D)$ est le nombre de portes binaires du circuit D.

Définition 1.4 (Complexité en circuit). *On définit la complexité en circuit d'une fonction $f : 2^n \to 2$ par :*

$$\text{size}(f) = \min\{\text{size}(D) : D \text{ calcule } f\}.$$ ◦

Soient bin_of_int et int_of_bin des fonctions sur les entiers effectuant la conversion décimale vers binaire et inversement.

Notation (Taille en circuit d'une chaîne)

Une fonction $f : 2^n \to 2$ peut se représenter comme une chaîne x_f de longueur 2^n où

$$x[i] = f(\text{bin\_of\_int}(i)).$$

De manière réciproque, on peut représenter une chaîne binaire x comme une fonction :

$$f_x : \left| \begin{array}{ccc} 2^{\lceil \log |x| \rceil} & \longrightarrow & 2 \\ y & \longmapsto & \begin{cases} x[\text{int\_of\_bin}(y)] \text{ si } |y| \leq n \\ 0 \text{ sinon} \end{cases} \end{array} \right.$$

où $2^{\lceil \log |x| \rceil}$ est la plus petite puissance de 2 plus grande que $|x|$. Attention, la fonction $x \mapsto f_x$ n'est pas injective, puisque par exemple : $f_{000} = f_{0000}$ (car $\lceil \log 3 \rceil = \lceil \log 4 \rceil$).

On peut alors parler de complexité en circuit d'une chaîne de caractères : nous avons une première mesure de complexité pour les chaînes. En section 3, nous en verrons une autre due à Kolmogorov, puis nous essayerons de les comparer.

2 Définitions “à la Allender” — Machines efficaces et circuits avec oracle

2.1 UTM efficace

Dans le papier [1], Allender confectionne une définition de machine de Turing universelle dite “efficace” permettant de démontrer un résultat clé : le lien entre C et Size. Une telle machine permet de simuler efficacement, à l'aide d'un oracle fini bien choisi, toute autre machine fonctionnant avec oracle fini. Cela signifie que la perte de temps par rapport à la machine qu'elle simule est petite, à un facteur multiplicatif log près.

Définition 2.1 (machine de Turing universelle efficace, [1]). *Une machine de Turing universelle U à deux bandes est dite universelle si :*

- ***simulation** : pour toute machine M il existe une constante c_M telle que pour tout oracle fini d et toute entrée x, il existe d' tel que $|d'| \leq |d| + c_M$ et $U^{d'}(x) = M^d(x)$, c'est-à-dire :*

$$(\forall M)(\exists c_M)(\forall d, x)(\exists d') \begin{cases} |d'| \leq |d| + c_M \\ U^{d'}(x) = M^d(x). \end{cases}$$

- ***performances** : le temps d'exécution est majoré par $c_M t \log t$ et l'espace utilisé par $c_M s$ où t et s sont respectivement le temps et l'espace utilisés par le calcul $M^d(x)$.* ◦

Il faut bien noter l'utilisation de l'oracle fini d' par U, autrement cela aurait été impossible que U simule différentes machines de Turing.

Positionner le code d de la machine simulée par U en tant qu'oracle fini plutôt qu'en argument (ce qui est fait plus habituellement) permet de simplifier ici nos analyses de complexité.

Remarque

Plus loin nous utiliserons la définition généralisée avec oracle de l'UTM efficace, présentée par Allender dans [1]. Le seul changement est l'ajout de la possibilité aux machines M et U d'avoir accès à un même oracle infini.

Théorème 2.2 (*Hennie, Stearns ([5]) et Fürer([6], [7])*). *Il existe une machine de Turing universelle, au sens de la définition 2.1.* ♢

On fixe alors pour la suite une machine universelle efficace U quelconque.

2.2 Circuit avec oracle

Un deuxième formalisme important proposé par Allender, également décrit dans [1], est l'ajout d'oracle à des circuits booléens. C'est important car cela permet de relativiser le lien “machine/famille de circuits” en établissant un nouveau lien plus général : “machine avec oracle / famille de circuits avec oracle”.

Définition 2.3 (Circuit booléen avec oracle, [1]). *Un circuit booléen avec oracle est un circuit booléen classique auquel on ajoute des portes d'oracle : pour toute arité $r \in \mathbb{N}$, une telle porte prenant r booléens en entrée décide si $b_1 \cdots b_r$ appartient à l'oracle ou non.* ◦

Il y a deux types d'oracles : les **oracles finis** et **les oracles infinis**. Cela ne change pas la définition, mais les finis peuvent être représentés par un mot binaire fini, tandis que les infinis ne peuvent l'être que par des mots binaires infinis.

Une porte d'oracle est bien définie pour toute arité. En effet, il n'y a pas de restriction sur la taille des éléments contenus dans l'oracle : ils peuvent avoir des tailles différentes. C'est-à-dire qu'une même porte pour un certain oracle peut être utilisée avec un nombre différent de branchements d'entrée.

Définition 2.4 (Taille d'un circuit booléen avec oracles). *La taille d'un circuit booléen $C^{A,y}$ avec un oracle infini $A \subseteq \mathbb{N}$ et un oracle fini $y \in \{0,1\}^*$ est le nombre de branchements du circuit. Les portes d'oracle pour A peuvent prendre n'importe quel nombre fini de bits d'entrée, et l'oracle fini y prend en entrée jusqu'à $\log |y|$ bits.* ◦

Dans la littérature, pour parler de circuit "classique" (sans oracle), la taille est souvent définie à partir du nombre de portes, ce qui est moralement la même chose que compter le nombre d'arêtes (ou branchements), puisque les portes ont une arité bien définie à l'avance (2 pour le OU et le ET, 1 pour le NON). Ici, nous raisonnons sur le nombre de branchements, ce qui n'est pas la même chose que le nombre de portes, car les portes d'oracles sont d'arité arbitraire.

Notation (Sous-circuit)

Pour un circuit C et une porte donnée de ce circuit, on peut définir le sous-circuit induit par cette porte, c'est-à-dire le circuit allant des bits d'entrée de C jusqu'au bit de sortie de cette porte. On peut ainsi décomposer un circuit porte par porte, en partant par la dernière (celle calculant le bit de sortie) puis considérer les sous-circuits produisant les entrées de cette dernière porte, et ce récursivement. Une circuit ayant pour dernière porte la porte NOT se décompose en une étape en un sous-circuit, ceux de porte finale OU et ET en deux sous-circuits, et ceux de porte finale une porte d'oracle de taille r, en r sous-circuits.

2.3 Un premier résultat : "TM = Circuits" se relativise

Comme annoncé, nous allons montrer que le lien entre machine et famille de circuit se relativise. Bien que les propositions figurent dans le papier d'Allender, elles ne sont pas prouvées. Nous les avons donc écrites en généralisant les preuves du livre de Sylvain Perifel ([3]).

Proposition 2.5 (Simulation d'un circuit, *adapté de Sylvain Perifel 5-X, [3]*). *Il existe une machine M telle que :*

- ***simulation*** *: pour tout circuit C^y de taille m ayant accès à l'oracle fini y calculant une fonction $2^n \to 2$, il existe un encodage d_C de C de taille $O(m(\log m + \log n))$ tel que pour toute chaîne x de taille n, $M^{d_C,y}(x) = C^y(x)$.*
- ***performances*** *: le temps d'exécution est en $O(m^2 \log m + m \log n)$*
- ***comportement en dehors du domaine*** *: pour les entrées de taille $\neq n$, $M^{d_C,y(x)}$ renvoie $*$ en temps $O(\log n)$.*

c_M est une constante, tandis que d, d' et x sont des chaînes binaires. ♢

Idée de preuve

Nous donnons un algorithme récursif réalisant, à partir d'un encodage d'un circuit C et d'une entrée x, le calcul $C(x)$. Nous majorons ensuite grossièrement le temps d'exécution d'un tel algorithme, ce qui s'avère suffisant.
Nous pourrions essayer d'améliorer la borne en représentant un circuit avec une structure récursive et en menant une analyse computationnelle plus fine. Nous aurions ainsi une complexité linéaire en la taille du code du circuit. La possibilité d'une telle efficacité par rapport à celle annoncée par Allender est peut-être liée au fait que nous relâchons l'hypothèse que la machine fonctionne avec seulement deux bandes de travail.

PREUVE. Soit un circuit C de taille m, décrivons un **encodage binaire** pour C, de taille $O(m(\log n + \log m))$.

On construit un graphe orienté connexe à m arêtes en prenant pour ensemble de sommets S les n bits d'entrée ainsi que toutes les portes sur un chemin d'un bit d'entrée vers le bit de sortie. Nous avons donc $|S| \leq n + m$ sommets car le graphe est connexe. Numérotons les sommets de 1 à $|S|$ et étiquetons-les par leur type (porte NON, OU, ET ou porte d'un oracle). Il y a $O(m)$ étiquettes différentes car on utilise au plus un oracle différent par arête. Ainsi, nous pouvons représenter le graphe comme une liste d'arêtes, avec une chaîne binaire de taille $O(m(\log n + \log m))$. La liste est de taille m et contient $2 * m$ sommets, chacun représenté comme un couple d'entiers (`type_de_sommet`, `numéro_de_sommet`) de taille :

$$\log m + \log(m+n) = O(\log(m+n)) \leq O(\log(m*n)) = O(\log m + \log n)$$

ce qui justifie l'encodage de taille $O(m(\log n + \log m))$.

Le code suivant utilise une représentation plus efficace qu'une liste d'adjacence : on représente un circuit comme une structure récursive afin d'obtenir plus facilement les sous-circuits (il suffit de détruire la structure d'un constructeur pour accéder aux sous-circuits d'une étape), qui coûte aussi une taille linéaire. Néanmoins, il est suffisant pour l'analyse temporelle que nous allons conduire de supposer que nous reconstruisons un nouveau circuit à chaque étape. Précisons qu'avec la structure initialement décrite, l'accès aux portes produisant les bits d'entrée de la porte finale s'effectue en $O(m(\log n + \log m))$, car nous devons parcourir toute la représentation du circuit pour l'interpréter, quitte à le faire à la volée.

```
(* On représente un oracle comme un ensemble de chaînes. *)
module ChaineSet = Set.Make(struct
    type t = bool list
    let compare = compare (* convertit une chaîne de booléens en un entier
        (e.g 0010 donne 10) puis compare les entiers *)
  end)

type chaine = bool array
type circuit =
          | Input of int
          | Constante of bool
          | Not of circuit
          | Or of circuit * circuit
          | And of circuit * circuit
          | Oracle of ChaineSet.t * (circuit list)

let eval (x: chaine) (c: circuit) : bool =
  let rec eval_aux : circuit -> bool = function
```

```
  | Input i -> x.(i)
  | Constante b -> b
  | Not c -> not (eval_aux c)
  | Or (c, c') -> eval_aux c || eval_aux c'
  | And (c, c') -> eval_aux c && eval_aux c'
  | Oracle (oracle, cs) ->
    let input = List.map eval_aux cs in
    ChaineSet.mem input oracle
in
eval_aux c
```

On note $c_m = O(m(\log n + \log m))$ correspondant au temps nécessaire pour produire un code pour un sous-circuit d'un circuit de taille m. En effet, en une passe nous devrions pouvoir extraire depuis un circuit le sous-circuit voulu.

L'analyse de complexité temporelle donne, en notant C^m un circuit de taille m, et $C_1^k, \cdots, C_k^{m-k}$ ses sous-circuits de taille $m-k$ où k est l'arité de la porte finale de C^m :

$$\begin{aligned} T(|C^m|) \leq \max\{ & rc_m + \Sigma_{i=1}^r T(|C_i^{m-i}|); && \textit{cas de porte d'oracle} \\ & 2c_m + T(|C_1^{m-2}|) + T(|C_2^{m-2}|); && \textit{cas de porte OU ou ET} \\ & c_m + T(|C_1^{m-1}|); && \textit{cas de porte NON} \\ & O(1)\} && \textit{cas de bit d'entrée ou de sortie} \end{aligned}$$

avec la convention que $C^m = 0$ pour $m < 0$.

Nous remarquons naturellement que les cas de porte NON et porte OU et ET se confondent avec les cas de porte d'oracle pour respectivement $k = 1$ et $k = 2$, dès lors le problème pour trouver une majoration revient à résoudre le problème d'optimisation pour s variables $k_1, \cdots, k_s \geq 1$ avec la contrainte $\Sigma_{i=1}^s k_i = m$ et maximisant $\Sigma_{i=1}^s k_i \times c_{(m-\Sigma_{\{k_j : j<i\}} k_j)}$. Mais, nous pouvons aussi effectuer la majoration suivante, plus simple. L'idée est la suivante : puisque le coût en temps de l'algorithme est la somme des temps pour construire les m sous-circuits, nous majorons par le coût de m sous-circuits de taille c_m :

$$T(|C^m|) \leq mc_m = O(m^2(\log m + \log n))$$

On obtient un facteur m supplémentaire sur le $\log n$ par rapport à la majoration attendue, nous allons travailler plutôt avec cette borne légèrement plus grande ce qui ne devrait pas avoir un impact très significatif. Tester la longueur de l'entrée peut se faire au début de l'algorithme, avec un compteur se déplaçant le long de x, et renvoyant $*$ s'il dépasse n. ∎

Proposition 2.6 (Simulation par circuits). *Pour toute machine M avec oracle A, il existe une constante c_M telle que pour tout oracle fini y et entrée x, si le temps d'exécution de $M^{A,y}(x)$ est t, alors il existe une famille de circuits $(C_n)_{n\in\mathbb{N}}$ de taille $c_M t(t^2 + |x| + \log|y|)$ calculant $M^{A,y}(x)$ avec l'oracle y, c'est-à-dire :*

$$(\forall M^A)(\exists c_M)(\forall y)(\exists (C_n)_{n\in\mathbb{N}})(\forall x) \begin{cases} C_{|x|}^{A,y}(x) = M^{A,y}(x) \text{ en temps } t = t(x,y) \\ \text{Size}(C_{|x|}^{A,y}) = c_M t(t^2 + |x| + \log|y|) \end{cases}$$

Formalisme d'une machine du Turing avec oracles

Une bande est une suite infinie (vers la droite) de cellules, numérotées de 0 à $+\infty$. Chaque cellule contient exactement un symbole d'un alphabet fini noté Γ. Nous considérons un formalisme classique de la machine de Turing avec un nombre arbitraire k de bandes de travail, et détaillons le formalisme lié aux oracles. Une machine à R oracles possède une **bande de requête** par oracle, un **état de requête** $q_{i?}$ par oracle, et deux **états de décision** q_y et q_n permettant de décider l'appartenance d'un mot écrit sur une bande de requête. Pour questionner l'oracle sur un mot x, on écrit x sur la bande de requête correspondant à cet oracle, entrons dans l'état de requête $q_{i?}$ correspondant, et l'oracle nous permettra en une étape de rentrer dans l'état q_y ou q_n.

Configuration d'une machine avec oracles

Une configuration, notée $\mathcal{C}$, représente parfaitement l'"état" d'une machine à une certaine moment de son exécution, au sens que deux configurations d'une même machine vont suivre la même exécution future. Elle contient :

- l'état courant q,
- le contenu des bandes de travail et de requête,
- les positions des têtes de lecture de la bande d'entrée, de sortie, et des bandes de travail et de requête.

Exécution d'une machine

On note $\mathcal{C} \vdash \mathcal{C}'$ si l'on peut atteindre la configuration C' à partir de C en une étape élémentaire, c'est-à-dire que la fonction de transition δ de la machine qui, à partir de :

- l'état courant $q \in Q$,
- le contenu des cellules pointées par les têtes de lecture $(b_1, \cdots, b_{k+R+2}) \in \Gamma^{k+R+2}$.

renvoie :

- le nouvel état courant q',
- l'indice d'une bande de travail ou de requête i,
- le symbole b à écrire sous la tête de lecture de la bande i,
- le sens (gauche ou droite) dans lequel déplacer la tête de lecture modifiée $S \in \{G; D\}$.

L'exécution d'une machine de Turing en temps $t(|x|)$ se représente comme une suite $(\mathcal{C}_{i \leq t(|x|)})$ de $t(|x|)$ configurations telles que :

- $\mathcal{C}_0$ correspond à la machine de Turing avec ses bandes en écritures (travail, requête, sortie) initialisées au caractère vide (#) et ses têtes de lectures sur la première cellule des bandes,
- $\mathcal{C}_i \vdash \mathcal{C}_{i+1},\ \forall i < t(|x|)$,
- l'état courant de $\mathcal{C}_{t(|x|)}$ est final, et nous appelons sortie de la machine l'entier représenté sur la bande de sortie.

Nous avons maintenant une idée du formalisme à atteindre pour simuler une machine avec nos circuits.

PREUVE. Soit x une entrée de taille n telle que $M^{A,y}(x)$ s'exécute en $t(|x|)$ étapes, construisons un circuit C simulant M. Le circuit C va comporter $t(|x|)$ niveaux, chacun correspondant à une configuration de l'exécution de la machine sur l'entrée x. Au niveau 0, nous encodons la configuration $\mathcal{C}_0$ avec un nombre constant (indépendant de x) de portes qui permettent de représenter l'état actuel (qui appartient à un alphabet fini), ainsi qu'un nombre $O(t(|x|))$ de bandes permettant de représenter les bits de l'entrée utilisés par la machine.

⚠ (Représentation en circuits des bandes d'une machine de Turing)

Naturellement, nous utilisons le fait qu'à tout moment de l'exécution d'une machine de Turing, chaque bande contient un nombre fini de cellules contenant un symbole différent de #, et nous représentons donc avec les circuits seulement les bandes jusqu'à la plus grande cellule (en indice) qui contient un symbole différent de #. En effet, se déplacer sur une bande coûte une étape de calcul, donc après k étapes de calcul nous sommes au plus à la position k de la bande, car les bandes sont initialisés avec les têtes en position 0.

Puis, à partir d'une configuration donnée, nous pouvons représenter la suivante avec un nombre $O(t(|x|))$ de portes : Si $\delta(q, b_1, \cdots, b_{k+R+2}) = (q', i, b, S)$, montrons que nous pouvons simuler δ avec un circuit :

Simulation par circuit d'une fonction de taille d'entrée fixe

Justifions que nous pouvons construire un circuit C de taille constante, à plusieurs sorties, qui simule une fonction de taille d'entrée et de sortie fixe $f : \{0,1\}^n \to \{0,1\}^m$. Chaque bit de la sortie est exactement déterminé par l'entrée de taille n, or pour toute fonction à n bits renvoyant un booléen, il existe une formule booléenne φ à n variables et de taille finie qui est satisfaite ssi la fonction renvoie le booléen "1". Mais alors, il existe un circuit de taille fini calculant ce bit de sortie. Finalement, en combinant m circuits de taille finie, nous construisons un circuit de taille finie calculant f.

Puisque $Q \times \Gamma^{k+R+2}$ et $Q \times [k+R+2] \times \{0,1\} \times \{0,1\}$ sont des ensembles finis, on peut simuler la fonction δ avec un circuit de taille constante. La configuration suivante peut donc bien se représenter avec un nombre $O(t(|x|))$ de portes, et au final le circuit simulant M possède $O(t(|x|)^2)$ portes logiques.

Donnons maintenant une borne sur le nombre de fils. Au plus, chaque configuration fait un appel à l'oracle le plus coûteux, ce qui nécessite $O(\max(t(|x|), \log n_1)) = O(n + \log n_1)$ fils. En effet, si nous appelons l'oracle infini A, cela peut être fait pour une taille arbitraire, majorée par $t(n)$. L'oracle fini y_1 peut être appelé pour une taille au plus $\log n_1$ où n_1 est la longueur de y_1. La longueur $n = |x|$ n'entre pas dans le calcul du nombre de fils utilisés, puisque la portion de x utilisée lors du calcul est majoré par le temps d'exécution $t(n)$. Mais, par le même argument, $y_1 = O(t(n))$. Puis, chaque configuration requiert le contenu de la configuration précédente, donc le circuit pour représenter une configuration $O(t(n) + t(n)) = O(t(n))$. Puisqu'il y a $O(t(n))$ configurations, nous pouvons conclure que la taille du circuit en nombre de fils est $O(t(n)^2)$.

Nous pourrions penser que puisque les configurations d'une machine de Turing varient lentement (on construit une requête à un oracle bit par bit), le coût d'une requête est amortie. Mais ça n'est pas le cas dans le modèle décrit, car nous pouvons très bien remplir une bande de requête, effectuer une requête, modifier d'un nombre constant de bits la requêtes, et effectuer une nouvelle requête. Ainsi dans notre comptage de fils nous devons considérer qu'un oracle par configuration peut être effectué. Néanmoins cela importe peu car le coût de recopier le contenu des bandes de la configuration précédente est déjà en $O(t(n))$.

Nous pourrions être surpris par le fait que la taille du circuit est indépendante de la taille des oracles, c'est parce que dans notre modèle le temps d'exécution d'une machine majore la taille des requêtes que nous pourrions effectuer à des oracles.

Cela conclut la preuve. ∎

3 Différentes complexités de Kolmogorov

3.1 "Classique"

La première complexité de Kolmogorov que nous allons étudier, initialement introduite en 1963 par Andreï Kolmogorov, s'énonce simplement mais possède déjà une grande profondeur conceptuelle.

> **Définition 3.1 (Complexité de Kolmogorov sur une machine quelconque).** *Pour une machine de Turing M et une chaîne binaire finie x, on définit :*
>
> $$C_M(x) = \min\{|d| : M(d) = x\},$$
>
> *la taille de la plus petite chaîne d permettant de restituer x à l'aide de M.* ◦

Si une machine M ne permet pas de produire une chaîne x, on prendra la convention que $C_M(x) = +\infty$. Ainsi, la mesure C_M est toujours définie.

L'intuition derrière cette complexité est la suivante : nous essayons de compresser au maximum x (en tant que descripteur d) de telle sorte que la machine M effectue la décompression, quand on lui donne d en entrée. En un sens, c'est une mesure du désordre ou encore de l'irrégularité de la chaîne x relativement à la machine M.

Exemple 1. *On peut imaginer une machine rudimentaire permettant d'atténuer les longues répétitions du même bit : un bit b se code $0b$ et une suite de $|x|$ fois le bit b, où x est une chaîne binaire représentant un entier codé en binaire, se code $1b1x_n1x_{n-1}1\dots1x_1$.*

Ainsi, $m(00)(11)(10)(10)(11)(10)(11)(00) = 01^{20}0$: un 0 suivi de $10100_b = 20$ fois le bit 1 suivi d'un autre 0. On a un descripteur de taille $2+2+5*2+2 = 16$ pour une chaîne de taille $1+2^4+2^2+1 = 22$. La définition a donc un sens, il est possible de compresser certaines chaînes. Une telle machine pourrait avoir le code suivant :

```
type bit = Z | O
type chaine = bit list
let rec m : chaine -> chaine = function
  | [] -> []
  | Z::b::t -> b :: m t
  | O::b::t -> begin
    let rec aux pow2 acc = function
      | O::Z::t -> aux (2*pow2) acc t
      | O::O::t -> aux (2*pow2) (acc + pow2) t
      | [] | [O] -> List.init acc (Fun.const b)
      | Z::t -> List.init acc (Fun.const b) @ m (Z::t)
    in
    aux 1 0 t
    end
  | _ -> failwith "undefined"
```

Pour aller plus loin, on peut construire une machine reconnaissant des motifs plus complexes ("alterner i fois entre 0 et 1 pour i allant de 1 jusqu'à 10", "afficher les 100 premières décimales de π") puis se questionner sur l'existence et le code d'une "meilleure machine", qui permettrait d'obtenir toutes les chaînes à partir de la description la plus courte possible.

Une première remarque qui peut être faite est que **pour toute chaîne x, il existe une machine M telle que $C_M(x) = 0$**, c'est-à-dire qui produit x à partir de la chaîne vide. En effet, nous pouvons prendre le code trivial suivant :

```
let m_x d =
  if d = "" then x else d
```

Il n'existe donc pas de machine ayant exactement une meilleure complexité de Kolmogorov que toutes les autres machines pour toutes les chaînes à produire.

Néanmoins, nous allons voir l'intérêt de raisonner à constante additive près.

Définition 3.2 (Machine universelle). *Une machine U est dite* ***universelle*** *si pour toute machine M,*

$$\exists c_M \in \mathbb{N},\ \forall x \in 2^{<\mathbb{N}},\ C_U(x) \leq C_M(x) + c_M.$$ ∘

C'est-à-dire que la machine U est optimale, à constante près. En considérant une machine universelle, nous pourrons parler plus généralement de complexité d'une chaîne sans fixer de machine. Montrons d'abord l'existence d'une telle machine.

Théorème 3.3 (Théorème d'invariance pour la mesure C). *Il existe une machine universelle.*♢

Preuve. On définit explicitement une telle machine, comme une variante d'une machine universelle. La différence avec la définition classique est que nous utilisons une astuce pour encoder l'entrée x et le code e donnés en argument afin de majorer plus facilement la taille de cet argument :

$$U : 0^n 1 e \mapsto \Phi_e(n),\ \text{pour } e \text{ un code binaire.}$$

La fonction partielle U est **bien définie**, car pour toute chaîne x, il existe au plus un couple $(n, e) \in \mathbb{N} \times 2^{<\mathbb{N}}$ tel que $x = 0^n 1 e$. En effet, soient (n, e) et (n', e') tels que $x = 0^n 1 e = 0^{n'} 1 e'$. Si $n < n'$, $x[n+1] = 1$ mais $x[\leq n'] = 0$ ce qui est absurde. Puis $0^n 1 e = 0^n 1 e' \implies (n, e) = (n', e')$.

Soit M une machine de Turing. Montrons qu'il existe une constante c_M telle que pour toute chaîne x, U est universelle. C'est-à-dire : $C_U(x) \leq C_M(x) + c_M$. Choisissons une description d de x de taille minimale pour M, c'est-à-dire $|d| = C_M(x)$. Alors, par définition de U,

$$U(0^d 1 \langle M \rangle) = M(d) = x$$

où $\langle M \rangle$ est une représentation binaire d'un code pour M. On possède un descripteur permettant de majorer la mesure C_U par :

$$C_U(x) \leq |0^d 1 \langle M \rangle| = |d| + |1\langle M \rangle| = C_M(x) + 1 + |\langle M \rangle|$$

Choisir la constante $c_M = 1 + |\langle M \rangle|$ nous permet de conclure. ∎

Remarque

On comprend grâce à la preuve du théorème précédent que **les définitions** de machine universelle au sens de la "compression optimale" pour la mesure C et au sens de la simulation, **coïncident**.

Définition 3.4 (Une première complexité de Kolmogorov). *Pour une machine universelle U (sans oracle, au sens de la définition 3.2), on définit :*

$$C(x) = \min\{|d| : U(d) = x\}.$$ ∘

Théorème 3.5. *La complexité de Kolmogorov n'est pas calculable, c'est-à-dire que la fonction :*

$$C_{\mathbb{N}} : \left| \begin{array}{ccc} \mathbb{N} & \longrightarrow & \mathbb{N} \\ n & \longmapsto & C(\delta(n)) \end{array} \right.$$

n'est pas calculable, δ étant une notation pour $2^{<\mathbb{N}}$. ♢

Mais, on peut montrer que $C_{\mathbb{N}}$ est calculable à la limite ($C_{\mathbb{N}} \leq_T \emptyset'$), et même qu'elle est complète pour la réduction Turing ($C_{\mathbb{N}} \equiv_T \emptyset'$).

Puisque δ est calculable, on peut parler indifféremment de calculabilité sur des fonctions $2^{<\mathbb{N}} \to 2^{<\mathbb{N}}$ et des fonctions $\mathbb{N} \to \mathbb{N}$.

3.2 Avec borne de temps

Nous introduisons des variantes de la complexité de Kolmogorov dans le but de faire le lien avec la complexité en circuit.

3.2.1 Du type "Allender"

Définition 3.6. *Soit U une machine universelle et soit $t : \mathbb{N} \to \mathbb{N}$ une fonction calculable, appelée borne en temps. On définit :*

$$C_U^t(x) = \min\{|d| : U(d)[t(|d|)] \downarrow = x\}$$

la complexité de Kolmogorov sur la machine U avec la borne de temps t, où $U(d)[t(|d|)] \downarrow = x$ signifie que la machine U sur l'entrée d s'arrête en $t(|d|)$ étapes avec la chaîne x écrite sur sa bande de sortie. ◦

Cette nouvelle complexité contient une contrainte supplémentaire : nous devons pouvoir restituer x à partir du descripteur d en temps borné par une fonction calculable fixée t. Pour toute fonction calculable t, cette mesure est donc plus grossière que C_U :

$$\forall x \in 2^{<\mathbb{N}},\ C_U(x) \leq C_U^t(x).$$

Elle l'est même strictement, puisqu'elle est calculable alors que C ne l'est pas.

Théorème 3.7. *Soit U une machine universelle (au sens de l'émulation). Alors pour tout time-out t, la fonction C_U^t est calculable.* ♢

Preuve. Donnons un code pour C_U^t :

```
val t : int -> int (* time-out             *)
val u : int -> int (* machine universelle *)

let c x =
  for d = 0 to +∞ do
    for t = 1 to t(|d|) do
      if U(d)[t] ↓= x then (* opti. possible en sortant prématurément *)
          |d|             (* de la boucle sur t lorsque U(d)[t] ↓≠ x     *)
```

On énumère les descripteurs potentiels par taille croissante et exécutons U pour chacun de ces descripteurs avec le timeout t. S'il existe un descripteur pour x, on est certain de terminer et de renvoyer la mesure attendue ; sinon, on boucle à l'infini (mesure indéfinie, x est en dehors du domaine). ■

Théorème 3.8 (Allender, [1]). *Le théorème d'invariance n'est pas vérifié pour la mesure C^t, c'est-à-dire qu'il n'existe pas de machine U telle que pour toute machine M, il existe c_M telle que pour toute chaîne x, $C_U^t(x) \leq C_M^t(x) + c_M$.* ♢

La définition C^t n'est donc pas satisfaisante, nous allons en étudier une autre. Définissons une nouvelle mesure bornée en temps, qui vérifiera le théorème d'invariance.

Définition 3.9 (*Leonid Levin, [8]*). *Pour une machine universelle U et un time-out t, on définit la mesure :*

$$\mathrm{Kt}_U(x) = \min\{|d| + \log t(|d|) : U(d)[t(|d|)] \downarrow = x\}$$ ∘

Exemple 2 (Éric Allender ([1])). *Soit* Kt *une mesure (fonction qui à une chaîne binaire finie associe un entier, voir 3.9). On définit $R_{\mathrm{Kt}} = \{x \in 2^{<\mathbb{N}} : \mathrm{Kt}(x) \geq \frac{|x|}{2}\}$ comme étant "les chaînes aléatoires selon la mesure Kt" (voir partie 2.2 pour plus de détails).*

Alors, la classe des langages reconnus par une machine de Turing non-déterministe en temps polynomial qui ont accès à un oracle pour R_{Kt} coïncide avec la classe des langages reconnus par une machine déterministe en temps exponentiel, ce qui se note :

$$\mathrm{NP}^{R_{\mathrm{Kt}}} = \mathrm{EXP}$$

Pourtant, on ne sait pas si NP = EXP *: l'utilisation de l'oracle permet de définir un modèle plus puissant.*

Comme l'explique Eric Allender dans son article, on pourrait trouver le log peu naturel et définir une mesure utilisant une fonction polynomiale en $t(|d|)$, comme $|d| \mapsto t(|d|)$. Mais, dans ce cas, nous obtiendrions une mesure triviale ce qui n'est pas intéressant. En revanche, en traduisant la définition de "problème de recherche" vers "problème de décision", nous pouvons construire une complexité non-triviale incluant simplement la fonction $t(|d|)$ dans sa définition.

Définition 3.10 (KT, *Allender, [1]*). *Soit U une machine universelle au sens de la définition 2.1. On définit la mesure* KT *par :*

$$\mathrm{KT}(x) = \min_{(d,T)\in 2^{<\mathbb{N}}\times\mathbb{N}} \{|d| + T : (\forall b \in \{0,1,*\})(\forall i \leq |x|+1),\ U^d(i,b)[T]\downarrow \wedge U^d(i,b)[T] = 1 \Leftrightarrow x_i = b\}$$ ∘

Intuitivement, cela revient à chercher le compromis optimal (au sens de la somme) entre une taille de descripteur $|d|$ et un time-out T tel que la machine universelle U avec oracle d calcule en temps T le i-ème bit de la chaîne x. Plus précisément, la machine U doit aussi pouvoir reconnaître la longueur de la chaîne (lorsque $i = |x| + 1$ et $b = *$).

Pour établir le lien entre la définition de la complexité de Kolmogorov (3.4) de celle de sa version incluant le temps d'exécution (3.10), nous donnons une nouvelle version de la définition de la complexité de Kolmogorov, dite décisionnelle, puis nous montrons que ces deux définitions coïncident.

Définition 3.11 (Complexité de Kolmogorov décisionnelle). *Pour une machine U universelle au sens de la définition 2.1, on définit la complexité de Kolmogorov d'une chaîne x comme la taille du plus petit oracle fini d permettant de décider un bit arbitraire de x à l'aide de U :*

$$\mathrm{Cd}(x) = \min\{|d| : \forall b \in \{0,1,*\},\ \forall i \leq |x|+1,\ U^d(i,b) = 1 \text{ ssi } x_i = b\}.$$ ∘

Proposition 3.12 (C = Cd). *Soit x une chaîne binaire. Montrons que $\mathrm{C}(x) = \mathrm{Cd}(x)$, à constante additive près.* ♢

Preuve. 1. Soit d tel que $\mathrm{Cd}(x) = |d|$. Pour toute machine universelle avec oracle, $U^d(i,b)$ décide si le i-ème bit de x est b. Montrons l'existence d'une machine sans oracle M telle que $\mathrm{M}(d) = x$.

```
(* u d i b renvoie "1" ou "0" selon si x[i] = b ou non, et renvoie "*"
    si i ≥ |x|. *)
val u : bool array -> int -> bool -> string
```

```
let m (d: int) : int =
  let i = ref 0 in
  let x = ref "" in
  while u d !i 1 <> "*" do
    x := !x ^ (u d !i 1);
    incr i
  done;
  int_of_string !x
```

Cela nous donne la première majoration : $\mathrm{C}(x) \leq \mathrm{Cd}(x)$.

2. Soit d tel que $\mathrm{C}(x) = |d|$.

```
let u d i b =
  val m : int -> int (* code d'une machine telle que m d = x*)
  let x = string_of_int (m d) in
  if i > String.length x - 1 then "*"
  else
    let r = (bool_of_string x.[i]) = b in string_of_bool r
```

On obtient la deuxième majoration : $\mathrm{Cd}(x) \leq \mathrm{C}(x)$. ∎

On confondra alors C et Cd.

Finalement, c'est la version de KT généralisée aux oracles proposée par Allender qui nous servira dans la suite.

Définition 3.13 (KT avec oracle, *Allender, [1]*). *Soit A un oracle infini et U une machine universelle au sens de la définition 2.1. On définit la mesure KT^A par :*

$$\mathrm{KT}^A(x) = \min_{(d,t)\in 2^{<\mathbb{N}}\times\mathbb{N}} \{|d| + t : (\forall b \in \{0,1,*\})(\forall i \leq |x|+1),\ U^{A,d}(i,b)[t]\downarrow \wedge U^{A,d}(i,b)[t] = 1 \Leftrightarrow x_i = b\}$$

∘

3.2.2 Du type "Balcázar"

José L. Balcázar a initialement introduit en 1997 dans son papier [2] la complexité de Kolmogorov suivante, pour établir des résultats que nous étudierons en section 4.

Définition 3.14 (Complexité de Kolmogorov, Balcázar). *Soit U une machine de Turing universelle, $f, g : \mathbb{N} \to \mathbb{N}$ et $\alpha \in 2^\omega$ une chaîne binaire **infinie**. Alors, $\alpha \in K[f,g]$ s'il existe $\beta \in 2^\omega$ telle que :*

$$U(n, \beta_{\leq m}) = \alpha_{\leq n}, \quad \text{en temps } g(n), \text{ pour tout } m \geq f(n)\ ,$$

où $\alpha_{\leq m}$ est la chaîne binaire finie constituée des m premiers bits de α. ∘

f est donc la taille du conseil nécessaire pour le calcul, et g le temps d'exécution.

Mais, le papier d'Éric Allender [1] paru onze années plus tard, en 2008, fournit entre autres des idées de définitions permettant d'élargir les résultats de Balcázar, pourvu que nous adaptions les preuves. C'est ce que nous allons faire : en nous inspirant de la définition KT d'Allender, nous définissons K' qui devrait conserver les propriétés de K tout en étant compatible en un certain sens avec KT.

Remarque

Effectivement, K et KT ne semblent pas directement reliées. Néanmoins, c'est un phénomène fréquent en complexité, où deux objets ont la même "nature" et définissent la même "hiérarchie", mais l'élément de base n'est pas forcément au même étage donc ils se sont pas directement égaux (par contre l' "union" des étages l'est).

Nous proposons donc la définition suivante.

Définition 3.15 (K′, Balcázar "corrigé"). *Soit $\alpha \in 2^{\mathbb{N}}$ une chaîne binaire infinie, et $f, g : \mathbb{N} \to \mathbb{N}$ des fonctions sur les entiers. On note $\alpha \in \mathrm{K}'[f,g]$ s'il existe une chaîne binaire infinie β appelée descripteur telle que pour tout entier n les $f(n)$ premiers bits de β permettent de calculer chacun des bits du préfixe de α de taille n à l'aide d'un circuit booléen de taille bornée par g. Formellement :*

$$\alpha \in \mathrm{K}'[f,g] \iff \exists \beta,\ \forall n,\ \exists C,\ |C| \leq g(n)$$
$$\wedge\, \forall (i,b) \in [n+1] \times \{0,1,*\},\ C^{\beta_{|f(n)}}(i,b) = 1 \text{ ssi } \alpha_i = b$$

Attention, la notation α_i est quelque peu abusive, elle représente le i-ème bit du préfixe $\alpha_{|n}$, c'est-à-dire que $\alpha_{n+1} = *$. ∘

Remarque

La restriction pour la machine d'autoriser $i = n+1$ impose à cette dernière de savoir reconnaître la longueur du mot dont on calcule le i-ème bit. En effet, nous souhaitons reprendre la convention d'Allender, où tout mot fini est succédé par une infinité de symboles $*$.

Notation

On note alors $\mathrm{K}'[\mathcal{F},\mathcal{G}]$ l'ensemble des chaînes $\alpha \in 2^\omega$ telles qu'il existe $f, g \in \mathcal{F} \times \mathcal{G}$ telles que $\alpha \in \mathrm{K}'[f,g]$.

Nous utiliserons cette mesure dans la section 4.

3.3 Un deuxième résultat : "$C = \mathrm{KT}^{\mathrm{H}} = \mathrm{Size}^{\mathrm{H}}$"

Notre objectif est de lier la complexité en circuit (Size) avec la complexité K définie par Balcázar (ou plus précisément K'), dans le but d'**exprimer les classes de complexités** mentionnées dans son article **en termes de circuits booléens**. Pour cela, redémontrons un résultat clé d'Allender : $\mathrm{C} = \mathrm{Size}^H$. On va procéder par "transitivité" : montrer que $\mathrm{Size}^H = \mathrm{KT}^H$, puis que $\mathrm{C} = \mathrm{KT}^H$. On obtient la première égalité en relativisant la preuve suivante :

Remarque (Égalité à polynôme près)

L'égalité entre les mesures dans le paragraphe précédent s'interprète comme "est polynomialement liée à". Par exemple, $\mathrm{C} = \mathrm{Size}^H$ signifie qu'il existe un polynôme p tel que pour toute chaîne x, $C(x) = p(\mathrm{Size}^H(x))$. Nous raisonnons à égalité polynomiale près (mais montrons des résultats un peu plus précis), et pour insister nous pourrons noter $=_p$.

Théorème 3.16 (Size = KT, Allender [1]). *Il existe une constante c telle que pour toute chaîne binaire x :*

1. $\mathrm{Size}(x) \leq c(\mathrm{KT}(x))^2((\mathrm{KT}(x))^2 + \log|x|)$

2. $\mathrm{KT}(x) \leq c(\mathrm{Size}(x))^2(\log \mathrm{Size}(x) + \log\log|x|)$

PREUVE. 1. Soit x une chaîne de taille n telle que $\mathrm{KT}(x) = m$. Alors, il existe un descripteur d_x de taille $\leq m$ et un temps $t \leq m$ permettant de calculer le i-ème bit de x, c'est-à-dire tel que :

$$U^{d_x}(i,b)[m]\downarrow \Leftrightarrow x_i = b.$$

En effet, on applique la définition de KT sachant $|d_x| + t = m \implies |d_x| \leq m \wedge t \leq m$. Construisons une machine M^{d_x} prenant en entrée un entier i, exécutant $U^{d_x}(i,1)$ sur m étapes renvoyant 0 si la machine ne s'arrête pas, et renvoyant 1 sinon. Le temps d'exécution de M est asymptotiquement le même que U. Donc, il existe un circuit $C_n^{d_x}$ de taille $O(t(|x|)^2)$ tel que $C_n^{d_x}(i) = M^{d_x}(i)$ pour tout $i \leq n+1$.

Avec une circuiterie prenant en entrée un indice i tel que $2^{k-1} < i \leq 2^k$ où 2^k est la plus petite puissance de deux plus grande que n, nous pouvons construire un circuit $C_n'^{d_x}$ calculant $f_x(i)$: avec un bout de circuit de taille $O(t(|x|))$ nous pouvons décider si $i > |x|$, si c'est le cas nous renvoyons 0, et sinon nous exécutons le circuit $C^{d_x}(i)$. Ce dernier est de taille $O(t(|x|)^2)$, en appliquant la proposition de simulation de machine par des circuits (2.6) à la machine M. Donc, $C_n'^{d_x}$ est aussi de taille $O(t(|x|)^2)$.

Lemme 3.17 (Taille d'un circuit simulant un oracle fini). *Soit C^y un circuit de taille m ayant recours à l'oracle fini y de taille n_1. Alors il existe un circuit sans oracle C de taille*

$$O(m \log\log n_1)$$

simulant C^y. ♢

PREUVE (LEMME). Si nous avons un circuit C^y de taille m utilisant un oracle fini y de taille n_1, il est possible de construire un circuit C effectuant les mêmes calculs que C^y mais n'ayant pas recours à l'oracle y. Étudions alors la taille d'un tel circuit C. L'idée est de construire un sous-circuit prenant en entrée un entier i codé sur $\log|y|$ bits, et renvoyant $y[i]$ si $i \leq |y|$, et 0 sinon. Pour cela nous pouvons employer un démultiplexeur $\log|y|$ vers $|y|$, qui met la $\log|y|$-ème sortie à $\top$ et les autres à $\bot$, appliquer à la i-ème sortie du démultiplexeur la porte ET avec comme deuxième entrée le i-ème bit de y (ou la constante 0 s'il n'existe pas), puis renvoyer le ET des $|y|$ sorties obtenues.

Un démultiplexeur k vers 2^k peut se coder avec k portes NON et 2^k portes ET de taille k. Une porte ET de taille k peut se coder avec $\lceil \log k \rceil$ portes ET de taille 2. Donc notre démultiplexeur peut se coder en utilisant $2^k * \lceil \log k \rceil$ portes binaires ET. Au final, le démultiplexeur a une taille :

$$k + 2 \times 2^k \times \lceil \log k \rceil = O(2^k \times \log k).$$

Puisque l'on ajoute à ce circuit 2^k portes ET, puis que l'on effectue un OU de taille 2^k (qui se code avec $\lceil \log k \rceil$ portes OU binaires), la taille du sous-circuit entier est asymptotiquement la même.

Enfin, pour construire C simulant C^y, nous pouvons reprendre la circuiterie de C^y, et chaque fois que nous utilisions une porte d'oracle pour y, nous la remplaçons par le sous-circuit nouvellement créé. Au pire, chaque fil de C^y est une entrée d'une porte d'oracle pour y dans C^y, donc nous faisons m/n_1 appels à l'oracle. Cela nous donne la taille suivante pour C :

$$O(m + \frac{m}{n_1} 2^{\log n_1} \log\log n_1) = O(m + m \log\log n_1) \leq O(m \log\log m)$$

■

En utilisant le lemme que nous venons de montrer, on justifie l'existence d'un circuit C_n'', sans oracle, de taille $O(t(|x|)^2 + t(|x|)^2 \log\log m)$ simulant $C_n'^{d_x}$ Cela nous permet de majorer la complexité en circuit de x :

$$\mathrm{Size}(x) \leq |C_n''| = O(t(|x|)^2(1 + \log\log m)) \leq O(m^2 \log\log m) = O(\mathrm{KT}(x)^2 \log\log \mathrm{KT}(x)).$$

2. Soit x une chaîne de taille n telle que $\text{Size}(x) = m$, c'est-à-dire qu'il existe un circuit C de taille m calculant f_x. Donc, il existe un encodage d_C de C de taille $O(m \log m)$ qui, donné en tant qu'oracle à la machine M de la proposition (2.6), permet de calculer C en temps $O(m^2 \log m)$. En modifiant légèrement le code de M, nous pouvons construire M' s'exécutant aussi en temps $O(m^2 \log m)$, mais prenant maintenant en argument aussi b comme attendu par la définition de KT :

```
val dc : bool array (* oracle fini donnant un code pour C *)
val m : bool array -> bool (* machine simulant efficacement le circuit
    C *)

let m' i b =
  if b = * then false
  else m i = b
```

Puisqu'il existe des machines universelles (au sens de la définition 2.1), et puisque M' existe, il existe U vérifiant la définition de KT.

On obtient ainsi une majoration de $\text{KT}(x)$:

$$\text{KT}(x) \leq |d_c| + t = O(m \log m + m^2 \log m) = O(\text{Size}(x)^2 \log \text{Size}(x)).$$ ∎

En relativisant aux oracles infinis la preuve du théorème 3.16, nous obtenons le résultat suivant :

Théorème 3.18. *Pour toute chaîne binaire x et tout oracle infini A, $\text{Size}^A(x)$ et $KT^A(x)$ sont liés polynomialement. Plus précisément :*

1. $\text{Size}^A(x) = O(\text{KT}^A(x)^2 \log \log \text{KT}^A(x))$
2. $\text{KT}^A(x) = O(\text{Size}^A(x)^2 (\log \text{Size}^A(x) + \log \log |x|))$

Remarque

En toute rigueur, dans le théorème précédent et le suivant, un facteur "x" apparaît alors que nous avons des polynômes en $\text{C}(x)$ ou $\text{KT}(x)$: ça n'est pas la même variable. Mais, puisque c'est une longueur (donc en $\log(x)$) appliquée au deuxième log itéré, on suppose ces facteurs négligeables : "on raisonne à log près".

Attelons-nous maintenant à la deuxième égalité : $C = \text{KT}^H$.

Théorème 3.19 ($C = \text{KT}^H$). *Notons H le problème de l'arrêt. Alors, les mesures KT^H et C sont liées polynomialement modulo des facteurs $\log |x|$, c'est-à-dire que pour toute chaîne x :*

1. *$C(x) = O(\text{KT}^H(x))$, c'est une propriété de force sur $C(x)$, qui ne doit pas être bien plus grand que $KT^H(x)$.*
2. $\text{KT}^H(x) = O((C(x) + \log |x|) \log(C(x) + \log |x|))$

Remarque

Pour être plus fidèles au papier d'Allender, nous montrons les résultats avec une chaîne x, mais nous aurions aussi pu nous concentrer sur la forme f_x.

Idée de preuve

Pour le premier sens, on peut majorer $C(x)$ par $O(m) = O(t + |d|)$, car $t = O(m)$ nous permet de construire un descripteur incluant une chaîne de taille $O(m)$ constituant la part utilisée de l'oracle dans l'exécution de U, la machine calculant KT^H en temps t.

Pour le sens réciproque, on utilise le même descripteur d_x pour $\mathrm{KT}(x)$ que celui donné par $\mathrm{C}(x)$, et montrons qu'il existe une machine M qui, en utilisant un oracle pour l'arrêt, calcul en temps $O(\mathrm{KT}(x) + \log|x|)$ le i-ème bit de x : l'idée est d'employer la réduction $\left|\begin{array}{ccc} A & \longrightarrow & H \\ x & \longmapsto & \langle\langle M'\rangle, x\rangle \end{array}\right.$ où A est l'ensemble des descripteurs permettant de calculer le i-ème bit de x et M' est une énumération de H. La réduction est possible car A est c.e et H est c.e-difficile. Nous concluons en utilisant l'existence d'une machine universelle simulant en particulier M en temps quasi-linéaire.

Preuve. 1. Soit x une chaîne binaire de longueur n telle que $\mathrm{KT}^H(x) = m$. Par définition, il existe un descripteur d_x et un temps t tels que $|d_x| + t = m$, et tel que toute machine universelle U ayant accès à l'oracle infini H et l'oracle fini d_x décide en temps t le i-ème bit de x. Pour montrer que $\mathrm{C}(x) = O((\mathrm{KT}^H(x)))$, donnons l'existence d'un descripteur d'_x de taille linéaire en m permettant de décider les bits de x, c'est-à-dire tel que :

$$\exists d'_x,\ \forall U' \text{ universelle },\ \forall b,\ \forall i \leq |x| + 1,\ \mathrm{U}'^{d'_x}(i, b) = 1 \text{ ssi } x_i = b.$$

Nous savons que la machine U^{H,d_x} s'exécute en temps au plus t, donc la portion d'oracle de H utilisée lors des exécutions de U est un sous-ensemble de H', l'ensemble des mots d'oracle de longueur au plus t :

$$H' = \{n \in H : |n| \leq t\},\ \text{où } |n| \text{ est la taille de } n \text{ en binaire.}$$

En effet, d'après la définition du modèle de la machine de Turing à oracle, nous devons au moins écrire sur la bande d'oracle les requêtes que nous désirons effectuer, et chaque bit à écrire sur la bande requiert une étape de calcul, ne nous laissant pas le temps d'effectuer de plus grosses requêtes.
Avec une machine M ayant comme oracles finis h' (un mot binaire codant H'), d_x et t nous pouvons donc simuler U^{H,d_x} sur toute entrée (i, b). Ainsi, il suffit de choisir :

$$d'_x = (h', d_x, t).$$

On peut vérifier que $|d'_x| = O(1) + O(m)$. On conclut que $C(x) = O(m)$.

2. Soit x une chaîne de taille n telle que $\mathrm{C}(x) = m$. Il existe un descripteur d_x de taille m tel que pour toute machine universelle U, $U^{d_x}(i, b)$ décide le i-ème bit de x. L'ensemble suivant est récursivement énumérable :

$$A = \{\langle d, i, b\rangle : U^d(i, b) \downarrow = 1\}.$$

Donc, il existe un code de machine e tel que $\Phi_e(x) = \begin{cases} 1 \text{ si } x \in A \\ \perp \text{ sinon} \end{cases}$. Et, la fonction suivante est calculable en temps linéaire en $|x| + |d_x|$, avec un oracle infini pour H et l'oracle fini d_x :

$$f : (i, b) \mapsto \mathrm{H}(\langle e, \langle d_x, i, b\rangle\rangle)$$

En effet, il suffit d'écrire un mot de longueur $O(1) + m + n + O(1)$ sur la bande de requête à H, puis d'écrire le résultat sur la bande de sortie.

Le temps d'exécution est donc $O(m+\log|x|)$ car $i \leq |x|$ est écrit en binaire. Puis, l'existence d'une machine universelle nous affirme que le temps d'exécution sur une machine universelle U est en :

$$O((m+\log|x|)\log(m+\log|x|)$$

ce qui nous permet de conclure. ■

Des théorèmes 3.18 et 3.19 nous déduisons le corollaire suivant :

Corollaire 3.20 ($\mathbf{C = Size^H}$, *Allender, [1]*). *La complexité en circuit avec oracle pour l'arrêt et la complexité de Kolmogorov sont liées polynomialement , ce qui se note* $C(x) =_p \text{Size}^H(x)$. ♢

Théorème 3.21. *Soit* $A \geq_T \emptyset'$. *Alors* $\text{Size}^A \leq_p \text{Size}^{\emptyset'}$. ♢

Théorème 3.22. *Pour tout oracle A avec la puissance de l'arrêt (pour la réduction many-one), c'est-à-dire tel que $H \leq_m A$, on a :*

$$C(x) =_p \text{Size}^A(x), \text{ pour toute chaîne } x.$$

Preuve. En répétant la preuve du théorème 3.19 (1.) pour un oracle quelconque, nous pouvons montrer que plus généralement, pour toute chaîne x et oracle A,

$$C(x) = O(\text{KT}^A(x)).$$

On en déduit (puisque $\text{KT}^A = \text{Size}^A$) : $C \leq_p \text{Size}^A$.

Pour le deuxième sens il faut montrer que $C \geq_p \text{Size}^A$, c'est-à-dire que l'oracle A permet de décompresser au moins aussi bien que C. Puisque nous avons déjà montré que $C \geq_p \text{Size}^H$ (corollaire 3.20), il suffit de montrer que sous réserve que $A \geq_T H$, alors $\text{Size}^H \geq_p \text{Size}^A$. Cela revient à montrer $\text{KT}^H \geq_p \text{KT}^A$. Soient d_H et t_H respectivement un descripteur et un temps d'exécution tels que $\text{KT}^H(x) = |d_H| + t_H$ pour une chaîne x fixée.

Soit f une réduction many-one de H vers A. Avec la même réduction que la preuve du deuxième point du théorème 3.19, nous construisons un code simulant H à partir de l'oracle A qui s'exécute en temps quasi-linéaire en la taille des requêtes. Puis, nous remplaçons en dur dans le code d_H les appels à H par ce nouveau code simulant A. C'est une opération syntaxique, donc c'est bien possible, et le code étant fini il n'y a qu'un nombre fini d'occurrences d'appels à H. Finalement nous utilisons le théorème d'existence de machine efficace généralisée pour justifier qu'il existe un code de taille polynomiale en d_H qui s'exécute en temps polynomial en t_H ce qui montre notre deuxième et dernier point. ■

On a vérifié que le résultat d'Allender peut se généraliser aux oracles plus puissants que l'arrêt.

4 Caractérisation de classes non-uniformes et hiérarchie

4.1 Réseau de neurones

Un réseau de neurones récurrent est un autre modèle de calcul construit par "couches". Partant d'une entrée représentée comme un vecteur de nombres, on applique une fonction de saturation σ et ajoutant à chaque nouvelle couche un poids conditionnant l'évolution vers l'état suivant. C'est l'ensemble X auxquels appartiennent les "poids" qui conditionne la puissance du réseau de neurones (rationnels, réels ..).

Notons $\mathcal{M}_{N,M}(X)$ l'ensemble des matrices de taille $n \times m$ à coefficients dans X.

Définition 4.1 (Réseau de neurones à poids dans X). *Soit $N \in \mathbb{N}$. Un réseau de neurones $\mathcal{N}(X)$ de taille N est la donnée d'une matrice carrée de $N \times N$ nombres de X, ainsi que deux matrices colonnes b et c chacune de N nombres de X, c'est-à-dire :*

$$\mathcal{N}(X) = (A, b, c) \in \mathcal{M}_{N,N}(X) \times \mathcal{M}_{N,1}(X)^2. \qquad \circ$$

Les ensembles X couramment utilisés seront des parties de $\mathbb{R}$. Nous étendons la définition aux poids représentés par des chaînes binaires (finies ou infinies) en définissant les choses de la manière suivante. La fonction $\delta_2 : 2^{\leq \mathbb{N}} \to [0, 1]$ est une bijection calculable entre les chaînes binaires et les réels entre 0 et 1. On peut parler indifféremment de poids réels et de poids dans un ensemble de chaînes binaires S, en prenant $X = \{r \in \mathbb{R} : \delta_2^{-1}(\{r\}) \in S\}$ où $\{x\}$ est la partie décimale de r. En effet, la complexité de Kolmogorov de la partie entière d'un réel est une constante (puisqu'elle est constituée d'un nombre fini de digits) et n'ajoute donc pas d'information.

Notation (Fonction de saturation)

Pour tout entier N, on défini la fonction de saturation σ :

$$\sigma : \left| \begin{array}{ccc} \mathcal{M}_{N,1}(\mathbb{Q}) & \longrightarrow & \mathcal{M}_{N,1}([0,1]) \\ (x_i) & \longmapsto & \begin{cases} 0 & \text{if } x_i < 0 \\ x_i & \text{if } 0 \leq x_i \leq 1 \\ 1 & \text{else} \end{cases} \end{array} \right.$$

On va appliquer composante par composante la fonction σ pour construire les différentes configurations (qui sont l'analogue des configurations d'une machine de Turing).

Définition 4.2 (Configurations d'un réseau de neurones). *Soit $x = (x_0, \ldots, x_{n-1}) \in \mathbb{Q}^n$ un vecteur de n rationnels, appelé "entrée", et $\mathcal{N}_N$ un réseau de neurone. On définit la fonction d'état de $\mathcal{N}_N$ par l'équation de récurrence suivante :*

$$\begin{cases} q(0) = 0_N \\ q(t+1) = \sigma(Aq(n) + x_{t-1}b + c) \end{cases}$$

avec la convention que $x_t = 0$ pour $t \geq n$. La suite des configurations du réseau sur l'entrée x est alors définie par $(q_i)_{i \in \mathbb{N}}$. $\circ$

De la même manière que les machines et les circuits, les réseau de neurones calculent une fonction.

Définition 4.3 (fonction partielle calculée par un réseau de neurones). *Un réseau de neurones $\mathcal{N}$ est défini pour une entrée $x \in \{0,1\}^n$ si la suite des configurations de $\mathcal{N}$ vérifie :*

$$(q_i) = 0^k 1^y 0^{\mathbb{N}}, \text{ pour } y, k \in \mathbb{N}.$$

On note alors $\Phi_{\mathcal{N}}(n) \downarrow = y$. Dans le cas contraire, on note $\Phi_{\mathcal{N}}(n) \uparrow$. ∘

Puis, on peut montrer que pour l'ensemble de poids $X = \mathbb{Q}$, les modèles de la machine de Turing et du réseau de neurones sont équivalents.

Théorème 4.4 (Équivalence MT et NN à poids rationnels, Balcázar [2]). *Soit $f :\subseteq 2^{<\mathbb{N}} \to 2^{<\mathbb{N}}$ une fonction partielle sur des chaînes binaires. Alors,*

$$\exists e \in \mathbb{N},\ \Phi_e(x) = f(x) \iff \exists \mathcal{N},\ \Phi_{\mathcal{N}}(x) = f(x),\ \forall x \in \operatorname{dom} f.$$

De plus, si f est calculable en temps $T(x)$ avec une machine de Turing, alors elle est calculable en temps $O(T(x) + n)$ avec un réseau. ♢

En autorisant l'ensemble de poids $X = \mathbb{R}$, nous définissons un modèle strictement plus puissant que celui des machines de Turing. Schématiquement, on peut remarquer que des poids rationnels donnent une information finie (sur un alphabet fini), alors qu'un réel quelconque ne peut *a priori* se représenter qu'avec un nombre infini de symboles, il ne peut donc pas être codé en dur dans le code d'une machine de Turing qui doit être fini, alors qu'un rationnel peut l'être.

Théorème 4.5 (Balcázar [2]). *La classe des langages acceptés en temps polynomial par un réseau de neurones avec poids réels est*

$$\mathrm{P} \,/\, \mathrm{poly},$$

où poly *est l'ensemble des fonctions $\mathbb{N} \to \mathbb{N}$ bornées par une fonction polynomiale.* ♢

Exemple 3. *Tout rationnel a une complexité (K') finie.*

4.2 Classes non-uniformes et poids de Kolmogorov

Une classe de fonctions $\mathcal{F}$ est close par O(.) si $\forall (f,g) \in \mathbb{N}^{\mathbb{N}},\ f \in \mathcal{F} \wedge g = \mathrm{O}(f) \implies g \in \mathcal{F}$.

Pref-C/H est défini comme C/H, mais où chaque conseil de taille n est préfixe des conseils de taille $m \geq n$. Si C/H est clos par O(.), alors $C/H =$ Pref-C/H.

Définition 4.6 (bornes de conseil raisonnables, Balcázar [2]). *Une classe de fonctions $\mathcal{F}$ forme des bornes de conseil raisonnables si :*

- *$\mathcal{F} \subseteq$ poly,*
- *elle est close par* O(.),
- *elle est close par borne polynomiale, c'est-à-dire :*

$$\forall (p,f) \in \mathrm{poly} \times \mathcal{F},\ \exists g \in \mathcal{F} \cap \mathrm{FP},\ f \circ p \leq g,$$

où FP *est l'ensemble des fonctions discrètes ($\in \mathbb{N}^{\mathbb{N}}$) calculables en temps polynomial.* ∘

Lemme 4.7 (Clôture par mélange, Balcázar [2]). *Soit $\mathcal{F}$ une fonction close par* O(.). *Alors, l'ensemble $S = K'[\mathcal{F}, \mathrm{poly}]$ est "clos par mélange" : pour tout nombre fini de mots finis ou infinis de S, l'énumération caractère par caractère alternant entre ces mots de manière*

ordonnée est aussi dans S. Formellement :

$$\forall(\alpha_i) \in S^k, \alpha_1[1]\dots\alpha_k[1]\cdots\alpha_1[n]\dots\alpha_k[n]\cdots \in S.$$

Pour deux ensembles de réels A et B, on note $A+B = \{a+B : a \in A \wedge b \in B\}$. Un ensemble est dit unaire si c'est un ensemble de mots sur l'alphabet $\{0\}$. De plus, pour un ensemble de mots (éventuellement infini) A, on définit χ_A sa chaîne caractéristique : $\chi_A[i] = 1$ si et seulement si le i-ème mot dans l'ordre lexicographique appartient à A.

Théorème 4.8 ($\mathbf{TM}^{\mathcal{T}_S} =_p \boldsymbol{\mathcal{N}_{S+\mathbb{Q}}}$, Balcàzar [2]). *Soit $S \subseteq 2^{\mathbb{N}}$ un ensemble de chaînes infinis clos par mélange et contenant $1^{\mathbb{N}}$. Soit $\mathcal{T}_S$ la classe des ensembles unaires engendrés par S :*

$$\mathcal{T}_S = \{T \subseteq 1^{\mathbb{N}} : \chi_T \in S\}.$$

Alors, les temps de calcul dans un réseaux de neurones avec poids dans $S+\mathbb{Q}$ et dans une machine de Turing qui consulte des oracles dans $\mathcal{T}$ sont polynomialement liés. ♢

Définissons un résultat primordial pour construire la hiérarchie infinie de la prochaine sous-section. On autorise des poids appartenant à l'ensemble $\mathrm{K}'[\mathcal{F}, \mathrm{poly}]$, c'est-à-dire d'après la définition de K' (3.15), l'ensemble des chaînes calculables bit-à-bit en temps polynomial à l'aide d'un conseil dont la taille appartient à la classe $\mathcal{F}$.

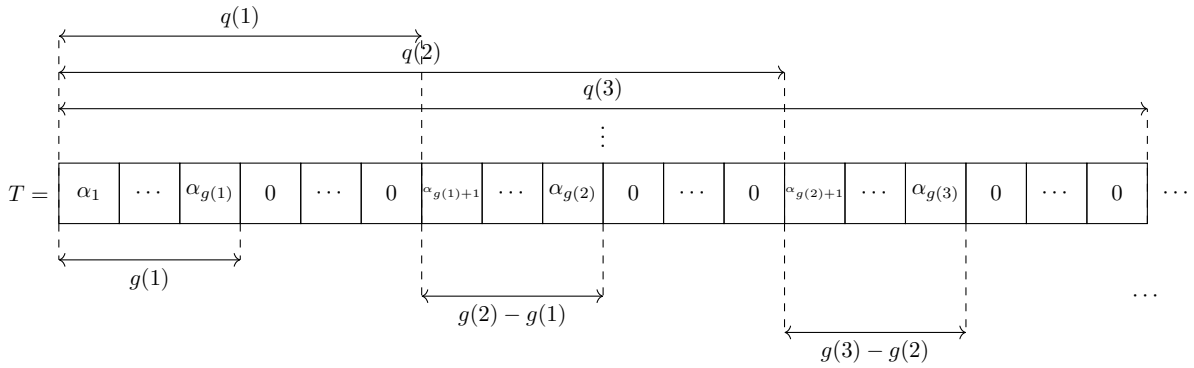
Théorème 4.9 ($\mathbf{P}/\mathcal{F} = \boldsymbol{\mathcal{N}_{\mathrm{K}'[\mathcal{F},\mathrm{poly}]}}$[FP], adapté de Balcázar, 6.2 [2]). *Soit $\mathcal{F}$ une classe de bornes de conseil raisonnables. Alors, la classe $\mathrm{P}/\mathcal{F}$ est exactement la classe des langages reconnus en temps polynomial par un réseau de neurones avec des poids dans $\mathrm{K}'[\mathcal{F}, \mathrm{poly}]$.* ♢

Preuve. Puisque $\mathcal{F}$ est close par O(.), $P/\mathcal{F} =$ Pref-$P/\mathcal{F}$ et toutes les constantes sont dans $\mathcal{F}$. Or, tout rationnel a une complexité finie (voir exemple 3). Donc, $\mathbb{Q} \cup \{1^{\mathbb{N}}\} \subseteq \mathrm{K}'[\mathcal{F}, \mathrm{poly}]$. En appliquant le théorème 4.8, nous nous ramenons à montrer que Pref-$P/\mathcal{F}$ est exactement la classe $\mathrm{TM}^{\mathcal{T}_{\mathrm{K}'[\mathcal{F},\mathrm{poly}]}}[\mathrm{poly}]$.

(Pref-$P/\mathcal{F} \subseteq \mathrm{TM}^{\mathcal{T}_{\mathrm{K}'[\mathcal{F},\mathrm{poly}]}}[\mathrm{FP}]$) : Soit $A \in$ Pref-P$/\mathcal{F}$. Il existe une chaîne infinie de conseils α et une machine M calculant A en temps polynomial à l'aide du préfixe $\alpha_{|f(|x|)}$ pour une certaine fonction $f \in \mathcal{F}$ et pour tout $x \in \mathbb{N}$. En effet, par définitions de Pref-P$/\mathcal{F}$, les conseils (a_n) sont préfixes les uns-des-autres ($a_i \prec a_j,\ \forall i \geq 0,\ j \geq i$), donc on peut construire la limite de ces conseils : $\alpha = \lim_{n\to\infty}(a_n)$.
Notons $g \in \mathcal{F} \cap \mathrm{FP}$ telle que $f \leq g$, dont l'existence est garantie par clôture polynomiale de $\mathcal{F}$. Puisque $g \in \mathrm{FP}$, il existe aussi une fonction polynomiale p telle que $f \leq g \leq p$. En posant

$$q(t) = (t+1)p(t),$$

nous pouvons définir l'oracle T comme suit :



Intuitivement, pour que nous puissions remplir l'écart entre $g(t+1)-g(t)$ et $q(t+1)-q(t)$, il faut que $g(t+1)-g(t) \leq q(t+1)-q(t)$. Or, c'est vérifié car

$$g(t+1)-g(t) \leq g(t+1) \leq p(t+1) \leq q(t+1)-q(t).$$

Vérifions que $\chi_T \in \mathrm{K}'[\mathcal{F}, \mathrm{poly}]$. En prenant $g \in \mathcal{F}$ précédemment défini, l'algorithme suivant permet de calculer les $q(m) \leq m$ premiers bits de χ_T à partir de m et des $g(m)$ premiers bits de α en temps polynomial :

```
let i_g = ref 1
and i_t = ref 1 in
for j = 1 to m do
  while !i_g ≤ g.(j) do
    print_int α.(g.(!i_g));
    incr i_g;
    incr i_t
  done;
  while !i_t ≤ q,(j) do
    print_int 0
  done
done
```

Nous pouvons nous convaincre qu'il est encore plus simple (en complexité temporelle) de construire une machine calculant seulement le i-ème bit de χ_T, donc applique le théorème de simulation d'une machine par une famille de circuits nous permet de conclure.

Puisque K' vérifie les mêmes hypothèses que K nécessaires à la preuve de l'inclusion réciproque (nous venons de montrer le sens "difficile" — *voir le chapitre "mathématiques à rebours" de [4] pour donner un sens plus précis à ce mot* —), nous aboutissons au résultat $\mathrm{P}/\mathcal{F} = \mathcal{N}_{\mathrm{K}'[\mathcal{F},\mathrm{poly}]}[\mathrm{FP}]$. ■

4.3 Théorème de hiérarchie

Notation (Ordre partiel $\prec$ sur les classes de fonctions, Balcázar [2])

On note $\mathcal{F} \prec \mathcal{G}$ s'il existe une fonction croissante $s(n) \in \mathcal{G}$ calculable en temps polynomial en n telle que $s(n) = \mathrm{o}(s(n))$ et pour tout polynôme p et fonction $r \in \mathcal{F}$, $r(p(n)) = (s(n))$.

Lemme 4.10. *Soit (θ_i) la suite de classes de fonctions :*

$$\theta_i = \{f : \mathbb{N}^{\mathbb{N}} : \exists c \in \mathbb{N}^*,\ f(n) \leq c \times (\log n)^i,\ \forall n\}.$$

Alors, pour tout $i \in \mathbb{N}$, $\theta_i \prec \theta_{i+1}$. ♢

Preuve. On raisonne à constante multiplicative près. Soit $i \in \mathbb{N}$. On définit $s(n) = (\log n)^{i+1} = \mathrm{o}(n) \in \theta_{i+1}$. Soient $r \in \theta_i$ et p un fonction polynomiale quelconque. Alors $r(n) \leq (\log n)^i$ et $p(n) \leq n^k$ pour un certain $k \in \mathbb{N}$. Montrons que $r(p(n)) = \mathrm{o}(s(n))$ sachant $r(p(n)) \leq (\log(n^k))^i$. On a :

$$\frac{(\log(n^k))^i}{(\log n)^{i+1}} = \frac{1}{\log n} \to +\infty$$

car :

$$\log(n^k) = \frac{\ln(e^{k \log n})}{\ln 10} = O(\log n)$$

ce qui conclut la preuve. ■

On note $\mathcal{P}(A)$ la classe des langages qui peuvent être décidés en temps polynomial par une machine de Turing avec oracle pour A, et on rappelle que χ_A dénote la chaîne caractéristique de l'ensemble de mots A. On note $\mathcal{T}_{\mathcal{F}}$ la famille des ensembles unaires T ($\subseteq \{0\}^{\mathbb{N}}$) tels que $\chi_T \in \mathrm{K}'[\mathcal{F}, \mathrm{poly}]$.

Théorème 4.11 (Balcázar, [2]). *Soient $\mathcal{F}$ et $\mathcal{G}$ des classes de fonctions non vides telles que $\mathcal{F} \prec \mathcal{G}$. Alors, $\mathcal{P}(\mathcal{T}_{\mathcal{F}})$ est strictement incluse dans $\mathcal{P}(\mathcal{T}_{\mathcal{G}})$.* ♢

Cela signifie que l'oracle pour les ensembles unaires qui sont de complexité $\mathrm{K}'[\mathcal{G}, \mathrm{poly}]$ est strictement plus puissant que celui utilisant des tailles de conseil dans $\mathcal{F}$.

En combinant ce résultat avec le théorème 4.8 établissant un lien entre la calculabilité avec les machines de Turing avec oracle unaire et les réseau de neurones, nous aboutissons au théorème suivant :

Théorème 4.12 (Théorème de Hiérarchie, adapté de Balcázar, [2]). *Soient $\mathcal{F}$ et $\mathcal{G}$ deux classes closes par $\mathrm{O}(.)$ telles que $\mathcal{F} \prec \mathcal{G}$. On note $\mathcal{N}_{\mathrm{K}'[\mathcal{F},\mathrm{poly}]}$ la classe des langages acceptés par un réseau calculant en temps polynomial à l'aide de poids dans $\mathrm{K}'[\mathcal{F}, \mathrm{poly}]$, et la même chose pour $\mathcal{G}$. Alors,*

$$\mathcal{N}_{\mathrm{K}'[\mathcal{F},\mathrm{poly}]} \subsetneq \mathcal{N}_{\mathrm{K}'[\mathcal{G},\mathrm{poly}]}.$$

En appliquant le théorème 4.9, nous obtenons :

Théorème 4.13. *Pour tout $k > 0$, $\mathrm{P}/\log^k = \mathcal{N}_{\mathrm{K}'[\log^k,\mathrm{poly}]}$ et, l'inclusion suivante est stricte :*

$$\mathrm{P}/\log^k \subseteq \mathrm{P}/\log^{k+1}.$$

Puisque notre K' est définit à l'aide de circuits (à la différence de K définit à l'aide de machines), on confirme l'intuition d'Oliver, à savoir que les classes non-uniformes de l'article de Balcázar cachent bien des circuits booléens. Mais pourrions-nous aller plus loin en utilisant le lien $\mathrm{C} = \mathrm{Size}^H$?

Conclusion

En étudiant la nature des circuits booléens, différentes complexités de Kolmogorov et ajouts d'oracles de puissance variable pour ces deux modèles, nous avons abouti au principal résultat de ce stage de recherche : le théorème 4.13. Ce dernier caractérise de manière naturelle les classes non-uniformes (c'est-à-dire, avec conseil) de la forme $\mathrm{P}/\log^k$. Pour cela, il faut utiliser le modèle du réseau de neurones récurrent (présenté en section 4), dont les poids sont définis par la complexité de Kolmogorov K' (3.15) exprimée en terme de circuit booléen bit-à-bit.

Ouverture

Nous avons montré que pour tout oracle A avec la puissance de l'arrêt ($A \geq_T \emptyset'$), $\mathrm{C} = \mathrm{Size}^A$. Mais alors, en notant $\emptyset^{(n)}$ le n-ème saut Turing de l'ensemble calculable, cela implique $\mathrm{C} = \mathrm{C}^{\emptyset^{(n)}}$, pour tout $n > 0$. On ne semble pas pouvoir construire de hiérarchie infinie à l'aide d'oracle plus puissant que l'arrêt. Par contre, on ne sait pas infirmer ou démontrer l'énoncé suivant :

$$\text{Soit } A \text{ un oracle, est-ce que } A <_T \emptyset' \text{ implique } \mathrm{KT}^A >_p \mathrm{KT}^{\emptyset'}.$$

C'est-à-dire, a-t'on besoin d'un oracle aussi puissant que l'arrêt pour compresser aussi bien avec la mesure KT qu'avec la mesure C ? En fait, nous ne savons même pas montrer que $\mathrm{KT} <_p \mathrm{KT}^{\emptyset'}$, bien que nous ayons des arguments nous laissant penser que c'est le cas. Il y a encore beaucoup de travail pour parvenir à séparer strictement les différentes complexités de Kolmogorov présentées dans [1], en étudiant comment lier une complexité à une autre en augmentant sa puissance calculatoire (typiquement, avoir $\mu_1 <_p \mu_2$ mais connaître un oracle A tel que $\mu_1 =_p \mu_2^A$).

Références

[1] Eric Allender, Harry Buhrman, Michal Koucký, Dieter Melkebeek, and Detlef Ronneburger. Power from random strings. *SIAM J. Comput.*, 35 :1467–1493, 01 2006.

[2] José Balcázar, Ricard Gavaldà, and Hava Siegelmann. Computational power of neural networks : A characterization in terms of kolmogorov complexity. *Information Theory, IEEE Transactions on*, 43 :1175 – 1183, 08 1997.

[3] Sylvain Perifel. *Complexité algorithmique*. Références sciences. Ellipses, 2014.

[4] Benoît Monin et Ludovic Patey. *Calculabilité*. Calvage et Mounet, 2022.

[5] F. Hennie and Richard Stearns. Two-tape simulation of multitape turing machines. *J. ACM*, 13 :533–546, 01 1966.

[6] Martin Fürer. The tight deterministic time hierarchy. In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing (STOC)*, pages 8–16. ACM, 1982.

[7] Martin Fürer. Data structures for distributed counting. *Journal of Computer and System Sciences*, 28(2) :231–243, 1984.

[8] Leonid A. Levin. Randomness conservation inequalities ; information and independence in mathematical theories. *Information and Control*, 61(1) :15–37, 1984.