Systèmes de numération et régularité des représentations

Michel Rigo

SYSTÈMES DE NUMÉRATION ET RÉGULARITÉ DES REPRÉSENTATIONS

MICHEL RIGO†

RÉSUMÉ. Dans ce survol, nous nous intéressons aux systèmes de numération du point de vue de la théorie des langages formels. On s'attache dès lors aux liens éventuels entre propriétés arithmétiques des nombres et propriétés syntaxiques des représentations.

1. INTRODUCTION

Depuis des siècles, l'homme a appris à compter et à manipuler les nombres (contentons-nous ici des nombres entiers). En particulier, il s'est donné les moyens pour permettre l'écriture de ces nombres. En effet, pouvoir représenter les nombres de manière symbolique sur un papyrus ou sur une feuille de parchemin revêt un intérêt pratique évident : comptage de bétail, recensement d'individus, payement de taxes, etc... (Ces quelques lignes introductives sont loin de constituer un réel aperçu historique des systèmes de numération, le lecteur intéressé pourra consulter l'excellent ouvrage [10].)

De nos jours, les enfants apprennent à compter sur leurs dix doigts à l'école élémentaire. Mais sans contestation possible, c'est le système binaire des ordinateurs qui est le plus utilisé pour représenter les nombres (du moins, pour des raisons de configuration physique, la base choisie de manière interne au sein de la machine est une puissance de deux). L'avènement de l'informatique a élargi ou consolidé de nombreux domaines de recherche en mathématique. Citons par exemple l'analyse numérique, la logique, l'algorithmique, la théorie de la complexité et de la calculabilité ou encore l'étude des langages formels. Les interactions entre mathématique et informatique sont nombreuses, le mathématicien s'inspirant ou généralisant des problèmes d'origine informatique et l'informaticien utilisant et appliquant grandement les méthodes développées par son collègue.

Dans cet exposé, nous nous intéresserons à la théorie des langages formels. Selon le point de vue, certains qualifieront cette branche comme étant de l'informatique (théorique), d'autres comme des mathématiques discrètes — nous ne trancherons pas. De façon rapide, on peut dire qu'on y définit de manière rigoureuse les notions de mot et de langage pour pouvoir ensuite en étudier les propriétés. Ainsi du point de vue de la théorie des langages formels, on s'intéressera principalement aux propriétés syntaxiques de mots.

Mais revenons aux nombres entiers et à leur représentation. Le mathématicien qui s'intéresse aux propriétés des entiers sera très certainement enclin à l'étude de l'arithmétique ou de la théorie des nombres. Nous pouvons cependant faire le

Date: 1er mai 2003.

† Chargé de recherches FNRS.

constat suivant. Si on représente un nombre, par exemple dans le système décimal, ce que l'on obtient est une suite de chiffres qui n'est autre qu'un mot sur l'alphabet $\{0,1,\ldots,9\}$. Ainsi, l'utilisation d'un système de numération comme le système décimal permet de passer d'une représentation symbolique : "l'entité nombre entier", à une représentation concrète : un mot écrit sur un alphabet de chiffres. Dès lors, on pourra s'intéresser aux propriétés arithmétiques des nombres mais aussi aux propriétés syntaxiques de leurs représentations. La question fondamentale étant clairement de déterminer s'il existe un lien entre ces deux types de propriétés. Par exemple, en base 10, le fait pour un entier n d'être divisible par dix se visualise trivialement sur la représentation décimale de n. En effet, il suffit de regarder si le dernier chiffre est ou non zéro. Dans cette situation, il y a donc un lien fort entre une propriété arithmétique des nombres — être divisible par dix — et une propriété syntaxique de leurs représentations — se terminer par zéro. Dans ce papier, nous allons préciser le type de lien que l'on peut espérer obtenir.

2. Un peu de théorie des langages formels

Commençons par quelques définitions élémentaires. Un *alphabet* est un ensemble fini. Par exemple $\{a,b\}$, $\{0,1,2\}$ ou $\{\uparrow,\downarrow,\rightarrow,\leftarrow\}$ sont des alphabets. Un mot sur l'alphabet Σ est une suite finie et ordonnée d'éléments de Σ; le mot vide (le seul mot de longueur nulle) se notera ε. Si $u = u_1 \cdots u_k$ et $v = v_1 \cdots v_\ell$ sont deux mots sur l'alphabet Σ (les u_i et v_i étant des éléments de Σ), la *concaténation* des mots u et v est le mot $w = w_1 \cdots w_{k+\ell}$ où $w_i = u_i$ pour $i = 1,\ldots,k$ et $w_{k+i} = v_i$ pour $i = 1,\ldots,\ell$. En général, on notera simplement uv la concaténation des mots u et v. Enfin, on dénote par $|u|$, la longueur du mot u. Par exemple, la concaténation de "*bon*" et "*jour*" est le mot "*bonjour*" qui est de longueur 7.

Un *langage* est tout simplement un ensemble de mots. L'ensemble de tous les mots sur Σ est noté Σ^*. Un langage est donc une partie de Σ^*. Autrement dit, Σ^* est le monoïde libre généré par Σ pour l'opération de concaténation des mots et ε en est le neutre. Un langage étant un ensemble, on peut appliquer aux langages les opérations booléennes habituelles et ainsi définir l'union ou l'intersection de deux langages (idem pour le complémentaire ou encore la différence symétrique). On peut aussi définir la concaténation de deux langages L et M comme étant le langage

$$LM = \{uv \mid u \in L,\ v \in M\}.$$

En particulier, la concaténation de L avec lui-même sera noté L^2 et on pourra définir pour tout $n > 0$, le langage L^n constitué des mots obtenus en concaténant n mots de L. En particulier, si on pose $L^0 = \{\varepsilon\}$, alors on définit *l'étoile de Kleene* d'un langage L comme

$$L^* = \bigcup_{i \geq 0} L^i.$$

Ainsi, par définition, L^* contient exactement les mots obtenus en concaténant un nombre arbitraire de mots de L. Par exemple, si L est le langage fini $\{ab, ba\}$, alors L^* contient une infinité d'éléments dont

$$\varepsilon,\ ab,\ ba,\ abab,\ abba,\ baab,\ baba,\ ababab,\ ababba,\ abbaab,\ \ldots$$

2.1. La hiérarchie de Chomsky.

Sans entrer dans les détails, on peut imaginer que certains ensembles de mots sont plus "simples" que d'autres. Ainsi le langage L_1 sur $\{a,b\}$ des mots comprenant un nombre pair de a est sans doute plus "simple"

à reconnaître que le langage L_2 des mots comprenant deux fois plus de a que de b. Et que dire du langage L_4 des mots comprenant un nombre premier de a. On pourrait définir le terme "simple" en décrivant un algorithme à mettre en oeuvre pour reconnaître exactement les mots d'un langage donné. Ainsi, pour tester si un mot appartient au langage L_1 il suffit de disposer d'une mémoire bornée à deux états (testant la parité du nombre de a dans le mot fourni à l'algorithme). Par contre pour le langage L_2, une mémoire non bornée est nécessaire (car les mots à tester peuvent être arbitrairement longs et il faut donc pouvoir mémoriser le nombre de a et de b rencontrés par exemple à l'aide d'une pile). Enfin, tester l'appartenance d'un mot à L_4 requiert clairement des calculs plus compliqués que dans les deux premiers exemples (tests de primalité). C'est à partir de ces constatations que l'on peut définir rigoureusement un classement des langages suivant leur complexité (i.e., suivant la complexité de l'algorithme reconnaissant exactement les mots du langage) :

(1) Les langages réguliers

(2) Les langages algébriques (ou hors-contexte)

(3) Les langages "context-sensitive"

(4) Les langages décidés par machine de Turing

Dans la suite, nous nous intéresserons uniquement aux langages les plus simples, à savoir les langages réguliers. (Le lecteur intéressé par les autres classes pourra par exemple consulter [16]. Noter à titre indicatif que le langage L_i introduit précédemment appartient à la classe (i), pour $i = 1, 2, 4$.)

2.2. **Deux caractérisations des langages réguliers.** Notre but est ici de définir précisément la notion de langage régulier. (Pour plus de détails, voir par exemple [5, 16, 17].) Pour aller au plus rapide, nous donnons ci-dessous une caractérisation des langages réguliers sur un alphabet Σ.

Proposition : *L'ensemble des langages réguliers sur un alphabet Σ est la plus petite famille de langages contenant $\emptyset$, $\{\varepsilon\}$, $\{\sigma\}$ pour tout $\sigma \in \Sigma$ et qui est stable pour l'union, la concaténation et l'étoile de Kleene.*

Cette proposition montre que les langages réguliers sont obtenus à partir des langages finis en appliquant un nombre fini de fois les opérations d'union, de concaténation et d'étoile de Kleene. Par exemple, le langage $\{ab, b\}^*\{b, baa\} \cup \{aaa\}^*$ est régulier. Ce langage contient les mots commençant par un nombre arbitraire de copies de ab et/ou de b et se terminant par b ou baa ainsi que les mots ne comprenant que des a en nombre multiple de 3.

Nous avons dit plus haut que l'algorithme à mettre en oeuvre pour reconnaître exactement les mots d'un langage régulier donné était particulièrement simple. Cet algorithme qui ne nécessite qu'une mémoire finie se visualise en termes d'automate fini. Commençons par donner un exemple d'une telle machine :

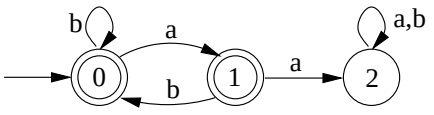

FIG. 1. Un automate fini déterministe.

Le rôle d'un automate est d'accepter ou non les mots qu'on lui fournit. Il lit les lettres du mot donné en entrée, une par une, de gauche à droite et en commençant par la lettre la plus à gauche. La lecture d'un mot démarre dans l'*état initial* de l'automate marqué d'une flèche entrante sans label (ici l'état 0). Ensuite, on se déplace dans le graphe en tenant compte, à chaque étape, de la lettre lue. Par exemple, le mot *abba* fournit le chemin suivant dans l'automate :

$$0 \xrightarrow{a} 1 \xrightarrow{b} 0 \xrightarrow{b} 0 \xrightarrow{a} 1$$

et pour le mot *baab* :

$$0 \xrightarrow{b} 0 \xrightarrow{a} 1 \xrightarrow{a} 2 \xrightarrow{b} 2.$$

Dans l'automate représenté sur la Figure 1, on note que les états 0 et 1 sont marqués d'un double cercle. On dira que ces états sont *finals*. Un mot est accepté par un automate si la lecture de ce mot fournit un chemin qui se termine dans un tel état. Ainsi, le mot *abba* est accepté (la lecture se termine en 1) alors que *baab* ne l'est pas (la lecture se termine en 2). On peut se convaincre assez facilement que cet automate accepte exactement les mots sur l'alphabet $\{a, b\}$ ne comprenant pas deux a consécutifs.

De manière formelle, un *automate fini déterministe* sur un alphabet Σ est la donnée d'un quintuple $(Q, q_0, F, \Sigma, \delta)$ où Q est un ensemble fini d'états, $q_0 \in Q$ est l'état initial, $F \subseteq Q$ est l'ensemble des états finals et $\delta : Q \times \Sigma \to Q$ est la fonction de transition. La fonction δ s'étend de manière naturelle à $Q \times \Sigma^*$ par $\delta(q, \varepsilon) = q$ et $\delta(q, \sigma w) = \delta(\delta(q, \sigma), w)$, $\sigma \in \Sigma$, $w \in \Sigma^*$. Si $\mathcal{A} = (Q, q_0, F, \Sigma, \delta)$ est un automate, l'ensemble des mots acceptés par $\mathcal{A}$ est

$$L(\mathcal{A}) = \{w \in \Sigma^* \mid \delta(q, w) \in F\}.$$

Comme le stipule le résultat suivant, les automates finis déterministes sont exactement les accepteurs des langages réguliers.

Théorème : Un langage est régulier si et seulement si il est accepté par un automate fini déterministe.

Les automates finis et les langages réguliers trouvent de nombreuses applications en informatique. Par exemple, pour rechercher efficacement un mot dans un texte, votre traitement de textes favori recourt plus que certainement à la construction préalable d'un automate : un état final étant atteint lorsque le mot recherché a été trouvé dans le texte. De même, les outils tels que `grep` ou `regex` permettant en autre de rechercher des fichiers dont les noms ont une forme prescrite comme "`my*.JPG`", sont aussi basés sur les langages réguliers [7]. Enfin, soulignons que lors de la phase de compilation d'un programme informatique, les mots clés du langage de programmation et les identificateurs sont répérés par le compilateur au moyen d'automates [1]. Notons aussi que la popularité des automates est certainement dûe au fait que la classe de problèmes dont ils relèvent est complétement algorithmisée. En résumé, on peut dire que les automates fournissent un moyen commode de recherche des motifs simples (mais pas nécessairement triviaux !) au sein de mots et de textes.

3. Bases entières, critères de divisibilité

Soit $k \geq 2$. Pour représenter les nombres entiers en base k, on utilise l'algorithme d'Euclide ou algorithme glouton [6]. Cette manière de procéder fournit pour

tout entier n, un mot unique $\rho_k(n)$ sur l'alphabet $\Sigma_k = \{0, \ldots, k-1\}$ qui est la représentation de n en base k. On peut donc dire qu'un système de numération définit une bijection entre $\mathbb{N}$ et les mots d'un certain langage (ici les mots sur $\Sigma_k^* \setminus \{0\}\Sigma_k^*$). Ainsi, si $n \in \mathbb{N} \setminus \{0\}$, alors il existe $\ell \geq 0$, $\alpha_0, \ldots, \alpha_\ell \in \Sigma_k$, $\alpha_\ell > 0$ tels que

$$n = \sum_{i=0}^{\ell} \alpha_i k^i \quad \text{et} \quad \rho_k : n \mapsto \alpha_\ell \cdots \alpha_0.$$

Par exemple,

$$11 = \mathbf{1}.2^3 + \mathbf{0}.2^2 + \mathbf{1}.2 + \mathbf{1}.2^0, \quad \rho_2(11) = 1011$$

et

$$127 = \mathbf{1}.3^4 + \mathbf{1}.3^3 + \mathbf{2}.3^2 + \mathbf{0}.3^1 + \mathbf{1}.3^0, \quad \rho_3(127) = 11201.$$

Si $X \subset \mathbb{N}$ est un ensemble d'entiers, du point de vue de la théorie des langages formels, il est naturel de s'intéresser au langage $\rho_k(X)$ formé des représentations des éléments de X. En particulier, on désire étudier les propriétés syntaxiques des mots constituant le langage $\rho_k(X)$. Nous espérons que la section précédente a convaincu le lecteur que les langages les plus simples syntaxiquement sont les langages réguliers. Il paraît donc raisonnable de déterminer sous quelles conditions $\rho_k(X)$ est un langage régulier. Dans l'affirmative, on dira que X est *k-reconnaissable* ou *reconnaissable en base k*. La motivation principale pour l'étude de ces ensembles reconnaissables étant que les tests à mettre en oeuvre pour vérifier si un mot représente ou non un élément de X sont particulièrement simples puisqu'ils pourront être réalisés par un automate fini (c'est-à dire par un algorithme ne nécessitant qu'une mémoire bornée et dont la complexité est proportionnelle à la longueur du mot fourni en entrée). Considérons quelques exemples.

(a) Les nombres pairs sont 2-reconnaissables. En effet, si un entier n peut s'ecrire sous la forme $\sum_{i=0}^{\ell} \alpha_i 2^i$ avec $\alpha_i \in \{0, 1\}$ alors, puisque toutes les puissances de 2 sont paires à l'exception de 2^0, il est clair qu'un nombre est pair si et seulement si $\alpha_0 = 0$. En d'autres termes, on vérifie la parité d'un nombre écrit en base deux, en regardant uniquement son dernier chiffre. L'automate donné en Figure 2 reconnaît exactement les représentations binaires des nombres pairs.

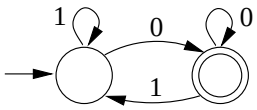

FIG. 2. Automate acceptant le langage $\rho_2(2\mathbb{N})$.

(b) Au vu de ce premier exemple, on peut se dire que les multiples de deux conviennent particulièrement bien à la base deux. Comme nous allons le montrer, les nombres pairs sont aussi 3-reconnaissables. En effet, si n est décomposé sous la forme $\sum_{i=0}^{\ell} \alpha_i 3^i$ avec des coefficients $\alpha_i \in \{0, 1, 2\}$, alors chaque apparition d'un coefficient égal à 0 ou 2 ne change pas la parité de n. Le facteur déterminant est en fait le nombre de coefficients α_i égaux à 1 apparaissant dans la décomposition. En effet, toute puissance de trois est toujours impaire. Donc n est pair si et seulement

si sa représentation ternaire $\rho_3(n)$ contient un nombre pair de 1. Un automate reconnaissant exactement les représentations ternaires[1] des nombres pairs est donné en Figure 3. Ainsi, les premiers nombres pairs écrits en base trois sont

$$2, 11, 20, 22, 101, 110, 112, 121, 200, 202, 211, 220, 222, 1001, \ldots$$

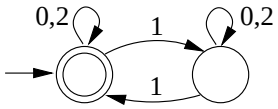

FIG. 3. Automate acceptant le langage $\rho_3(2\mathbb{N})$.

(c) Comme nous allons le voir plus loin, tout critère de divisibilité peut se traduire en termes de propriétés syntaxiques des représentations et ce, qu'elle que soit la base choisie. A titre indicatif, nous allons présenter le critère de divisibilité par 6 pour le système décimal usuel. L'obtention des automates suit toujours un même schéma que nous présentons ici sommairement. Tout d'abord, on remarque que, quel que soit $p \geq 2$, la suite des puissances de 10 modulo p est périodique à partir d'un certain rang. Pour $p = 6$, on a $10^0 \equiv 1 \pmod 6$ et pour tout $r > 0$, $10^r \equiv 4 \pmod 6$. Pour maîtriser ce phénomène de périodicité, on construit un automate lisant cette fois les représentations de *droite à gauche* (i.e., le chiffre de moins haut poids en premier lieu). On ajoute un état pour chaque reste possible modulo p (on inscrira ce reste dans l'état de l'automate pour plus de facilité). Si la suite des puissances de 10 a une partie non périodique de longueur $k > 0$ et une partie périodique de longueur t, l'ajout de p états est réalisé $k + t - 1$ fois. Sur notre exemple, la partie non périodique et la période sont toutes deux de longueur 1, l'automate contiendra donc six états (plus l'état initial). Lorsqu'on lit le chiffre α_0 le plus à droite, partant de l'état initial, on bascule dans un état correspondant à un reste $\alpha_0.1 \pmod 6$. Par contre, quand on lit un chiffre α qui n'est pas le dernier chiffre, α est multiplié par une puissance de 10 d'exposant r strictement positif. Donc se trouvant dans un état de reste m et lisant α, on bascule dans un état de reste $m + 4.\alpha \pmod 6$ (car on a vu que $10^r \equiv 4 \pmod 6$, si $r > 0$). On obtient alors l'automate donné en Figure 4 (nous n'avons représenté que quelques arcs pour ne pas allourdir le croquis). Cette construction permet à chaque étape de savoir à quelle position on se situe dans la lecture d'un mot et donc de choisir le "bon" facteur multiplicatif compte tenu de la périodicité des puissances de dix. Enfin, si nécessaire, il existe des méthodes bien connues pour "retourner" l'automate et alors pouvoir lire les mots de gauche à droite (i.e., en commençant par le chiffre de plus haut poids). La méthode sommairement décrite ci-dessus peut être automatisée (algorithmisée) pour des bases et des progressions arithmétiques arbitraires.

(d) Essayons de construire un automate acceptant les multiples de 7 écrits en base 2. Tout d'abord, on a modulo 7

$$2^0 \equiv 1,\ 2^1 \equiv 2,\ 2^2 \equiv 4,\ 2^3 \equiv 1,\ 2^4 \equiv 2,\ 2^5 \equiv 4,\ 2^6 \equiv 1, \ldots$$

[1]Le lecteur attentif remarquera que cet automate accepte également les représentations débutant par un nombre arbitraire de 0. Autoriser des zéros de tête ne modifie pas la régularité des ensembles de représentations. Par commodité, on autorise parfois de telles représentations.

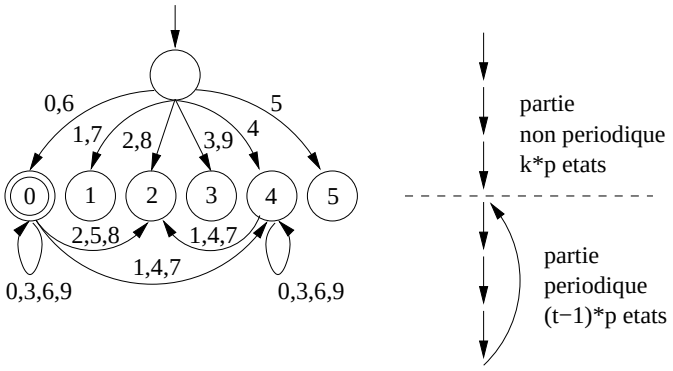

FIG. 4. Automate acceptant les mots de $\rho_{10}(6\mathbb{N})$ lus de droite à gauche.

Ici, la suite est purement périodique de période 3. On va dès lors considérer 18 états de la forme i_j où $0 \leq i \leq 6$ et $j = 1, 2, 4$ (i donne le "reste actuel" et j donne la valeur modulo 7 de la puissance de 2 à considérer ensuite). L'automate comprenant un grand nombre de transitions, il est préférable de donner sa table de transition résumée par :

$$i_1 \xrightarrow{0} i_2, \quad i_2 \xrightarrow{0} i_4, \quad i_4 \xrightarrow{0} i_1$$

$$i_1 \xrightarrow{1} (i+1 \mod 7)_2, \quad i_2 \xrightarrow{1} (i+2 \mod 7)_4, \quad i_4 \xrightarrow{1} (i+4 \mod 7)_1$$

où i prend les valeurs de 0 à 6. L'état initial est bien sûr 0_1 et les états finals sont 0_1, 0_2 et 0_4. Remarquons que cet automate lit toujours les mot de droite à gauche.

Pour aller plus loin, il serait tentant de croire que toute propriété arithmétique peut se traduire par une propriété simple des représentations. Par exemple, pourrait-on distinguer si un nombre est ou non un carré parfait par simple lecture de sa représentation en base 2 ? Si on observe les représentations des premiers carrés parfaits,

$$1, 100, 1001, 10000, 11001, 100100, 110001, 1000000, 1010001, \ldots$$

il n'est pas clair d'y voir apparaître un motif. De même, existerait-il une base "miraculeuse" pour la cryptographie dans laquelle tester si un nombre est premier se ferait simplement par un automate fini ? Dans les deux cas, la réponse est non. Les carrés parfaits et les nombres premiers ne sont jamais reconnaissables et ce qu'elle que soit la base choisie pour les représenter.

On a montré dans les exemples (a) et (b) que les nombres pairs sont 2-reconnaissables et aussi 3-reconnaissables. Les puissances de 2 sont 2-reconnaissables (un mot sur l'alphabet $\{0, 1\}$ représente une puissance de deux s'il est de la forme $10 \cdots 0$) ; par contre, on peut montrer qu'elles ne sont pas 3-reconnaissables (mais pour cela, il faudrait étudier d'un peu plus près les propriétés des langages réguliers). Ce dernier exemple est intriguant et semble donc montrer que la propriété d'être reconnaissable dépend de la base choisie. On a en fait le résultat fondamental suivant qui explique complètement que les critères de divisibilité, et eux seuls, ne dépendent pas de la base choisie.

3.1. **Théorème de Cobham** [4]. Deux entiers k et ℓ sont *multiplicativement indépendants* si la seule solution de $k^m = \ell^n$ est $m = n = 0$. Si k et ℓ sont multiplicativement indépendants, les seuls ensembles simultanément k- et ℓ-reconnaissables sont exactement les unions finies de progressions arithmétiques.

Pour être complet, notons que si k et ℓ sont multiplicativement dépendants, alors $X \subset \mathbb{N}$ est k-reconnaissable si et seulement si il est ℓ-reconnaissable.

En particulier, il est remarquable que tout caractère de divisibilité se traduit, quelle que soit la base choisie, par une propriété syntaxique des représentations. Pour tous $p, q \in \mathbb{N}$, $\rho_k(p\mathbb{N} + q)$ est k-reconnaissable.

La dépendance multiplicative est une relation d'équivalence, le plus petit élément d'une classe d'équivalence est dit *simple*. Au vu de ce qui précède, on peut donc dire qu'il existe trois types d'ensembles d'entiers :

- Les unions finies de progressions arithmétiques sont k-reconnaissables pour toute base k.
- Il existe des ensembles qui sont uniquement k^m-reconnaissables, pour k entier simple fixé, $m \geq 1$. (Par exemple, l'ensemble des puissances de k.)
- Il existe des ensembles qui ne sont reconnaissables dans aucune base entière. (Par exemple, l'ensemble des carrés parfaits [5].)

Dans la littérature, on trouve de nombreuses caractérisations des ensembles k-reconnaissables. L'une d'elles concerne la logique du premier ordre. Si $n > 0$, $V_k(n)$ désigne la plus grande puissance de k qui divise n. On pose $V_k(0) = 1$.

Proposition [2] : Un ensemble $X \subset \mathbb{N}$ est k-reconnaissable si et seulement si il existe une formule $\varphi(n)$ de la logique du premier ordre $\langle \mathbb{N}, +, V_k \rangle$ telle que

$$X = \{n \in \mathbb{N} \mid \langle \mathbb{N}, +, V_k \rangle \models \varphi(n)\}.$$

4. Comment généraliser la base entière

On pourrait estimer que le théorème de Cobham clôt le débat sur les ensembles reconnaissables. Bien au contraire, tout d'abord la question du caractère reconnaissable peut être traîtée dans une plus grande généralité en considérant d'autres méthodes de représentation. Par exemple, on peut se demander s'il n'existerait pas un système de numération suffisamment simple (mais différent des bases entières) dans lequel les carrés parfaits seraient reconnaissables. Le problème de départ, d'origine purement algorithmique, se généralisant dès lors à des situations mathématiques plus abstraites. Cette généralisation permet d'une part de faire apparaître de nouveaux phénomènes d'intérêt propre. D'autre, elle peut entraîner une meilleure compréhension des cas classiques *a priori* plus simples.

D'autre part, le praticien a lui aussi tout intérêt à étudier la façon dont les nombres sont représentés. En effet, le coût des algorithmes dépend souvent de la manière dont sont représentés les nombres. A titre d'exemple, l'addition de deux entiers en base 2 écrits sur l'alphabet $\{0, 1\}$ requiert un temps proportionnel à la taille des données. Par contre, si on utilise une représentation au moyen des chiffres $\{-1, 0, 1\}$ alors l'addition peut être réalisée en parallèle en un temps indépendant de la taille des données. Des tels exemples se rencontrent aussi en cryptographie ou l'emploi d'autres représentations permet parfois d'obtenir des algorithmes plus efficaces. Ainsi, des représentations non standards peuvent éviter ou minimiser les

opérations coûteuses en temps machine comme par exemple l'addition pour un cryptosystème basé sur une courbe elliptique.

Dans cette section, on étudie le caractère reconnaissable des parties de $\mathbb{N}$ pour des systèmes de numération généralisés. La généralisation envisagée consiste à ne plus utiliser les puissances d'un entier k donné pour décomposer tout nombre entier mais à utiliser les termes d'une suite linéaire récurrente. Commençons par un exemple introductif. La suite de Fibonacci est donnée par

$$\begin{cases} U_0 = 1,\ U_1 = 2, \\ U_{n+2} = U_{n+1} + U_n,\ n \geq 0. \end{cases}$$

Les premiers termes de cette suite sont 1, 2, 3, 5, 8, 13, 21, 34, 55, ... On peut aussi, tout comme pour la base k, utiliser l'algorithme glouton pour représenter de manière unique les entiers :

	13	8	5	3	2	1
4				1	0	1
10		1	0	0	1	0
19	1	0	1	0	0	1
20	1	0	1	0	1	0

Ainsi, la représentation de 19 est le mot 101001 et on a donc toujours cette correspondance entre entier et mot sur un alphabet fini. L'algorithme glouton stipule qu'il faut à chaque étape considérer le plus grand terme possible de la suite $(U_n)_{n\in\mathbb{N}}$. C'est pour cette raison que "dix" se représente par "10010" et non pas "1110". En d'autres termes, $c_\ell \cdots c_0$ est une représentation valide si et seulement si

$$\sum_{i=0}^{j} c_i U_i < U_{j+1}, \quad \forall j = 0, \ldots, \ell$$

Ainsi, les représentations obtenues ne contiendront jamais deux "1" consécutifs puisque par définition de la suite de Fibonacci, deux "1" consécutifs correspondant aux termes U_{i+1} et U_i (nous sommes en présence d'un système de position) peuvent être remplacés par un seul "1" correspondant au terme U_{i+2}. On peut vérifier que l'ensemble des représentations de tous les entiers dans le système de Fibonacci est exactement formé des mots sur l'alphabet $\{0,1\}$ ne contenant pas deux 1 consécutifs. Nous pouvons formaliser quelque peu cet exemple.

Définition : Un *système de numération linéaire* est la donnée d'une suite $(U_n)_{n\in\mathbb{N}}$ strictement croissante d'entiers telle que

(1) $U_0 = 1$,

(2) le rapport $\frac{U_{n+1}}{U_n}$ est borné,

(3) la suite $(U_n)_{n\in\mathbb{N}}$ satisfait une relation de récurrence linéaire à coefficients constants et entiers.

La condition (1) assure que tout entier possède au moins une représentation. La condition (2) conduit à un alphabet fini de "chiffres" pour les représentations calculées par l'algorithme glouton. Pour employer une notation semblable au cas de la base entière, on notera encore $\rho_U(n)$ la représentation de n obtenue par l'algorithme glouton. La numération en base k est en fait un système de numération linéaire donné par la suite $(U_n)_{n\in\mathbb{N}} = (k^n)_{n\in\mathbb{N}}$.

Les chercheurs ont mis en évidence des systèmes de numération linéaires jouissant de propriétés remarquables vis-à-vis du caractère reconnaissable. Un nombre de Pisot est un entier algébrique $\theta > 1$ tel que les autres racines de son polynôme minimum ont toutes un module strictement inférieur à 1.

On va considérer des systèmes de numération linéaires tels que le polynôme caractéristique de la suite U_n soit le polynôme minimum d'un nombre de Pisot. Une propriété fondamentale des systèmes construits sur un nombre de Pisot θ est que

$$U_n \simeq \theta^n.$$

Par exemple, les systèmes à base entière sont de ce type (en effet, tout entier $k > 1$ est un nombre de Pisot ayant comme polynôme minimum $X - k$), le système de Fibonacci aussi. En effet, le polynôme caractéristique de cette suite est $X^2 - X - 1$ et ses racines sont

$$\theta = \frac{1+\sqrt{5}}{2} \quad \text{et} \quad \theta' = \frac{1-\sqrt{5}}{2}$$

où $\theta > 1$ et $|\theta'| < 1$. En particulier, il existe des constantes A et B entièrement déterminées par les conditions initiales telles que

$$U_n = A\,\theta^n + B\,\theta'^n.$$

Dès lors, $\lim_{n\to\infty} U_{n+1}/U_n = \theta$.

Pour de tels systèmes construits sur un nombre de Pisot, on dispose des propriétés suivantes [3] :

(i) $\mathbb{N}$ est U-reconnaissable, i.e., $\rho_U(\mathbb{N})$ est régulier.

(ii) L'addition préserve le caractère reconnaissable, i.e., si X et Y sont U-reconnaissables, alors $X + Y$ l'est aussi. En particulier, la multiplication par une constante préserve aussi ce caractère. Si X est U-reconnaissable, alors pour tout $\lambda \in \mathbb{N}$, λX est encore U-reconnaissable.

(iii) On dispose de plusieurs caractérisations des parties U-reconnaissables, en particulier, on dispose toujours d'une caractérisation en termes de logique du premier ordre (semblable à $\langle \mathbb{N}, +, V_k \rangle$ pour la base entière k).

La propriété (i) est fort intéressante car elle signifie qu'on peut tester aisément (c'est-à-dire par automate fini) si un mot écrit sur l'alphabet des chiffres est ou non une représentation valide obtenue par algorithme glouton. Cette propriété est d'autant plus remarquable, qu'en général, pour une suite arbitraire $(U_n)_{n\in\mathbb{N}}$, le langage $\rho_U(\mathbb{N})$ n'est pas régulier [15, 9].

Pour terminer cette section, citons une conséquence immédiate de l'algorithme glouton, pour tous $x, y \in \mathbb{N}$,

$$x < y \Leftrightarrow \rho_U(x) \prec \rho_U(y) \tag{1}$$

où $\prec$ est l'ordre généalogique (parfois appelé ordre militaire) et est défini comme suit : si u et v sont deux mots sur l'alphabet totalement ordonné $(\Sigma, <)$, alors $u \prec v$ si $|u| < |v|$ ou si les deux mots sont de même longueur et il existe $\sigma, \tau \in \Sigma$, $u', v', x \in \Sigma^*$ tels que $u = x\sigma u'$, $v = x\tau v'$ et $\sigma < \tau$. Autrement dit, pour des mots de même longueur, l'ordre généalogique coïncide avec l'ordre usuel du dictionnaire.

5. Généraliser encore plus

Au vu de la section précédente, on peut faire le constat suivant. Il est naturel (pour faciliter d'éventuels tests) que, pour un système de numération donné, l'ensemble des représentations de tous les entiers soit un langage régulier. De plus, il paraît clair d'imposer une condition analogue à (1). Partant de là, on définit assez naturellement un *système de numération abstrait* par un triplet $S = (L, \Sigma, <)$ où L est un langage régulier infini sur un alphabet totalement ordonné $(\Sigma, <)$. Enumérer les mots de L par ordre généalogique croissant fournit une bijection croissante ρ_S entre $\mathbb{N}$ et L. On note π_S, l'application réciproque, qui à un mot de L associe sa position (comptée à partir de 0). On dit que $\rho_S(n)$ est la représentation de l'entier n et que $\pi_S(w)$ est la valeur numérique de $w \in L$. Ces systèmes ont été introduits récemment dans [11] et généralisent notamment les systèmes construits sur un nombre de Pisot.

Comme exemple, prenons $S = (a^*b^*, \{a, b\}, a < b)$. On considère donc les mots commençant par un nombre arbitraire de a suivi par un nombre arbitraire de b. Par ordre généalogique, les premiers mots du langage sont

$$\varepsilon,\ a,\ b,\ aa,\ ab,\ bb,\ aaa,\ aab,\ abb,\ bbb, \ldots$$

Ainsi, dans ce système, $\rho_S(3) = aa$, $\rho_S(7) = aab$ et $\pi_S(bbb) = 9$. Par rapport aux systèmes construits sur une suite d'entiers, on remarquera qu'ici les lettres n'ont plus réellement de poids. On ne dispose plus d'un système de position. La valeur d'un mot est uniquement déterminée par la position de ce mot dans le langage ordonné. En particulier, pour la numération construite sur a^*b^*, on voit facilement que

$$\pi_S(a^p b^q) = \frac{1}{2}(p+q)(p+q+1) + q.$$

Dans la suite de ce papier, nous n'allons qu'effleurer quelques propriétés de ces systèmes généralisés. (Les preuves et d'autres résultats se trouvent notamment dans [11, 13].)

5.1. **Une formule de comptage.** Notre but est de montrer comment on peut déterminer la position d'un mot (donc sa valeur) à partir d'un automate fini déterministe. Soit $S = (L, \Sigma, <)$ un système abstrait et $\mathcal{A} = (Q, q_0, F, \Sigma, \delta)$ un automate fini déterministe tel que $L(\mathcal{A}) = L$. Pour tout $q \in Q$, on définit le langage régulier L_q des mots acceptés dans $\mathcal{A}$ depuis l'état q,

$$L_q = \{w \in \Sigma^* \mid \delta(q, w) \in F\}.$$

Pour chaque q, on peut donc définir un système de numération $S_q = (L_q, \Sigma, <)$. Si L_q est fini, alors le domaine de définition de ρ_{S_q} est réduit à $\{0, \ldots, \#L_q - 1\}$. Pour plus de facilité, on notera ρ_q et π_q les fonctions relatives à S_q.

Notons encore $u_q(n)$ (resp. $v_q(n)$) le nombre de mots de longueur n (resp. au plus n) acceptés depuis q, i.e.,

$$u_q(n) = L_q \cap \Sigma^n, \quad v_q(n) = L_q \cap \Sigma^{\leq n}.$$

On a le résultat suivant.

Proposition : Si σw appartient à L_q, $\sigma \in \Sigma$, $w \in \Sigma^*$ alors

$$\pi_q(\sigma w) = \pi_{\delta(q,\sigma)}(w) + v_q(|w|) - v_{\delta(q,\sigma)}(|w| - 1) + \sum_{\sigma' < \sigma} u_{\delta(q,\sigma')}(|w|)$$

Ce résultat est facile à prouver. Il découle de la définition de l'ordre généalogique. La valeur du mot $\sigma w \in L_q$ est exactement le nombre de mots de L_q généalogiquement strictement inférieurs à σw (en effet, la valeur d'un mot correspond à sa position). Il y a tout d'abord les mots de longueur plus petite que $|\sigma w| = |w| + 1$, par définition, il y en a $v_q(|w|)$.

Ensuite, il y a les mots de L_q de longueur $|\sigma w|$. Il y en a de deux types : ceux de la forme $\sigma' w'$ avec $\sigma' < \sigma$ et $|w'| = |w|$ et ceux de la forme $\sigma w'$ avec $w' \prec w$, $|w'| = |w|$.

Il est clair que $\sigma' w'$ appartient à L_q si et seulement si w' appartient à $L_{\delta(q,\sigma')}$. Il faut donc compter le nombre de mots de longueur $|w|$ dans $L_{\delta(q,\sigma')}$ pour toute lettre $\sigma' < \sigma$, i.e., $\sum_{\sigma'<\sigma} u_{\delta(q,\sigma')}(|w|)$.

Enfin, $\sigma w'$ appartient à L_q si et seulement si w' appartient à $L_{\delta(q,\sigma)}$. Nous voulons compter le nombre de mots w' dans $L_{\delta(q,\sigma)}$ qui sont inférieurs à w et de même longueur, i.e., $\pi_{\delta(q,\sigma)}(w) - v_{\delta(q,\sigma)}(|w| - 1)$.

5.2. **Reconnaître les polynômes.** Il faut rappeler que dans une base entière, l'ensemble des carrés parfaits n'est jamais reconnaissable [5]. Disposant de systèmes généralisés, nous pouvons espérer reconnaître un plus grand nombre d'ensembles d'entiers. Il est assez facile de trouver un langage régulier "reconnaissant" les carrés parfaits : $S = (a^*b^* \cup a^*c^*, \{a, b, c\}, a < b < c)$ convient. En effet, énumérons les premiers mots du langage,

0	**1**	2	3	**4**	5	6	7	8	**9**	10	...
ε	**a**	b	c	**aa**	ab	ac	bb	cc	**aaa**	aab	...

On remarque que k^2 est représenté par a^k, $k \geq 0$. On peut expliquer ce résultat en observant que le langage $a^*b^* \cup a^*c^*$ contient exactement $(n+1)^2 - n^2$ mots de longueur n. Dans [14], on montre comment, étant donné un polynôme $P \in \mathbb{Q}[x]$ tel que $P(\mathbb{N}) \subset \mathbb{N}$, on peut construire un langage régulier L reconnaissant $P(\mathbb{N})$.

6. Quelques références

La bibliographie donnée plus bas est bien loin d'être exhaustive. Le lecteur intéressé par les systèmes de numération de position pourra consulter le chapitre 7 de [12]. Un survol détaillé des parties reconnaissables pour les systèmes de numération en base entière se trouve dans l'excellent [2] (En particulier, on y donne une preuve du théorème de Cobham étendu aux parties de $\mathbb{N}^m$). Concernant les systèmes construits sur un nombre de Pisot, on pourra consulter [3] mais aussi [8].

Références

[1] A. V. Aho, R. Sethi, J. D. Ullman, *Compilers : Principles, Techniques, and Tools*, Addison-Wesley, (1986).

[2] V. Bruyère, G. Hansel, C. Michaux, R. Villemaire, Logic and p-recognizable sets of integers, *Bull. Belg. Math. Soc.* **1** (1994), 191–238.

[3] V. Bruyère, G. Hansel, Bertrand numeration systems and recognizability, *Theoret. Comput. Sci.* **181** (1997), 17–43.

[4] A. Cobham, On the base-dependence of sets of numbers recognizable by finite automata, *Math. Systems Theory* **3** (1969), 186–192.

[5] S. Eilenberg, *Automata, Languages and Machines*, Vol. A, Academic Press, New-York, (1974).

[6] A. S. Fraenkel, Systems of numeration, *Amer. Math. Monthly* **92** (1985), 105–114.

[7] J. E. F. Friedl, *Mastering Regular Expressions*, 2nd Edition , O'Reilley, (2002).

[8] C. Frougny, Representations of numbers and finite automata, *Math. Systems Theory* **25** (1992), 37–60.

[9] M. Hollander, Greedy numeration systems and regularity, *Theory Comput. Syst.* **31** (1998), 111-133.

[10] G. Ifrah , *Histoire universelle des chiffres, "L'intelligence des hommes racontée par les nombres et le calcul"*, Robert Laffont, collection Bouquins, (1994).

[11] P.B.A. Lecomte, M. Rigo, Numeration systems on a regular language, *Theory Comput. Syst.* **34** (2001), 27–44.

[12] M. Lothaire, *Algebraic combinatorics on words*, Cambridge University Press, Cambridge, (2002).

[13] M. Rigo, Numeration systems on a regular language : Arithmetic operations, recognizability and formal power series, *Theoret. Comput. Sci.* **269** (2001), 469–498.

[14] M. Rigo, Construction of regular languages and recognizability of polynomials, *Discrete Math.* **254** (2002), 485–496.

[15] J. Shallit, Numeration systems, linear recurrences, and regular sets, *Inform. and Comput.* **113** (1994), 331–347.

[16] T. Sudkamp, *Languages and Machines : An Introduction to the Theory of Computer Science*, second edition, Addison Wesley (1997).

[17] P. Wolper, *Introduction à la calculabilité*, seconde édition, Dunod, (2001).

(Michel Rigo)
Université de Liège,
Institut de Mathématique,
Grande Traverse 12 (B 37),
B-4000 Liège,
Belgique.
E-mail address: M.Rigo@ulg.ac.be