Réseaux linéaires sur nombre de Nim
Rapport de stage de L3

Jules Bertrand
ENS de Lyon

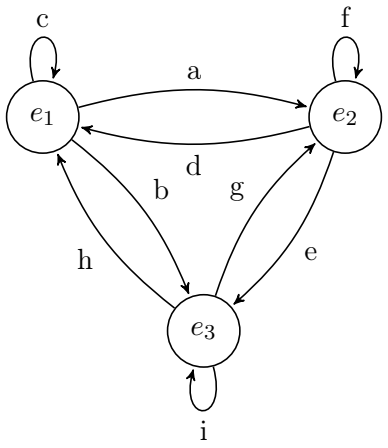
Encadré par:
Christophe Papazian et **Enrico Formenti**
Équipe MDSC, Sophia-Antipolis, Laboratoire I3S, France

Table des matières

Introduction

Dans ce rapport, je traite des réseaux linéaires dont les poids sont des nombres de Nim. Les réseaux linéaires sont des graphes d'états dont les nœuds changent d'état selon une fonction linéaire des poids des arêtes et des états des nœuds.

On peut par exemple considérer un réseau linéaire comme celui-ci :



Après une itération du processus, si f est la fonction linéaire de changement d'état, le premier nœud sera dans l'état e'_1 valant $f(c, e_1, d, e_2, h, e_3)$.

J'ai restreint mon travail aux cas où le nouvel état du nœud est la somme du produit des poids des arêtes et des nœuds. Dans le cas précédent, le premier nœud aura changé d'état pour $c \cdot e_1 + d \cdot e_2 + h \cdot e_3$.

L'un des problèmes classiques lié aux réseaux linéaires est celui de l'existence et de la taille des orbites finies. Plus simplement, on se demande si l'on revient à l'état initial après plusieurs itérations et si oui, on cherche à déterminer ce nombre d'itérations. On cherche à déterminer, pour un réseau fini, les différentes tailles d'orbites possibles, dépendant des états de départ.

Je m'intéresserai dans ce rapport au cas où les poids des arêtes ainsi que les états des nœuds sont des nombres de Nim ou nimbers. Ces objets mathématiques, proches des ordinaux de la logique, peuvent être dotés d'une addition et d'une multiplication assurant l'existence d'orbites finies et permettant ainsi de restreindre mon travail à la détermination de la taille de celles-ci.

J'ai donc commencé par m'intéresser aux ordinaux de la théorie des ensembles afin de comprendre les opérations dont ils sont usuellement dotés. En effet, les opérations de Nim, que ce soit l'addition ou la multiplication, coïncident avec les applications usuelles dans les cas les plus simples. J'ai ensuite réduit mon étude des opérations de Nim aux entiers avant de l'étendre à ω^{ω^ω}, espace dans lequel je travaille pour les réseaux linéaires. Enfin, j'ai travaillé sur l'extraction de racines de polynômes. En effet, si l'on utilise la matrice de transition associée à un réseau linéaire, on peut chercher à déterminer ses valeurs propres. On peut alors étudier leur ordre pour en déduire les tailles d'orbites possibles.

1 Familiarisation avec les ordinaux de la théorie des ensembles

Les ordinaux sont des outils mathématiques permettant de caractériser les relations d'ordre d'un ensemble quelconque. Les ordinaux peuvent être définis de deux façons, décrites ci-dessous.

On peut tout d'abord définir les ordinaux comme des classes d'équivalence sur des ensembles bien ordonnés. Ainsi, un ordinal est un ensemble bien ordonné à isomorphisme d'ordre près, les isomorphismes d'ordre étant des bijections croissantes. De cette façon, si seuls les éléments de l'ensemble changent, l'ordinal utilisé est invariant. On ne s'intéresse donc qu'à la structure de l'ordre.

L'autre définition due à John Von Neumann propose une construction récursive [3]. Un ordinal est défini par l'ensemble des ordinaux qui le précèdent.
Plus précisément, α est un ordinal si :

- $\in$ est un bon ordre strict pour cet ensemble.
  Un ordre strict $\in$ est une relation sur un ensemble vérifiant les propriétés suivantes :
  - $\forall x \in \alpha \; x \notin x$ (antiréflexivité)
  - $\forall x, y, z \in \alpha \; (x \in y \wedge y \in z) \Rightarrow x \in z$ (transitivité)

  Un ordre strict est un bon ordre strict sur un ensemble E s'il munit E d'une relation d'ordre qui vérifie la propriété suivante : toute partie de E non vide possède un plus petit élément.
  L'ordre sur E doit donc être total, puisque pour tout x, y $\{x, y\}$ étant une partie non vide de E, elle admet un plus petit élément et on a donc x appartient à y ou y appartient à x.
- L'ensemble α doit être transitif c'est à dire qu'il vérifie : $\forall x \in \alpha, \forall y \in x, y \in \alpha$.

La définition de Von Neumann permet de construire les ordinaux finis en les liant aux entiers naturels :

0		$\varnothing$
1	{0}	$\{\varnothing\}$
2	{0,1}	$\{\varnothing, \{\varnothing\}\}$
3	{0,1,2}	$\{\varnothing, \{\varnothing\}, \{\varnothing, \{\varnothing\}\}\}$
4	{0,1,2,3}	$\{\varnothing, \{\varnothing\}, \{\varnothing, \{\varnothing\}\}, \{\varnothing, \{\varnothing\}, \{\varnothing, \{\varnothing\}\}\}\}$

Ce parallèle est dû au fait que les entiers naturels sont définis par l'ensemble des entiers qui les précèdent dans l'arithmétique de Peano, ce qui coïncide avec la construction des ordinaux de Von Neumann.

L'axiome de l'infini énonçant l'existence d'un ensemble auquel appartient l'ensemble vide et clos par application du successeur $x \to x \cup \{x\}$ assure quant à lui l'existence des ordinaux infinis et en particulier de ω, le premier des ordinaux infinis.

Dans la suite de ce rapport, j'utiliserai principalement la définition de Von Neumann car la première définition ne saurait être formalisée dans ZF ou ZFC. La définition de Von Neumann évite cet écueil en construisant un représentant de chaque classe sans construire

celle-ci. La première définition sera cependant utile pour mieux comprendre les opérations définies sur les ordinaux.

1.1 Les opérations usuelles

Afin de créer des ordinaux de plus en plus grand au sens de Von Neumann, on munit ceux-ci de différentes opérations. Les plus classiques et celles que j'utiliserai au sein de mon rapport sont l'addition, la multiplication et l'exponentiation de deux ordinaux.

1.1.1 L'addition

L'addition est définie par les deux règles suivantes :

- $\forall \alpha,\ \alpha + 0 = \alpha$
- $\forall \alpha, \beta,\ \ \alpha + \beta = \sup\{\alpha + \gamma + 1 \text{ tels que } \gamma \in \beta\}$

Dans le cas des ordinaux finis, l'addition coïncide avec celle définie sur les entiers. Elle présente cependant une particularité majeure qui la distingue de l'addition usuelle : elle est associative mais n'est pas commutative.

Par exemple, si l'on considère l'ordinal ω et l'ordinal 1, on a :

$$\begin{aligned} 1{+}\omega &= \sup\{1 + \gamma + 1 \text{ tels que } \gamma \in \omega\} \\ &= \sup\{\gamma \text{ tels que } \gamma \in \omega\} \\ &= \omega \end{aligned}$$

$$\begin{aligned} \omega + 1 &= \sup\{\omega + \gamma + 1 \text{ tels que } \gamma \in 1\} \\ &= \sup\{\omega + 1\} \\ &= \omega + 1 \end{aligned}$$

On a donc $\omega + 1 \neq 1 + \omega$

Cependant, cela ne nous permet pas de comprendre ce qu'est $\omega + 1$. Pour cela, il faut revenir à la définition des ordinaux comme structure d'un ensemble. Ainsi, si on a deux ordinaux α et β, munis respectivement des relations d'ordre $<_\alpha$ et $<_\beta$, l'ordinal $\alpha + \beta$ correspond à un ensemble $\alpha \cup \beta$ muni de la relation d'ordre $<_{\alpha+\beta}$ suivante :

- $\forall x, y,\ (x \in \alpha \wedge y \in \alpha) \Rightarrow (x <_{\alpha+\beta} y \Leftrightarrow x <_\alpha y)$
- $\forall x, y,\ (x \in \beta \wedge y \in \beta) \Rightarrow (x <_{\alpha+\beta} y \Leftrightarrow x <_\beta y)$
- $\forall x, y,\ (x \in \alpha \wedge y \in \beta) \Rightarrow x <_{\alpha+\beta} y$

Pour illustrer cela, on peut se placer dans le cas de l'hôtel de Hilbert. C'est un hôtel avec un nombre infini mais dénombrable de chambres. On compare trois hôtels :

- Un hôtel avec une infinité de chambre au rez-de-chaussée.
- Un hôtel avec une infinité de chambre au rez-de-chaussée et une chambre à l'étage -1.
- Un hôtel avec une infinité de chambre au rez-de-chaussée et une chambre au premier étage.se déplacer pour

On s'intéresse à l'ordre sur les chambres dans ces hôtels, les chambres d'un même étage étant numérotées par les entiers naturels. Les chambres d'étages différents sont classées selon leur étage. Ainsi, une chambre à l'étage 0 sera toujours considérée comme plus petite qu'une chambre à l'étage 1.

On remarque alors que les deux premiers hôtels sont en fait dotés d'un ordre semblable. En effet, on peut faire correspondre la chambre de l'étage -1 du deuxième hôtel à la première chambre du premier hôtel et décaler toutes les autres chambres d'un dans leur numérotation. Ainsi, on a construit une bijection croissante, soit un isomorphisme d'ordre entre les deux hôtels.

On constate cependant que le même procédé n'est pas possible pour faire correspondre les ordres du premier et du dernier hôtel. En effet, si une bijection croissante entre ceux-ci existait, l'image de la chambre au premier étage n'aurait qu'un nombre fini de chambres qui lui soient inférieures, ce qui est contradictoire.

Comme le premier hôtel correspondait à l'ordinal ω, le deuxième à $1 + \omega$ et le dernier à $\omega + 1$ puisque ω est l'ordre usuel sur les entiers, on a montré, via la première définition qu'on avait $1 + \omega = \omega \neq dplacerpoura + 1$

On remarque qu'avec l'addition, on ne peut sommer qu'un nombre fini de termes et que tous les ordinaux constructibles ainsi sont de la forme $\omega \cdot n + m$ avec n et m des entiers et $\omega \cdot n$ l'itérée n fois de la somme de ω. La multiplication va nous permettre de créer des ordinaux encore plus grands.

1.1.2 La multiplication

La multiplication est définie par les deux règles suivantes :

- $\forall \alpha,\ \alpha \cdot 0 = 0$
- $\forall \alpha, \beta,\quad \alpha \cdot \beta = \sup\{\alpha \cdot \gamma + \alpha \text{ tels que } \gamma \in \beta\}$

La multiplication vérifie des propriétés proches de celles de l'addition. En effet, elle coïncide avec la multiplication usuelle sur les entiers, est associative mais n'est pas commutative.

$$\begin{aligned} \omega \cdot 2 &= \omega + \omega \\ 2 \cdot \omega &= \sup\{2 \cdot n + 2 \text{ tels que } n \in \omega\} \\ &= \sup\{n \text{ tels que } n \in \omega\} \\ &= \omega \end{aligned}$$

On a donc $\omega \cdot 2 \neq 2 \cdot \omega$

On peut aussi définir la multiplication à l'aide de la première définition des ordinaux. Si α et β sont deux ordinaux munis respectivement des relations d'ordre $<_\alpha$ et $<_\beta$, l'ordinal $\alpha \cdot \beta$ correspond à un ensemble $\alpha \times \beta$ muni de la relation d'ordre $<_{\alpha \cdot \beta}$ suivante :
$\forall \alpha_1, \alpha_2, \beta_1, \beta_2,\quad (\alpha_1, \beta_1) <_{\alpha \cdot \beta} (\alpha_2, \beta_2) \Leftrightarrow [\beta_1 <_\beta \beta_2 \vee (\beta_1 = \beta_2 \wedge \alpha_1 <_\alpha \alpha_2)]$

Pour reprendre l'analogie avec l'hôtel de Hilbert, $\omega \cdot 2$ correspond à un hôtel à deux niveaux, tous deux ayant une infinité de chambres tandis que $2 \cdot \omega$ correspond à un hôtel ayant une infinité d'étages mais avec seulement deux chambres par étage. On peut ainsi retrouver $2 \cdot \omega = \omega$ en réalisant un isomorphisme d'ordre de ce second hôtel avec un hôtel avec un nombre infini de chambres au rez-de-chaussée.

Elle vérifie aussi la distributivité à gauche : $\forall \alpha, \beta, \gamma,\ \alpha \cdot (\beta + \gamma) = \alpha \cdot \beta + \alpha \cdot \gamma$.

Grâce à la multiplication, on peut ainsi construire des ordinaux de la forme $\sum_{i=0}^{k} \omega^{n_i} \cdot m_i$ avec n_i, m_i des entiers. L'exponentiation est une opération permettant de construire des ordinaux encore plus grands.

1.1.3 L'exponentiation

L'exponentiation est définie ainsi :

- $\forall \alpha, \alpha^0 = 1$
- $\forall \alpha, \beta,\ \alpha^\beta = \sup\{\alpha^\gamma \cdot \alpha \text{ tels que } \gamma \in \beta\}$

Comme l'addition et la multiplication, l'exponentiation coïncide sur les entiers avec l'exponentiation usuelle. Elle vérifie notamment deux propriétés qui me seront utiles au cours de ce rapport :

- $\forall \alpha, \beta, \gamma,\ \alpha^{\beta+\gamma} = \alpha^\beta \cdot \alpha^\gamma$
- $\forall \alpha, \beta, \gamma,\ \alpha^{\beta \cdot \gamma} = (\alpha^\beta)^\gamma$

Illustrer la définition de l'exponentiation en terme de structure d'ordre engendrée est plus compliquée puisque cela repose sur les fonctions de α dans β à support fini.

On remarquera que $n^\omega = \omega$ si n est entier. Cela sera utile dans la suite du rapport.

Il existe des ordinaux bien plus grand que les itérés de l'exponentiation un nombre fini de fois tels que $\epsilon_0 = \sup\{u_n\}$ avec $u_0 = 0$ et $u_{n+1} = w^{u_n}$. Cependant, la connaissance d'ordinaux de cette sorte n'est pas nécessaire pour la suite de ce rapport.

1.2 Mise sous forme normale de Cantor

Comme on l'a vu, un même ordinal peut avoir de multiples écritures. La forme normale de Cantor est une notation unique pour chaque ordinal de sa décomposition en base ω. Elle présente un intérêt lorsque l'on étudie des ordinaux inférieurs à ϵ_0, ce qui sera le cas ici.

On peut ainsi montrer que tout ordinal inférieur à ϵ_0 peut s'écrire $\sum_{i=0}^{k} \omega^{\beta_i} \cdot k_i$ avec k_i entiers [2]. Les β_i sont eux-mêmes des ordinaux. C'est pourquoi, en écrivant les β_i de la même façon, on peut faire correspondre une écriture unique à chaque ordinal inférieur à ϵ_0.

Mon premier travail a donc été de réussir à comprendre les trois opérations définies précédemment afin de concevoir puis implémenter un algorithme calculant la forme normale de Cantor de tout ordinal inférieur à ϵ_0.

Soient α et β deux ordinaux non nuls sous forme normale de Cantor :

- $\alpha = \sum_{i=0}^{k} \omega^{\alpha_i} \cdot n_i$

- $\beta = \sum_{j=0}^{l} \omega^{\beta_j} \cdot m_j$

1.2.1 L'addition

Alors, $\alpha + \beta = \sup\{\alpha + \gamma + 1 \text{ tel que } \gamma < \beta\}$ vaut :

- β si $\beta_0 > \alpha_0$
- $\omega^{\alpha_0} \cdot n_0 + (\sum_{i=1}^{k} \omega^{\alpha_i} \cdot n_i + \beta)$ si $\beta_0 < \alpha_0$
- $\omega^{\alpha_0} \cdot (n_0 + m_0) + \sum_{j=1}^{l} \omega^{\beta_j} \cdot m_j$ si $\beta_0 = \alpha_0$

1.2.2 La multiplication

Alors, $\alpha \cdot \omega^b \cdot m = \sup\{\alpha \cdot \gamma + \alpha \text{ tel que } \gamma < \omega^b \cdot m\}$ vaut :

- $\omega^{\alpha_0} \cdot m + \sum_{i=1}^{k} \omega^{\alpha_i} \cdot n_i$ si b=0
- $\omega^{\alpha_0 + b} \cdot m$ sinon

A l'aide de la distributivité à gauche de la multiplication par rapport à l'addition, on en déduit un algorithme simple pour la multiplication de deux ordinaux.

1.2.3 L'exponentiation

La forme normale de Cantor de l'exponentiation de deux ordinaux peut être déterminée par les règles suivantes :

- $\alpha^{\beta} = \prod_{j=0}^{l} (\alpha^{\omega^{\beta_j}})^{m_j}$
- $\alpha^{\omega^{\gamma}} = \omega^{\alpha_0 \cdot \omega^{\gamma}}$ si γ est non nul
- $\alpha^m = \sum_{i=0}^{k} \omega^{\alpha_0 \cdot (m-1) + \alpha_i} \cdot n_i$ si α ne possède pas de terme entier dans sa décomposition en forme normale de Cantor et $m \in \omega$.
- $\alpha^m = \sum_{j=1}^{k} \sum_{i=0}^{k} \omega^{\alpha_0 \cdot (m-j) + \alpha_i} \cdot n_i$ si α possède un terme entier dans sa décomposition en forme normale de Cantor et $m \in \omega$
- $n^{\omega^s} = \omega^{\omega^{s-1}}$ avec $n, s \in \omega$ et $s \geqslant 1$
- $n^{\omega^{\gamma}} = \omega^{\omega^{\gamma}}$ si γ n'est pas entier

Les démonstrations de ces égalités sont laissées en annexe.

Après avoir implémenté cet algorithme pour unifier l'écriture des ordinaux inférieurs à ϵ_0, j'ai pu commencer à m'intéresser aux opérations de Nim. En effet, celles-ci coïncidaient avec les opérations usuelles dans certains cas sur lesquels je voulais m'appuyer pour les calculer. Je considère ainsi, dans la suite de ce rapport, les ordinaux comme déjà sous forme normale de Cantor.

2 Étude de l'addition et de la multiplication de Nim

Les nombres de Nim ou nimbers sont couramment utilisés en théorie des jeux puisqu'ils permettent de déterminer les stratégies gagnantes dans des jeux à deux joueurs à information complète. Je ne me suis pas intéressé aux nimbers pour leur intérêt pratique, mais pour les opérations dont ils peuvent être dotés.

Dans la suite de ce rapport, je distinguerai les opérations usuelles sur les ordinaux des opérations de Nim en encadrant les premières entre crochets.

Les nimbers sont donc des ordinaux que l'on dote d'une addition et d'une multiplication différentes qui permettent d'obtenir des sous-corps.

On note $mex(E)$ le plus petit ordinal n'appartenant pas à l'ensemble E.
Ainsi, $mex(\{0,1,3\}) = 2$

On dote les ordinaux de l'addition et la multiplication suivantes :

- $\forall \alpha, \beta,\ \alpha + \beta = mex(\{\alpha' + \beta \text{ tels que } \alpha' \in \alpha\} \cup \{\alpha + \beta' \text{ tels que } \beta' \in \beta\})$
- $\forall \alpha, \beta,\ \alpha \cdot \beta = mex(\{\alpha' \cdot \beta + \alpha \cdot \beta' + \alpha' \cdot \beta' \text{ tels que } \alpha' \in \alpha, \beta' \in \beta\})$

Cela nous permet de créer les deux tableaux suivants sur les 5 premiers entiers :

+	0	1	2	3	4
0	0	1	2	3	4
1	1	0	3	2	5
2	2	3	0	1	6
3	3	2	1	0	7
4	4	5	6	7	0

×	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	3	1	8
3	0	3	1	2	12
4	0	4	8	12	6

Cependant, on remarque que la complexité du calcul de l'addition, et encore plus de la multiplication, sont très élevées. En effet, même en réalisant une programmation dynamique, la somme de n et m est réalisée en $O(n^2 + m^2)$. De plus, de telles méthodes ne sauraient s'étendre aux ordinaux. En effet, rien que dans le cas de ω, il nous faudrait déterminer le mex d'un ensemble infini. Il est donc nécessaire de trouver une autre façon de réaliser les opérations sur les nimbers.

2.1 Étude restreinte sur les entiers

J'ai tout d'abord cherché à comprendre les méthodes proposées par J. H. Conway [1]. Celui-ci démontre des règles simples pour l'addition et la multiplication :

- $\forall n, m,\ n \neq m \Rightarrow 2^n + 2^m = [2^n + 2^m]$
- $\forall n,\ 2^n + 2^n = 0$
- $\forall n, m,\ n \neq m \Rightarrow 2^{2^n} \cdot 2^{2^m} = \left[2^{2^n} \cdot 2^{2^m}\right]$
- $\forall n,\ 2^{2^n} \cdot 2^{2^n} = \left[2^{2^n - 1}\right]$

Il montre de plus l'associativité et la commutativité de l'addition ainsi que celles de la multiplication. La multiplication est aussi distributive à droite et à gauche par rapport à l'addition.

On peut déduire de ses règles un algorithme simple pour calculer l'addition de deux nombres. En effet, il suffit de décomposer les deux nombres en base 2 et ne conserver que les puissances de 2 distinctes.

Par exemple, $9 = 2^3 + 2^0$ et $13 = 2^3 + 2^2 + 2^0$ donc $13 + 9 = 2^2 = 4$.

Le calcul de la multiplication est plus complexe. En effet, il est d'abord nécessaire de décomposer les deux termes en puissance de 2 puis de développer les deux sommes afin de n'obtenir que des produits de puissances de 2 qu'on saura calculer à l'aide des règles sur la multiplication.

Réalisons par exemple le calcul de 9×13.

$$\begin{aligned}
9{\times}13 &= \left(2^3 + 2^0\right) \times \left(2^3 + 2^2 + 2^0\right) \\
&= 2^3 \cdot 2^3 + 2^3 \cdot 2^2 + 2^3 + 2^3 + 2^2 + 2^0 \\
&= 2^{2^1} \cdot 2^{2^0} \cdot 2^{2^1} \cdot 2^{2^0} + 2^{2^1} \cdot 2^{2^0} \cdot 2^{2^1} + 2^{2^1} \cdot 2^{2^0} + 2^{2^1} \cdot 2^{2^0} + 2^{2^1} + 2^0 \\
&= 6{\cdot}3 + 6 \cdot 2 + 8 + 8 + 4 + 1 \\
&= \left(2^2 + 2^1\right) \cdot \left(2^1 + 2^0\right) + \left(2^2 + 2^1\right) \cdot 2^1 + 5 \\
&= 2^{2^1} \cdot 2^{2^0} + 2^2 + 2^{2^0} \cdot 2^{2^0} + 2^1 + 2^{2^1} \cdot 2^{2^0} + 2^{2^0} \cdot 2^{2^0} + 5 \\
&= 8{+}4{+}3{+}2{+}8{+}3{+}5 \\
&= 4{+}2{+}5 \\
&= 3
\end{aligned}$$

La complexité de cet algorithme reste encore à déterminer, je n'ai pas travaillé dessus puisque je cherchais simplement à comprendre la multiplication de Nim.

2.2 Étude restreinte à ω^{ω^ω}

Le calcul de l'addition et du produit de Nim de deux ordinaux constitue une importante partie de mon travail.

L'addition de Nim de deux ordinaux se calcule de façon similaire à celle sur les entiers. En effet, en s'appuyant sur l'égalité $2^\omega = \omega$ et sur la forme normale de Cantor, on peut écrire la décomposition binaire de n'importe quel ordinal inférieur à ϵ_0. Il suffit alors de ne conserver que les puissances de 2 distinctes pour obtenir le résultat de l'addition de Nim.

Ainsi, on a par exemple :

$$\begin{aligned}
& \left[\omega^\omega \cdot 3 + \omega + 5\right] + \left[\omega^\omega \cdot 2 + \omega \cdot 9 + 4\right] \\
= \; & \left[2^{\omega^2} \cdot \left(2^1 + 2^0\right) + 2^\omega \cdot 2^0 + 2^2 + 2^0\right] + \left[2^{\omega^2} \cdot 2^0 + 2^\omega \cdot \left(2^3 + 2^0\right) + 2^2\right] \\
= \; & \left[2^{\omega^2+1} + 2^{\omega^2} + 2^\omega + 2^2 + 2^0\right] + \left[2^{\omega^2} + 2^{\omega+3} + 2^\omega + 2^2\right] \\
= \; & 2^{\omega^2} \cdot 2^1 + 2^\omega \cdot 2^3 + 2^0 \\
= \; & \omega^\omega + \omega \cdot 8 + 1
\end{aligned}$$

On remarque qu'il n'est pas nécessaire de réitérer l'écriture binaire aux exposants. Cela sera d'ailleurs utile pour calculer la multiplication de deux ordinaux.

Pour calculer le produit de deux nimbers, on peut se restreindre au calcul de $2^\alpha \cdot 2^\beta$. En

effet, il suffit de décomposer les deux facteurs en sommes de puissances de deux, comme pour le calcul du produit de deux entiers.

On peut remarquer que dans la suite $\omega, \omega^3, \omega^9, \omega^{27}$..., chaque terme est le cube de son prédécesseur pour la multiplication de Nim [1].
De même, dans la suite $\omega^\omega, \omega^{\omega \cdot 5}, \omega^{\omega \cdot 25}, \omega^{\omega \cdot 125}$..., chaque terme vaut son prédécesseur élevé à la cinquième puissance de Nim.

Dans la suite des calculs, on note $x_{p^{n+1}}$ l'ordinal $2^{\omega^k \cdot p^n}$, avec p le $k+1^{\text{ième}}$ nombre premier.
Ainsi, on peut décomposer toute puissance de 2 sous la forme $\prod_{i,j} x_{p^n}^{m_{i,j}}$ avec $m_{i,j} < p$.

En effet, si l'exposant α est de la forme $\sum_{i=0}^{r} \omega^{k_i} \cdot n_i$, on peut décomposer chacun des n_i selon la base des p_i^n avec p_i le $k_i + 1^{\text{ième}}$ nombre premier. On peut alors écrire α comme $\sum_{i=0}^{r} \sum_{j=0}^{t} \omega^{k_i} \cdot {p_i}^j \cdot m_{i,j}$ et donc 2^α comme $\prod_{i,j} x_{p_i^{j+1}}^{m_{i,j}}$

Or, on a $\left[\prod_{i,j} x_{p_i^{j+1}}^{m_{i,j}}\right] = \prod_{i,j} x_{p_i^{j+1}}^{m_{i,j}}$ si $m_{i,j} < p$ [4].

Il nous faut cependant déterminer le résultat de $[x_{p^n}]^n$
On a trois cas à traiter :

- Si $k = 0$, on est dans le cas des entiers déjà précédemment traité. On a donc $[x_{2^n}]^2 = x_{2^n} + \prod_{i=0}^{n-1} x_{2^i}$
- Si $k > 0$ et $n > 0$, on a $\left[x_{p^{n+1}}\right]^n = x_{p^n}$. Cela coïncide avec les propriétés que l'on avait notamment constaté sur les cubes et les puissances cinquièmes.
- Si $k > 0$ et $n = 0$, on a $[x_{p^n}]^n = \alpha_p$ avec α_p le plus petit ordinal n'ayant pas de racine $p^{\text{ième}}$ dans x_{p^n}

Le calcul des α_p est détaillé par H. W. Lenstra dans [4]. Je ne l'ai pas expliqué car celui-ci relève plus d'un calcul préliminaire à la multiplication.

Voici cependant le tableau suivant qui nous permettra de réaliser des multiplications de Nim sur les ordinaux les plus petits :

k	p	x_p	α_p
1	3	ω	2
2	5	ω^ω	4
3	7	ω^{ω^2}	$\omega + 1$
4	11	ω^{ω^3}	$\omega^\omega + 1$
5	13	ω^{ω^4}	$\omega + 4$
6	17	ω^{ω^5}	16

On peut ainsi calculer :

$$
\begin{aligned}
\left[\omega^{\omega\cdot 4}\cdot\omega^2\cdot 9\right]\cdot\left[\omega^{\omega\cdot 3}\cdot\omega\cdot 12\right] &= \left[(2^{\omega^2})^4\cdot(2^{\omega})^2\cdot 9\right]\cdot\left[(2^{\omega^2})^3\cdot 2^{\omega}\cdot 12\right]\\
&= \left[(x_5)^4\cdot(x_3)^2\cdot 9\right]\cdot\left[(x_5)^3\cdot x_3\cdot 12\right]\\
&= [x_5]^7\cdot[x_3]^3\cdot 9\cdot 12\\
&= [x_5]^2\cdot[x_5]^5\cdot\alpha_3\cdot 10\\
&= (\mathrm{x}_5)^2\cdot\alpha_5\cdot 2\cdot 10\\
&= \omega^{\omega\cdot 2}\cdot 4\cdot 15\\
&= \omega^{\omega\cdot 2}
\end{aligned}
$$

Je n'ai pas explicité les calculs sur les entiers car je les ai déjà étudiés dans la partie précédente.

Cette construction nous assure des propriétés intéressantes. En effet, on sait ainsi que ω est quadratiquement clos , c'est à dire que tout polynôme de degré 2 à coefficients dans ω y admet une racine. De même, ω^{ω} ou 2^{ω^2} est cubiquement clos. On montre ainsi que $\omega^{\omega^{\omega}}$ est un corps algébriquement clos, c'est à dire que tout polynôme ayant pour coefficients des éléments de $\omega^{\omega^{\omega}}$ y admet une racine. On en déduit même que tous ces polynômes sont scindés, toutes leurs racines appartiennent à $\omega^{\omega^{\omega}}$ [1].

Cette dernière propriété est particulièrement intéressante puisqu'elle nous assure l'existence des valeurs propres des matrices associées aux réseaux linéaires. Le travail de la prochaine partie sera de chercher à déterminer les racines de ces polynômes.

3 Recherche d'orbites et calcul de leur cardinal

Comme on l'a vu plus tôt, $\omega^{\omega^{\omega}}$ constitue un corps algébriquement clos. J'ai cherché à utiliser cette propriété afin de travailler sur les tailles des orbites finies des réseaux linéaires.

A chaque réseau linéaire, il est possible d'associer sa matrice de transition. Ainsi, changer les états du réseau équivaut à la multiplication d'un vecteur par une matrice. Si l'on souhaite réaliser plusieurs itérations du procédé, il suffit donc de calculer la puissance de la matrice associée.

Ainsi, calculer les changements d'états du réseau linéaire de l'introduction revient à réaliser le produit de la matrice M avec le vecteur E :

$$
M = \begin{pmatrix} \mathrm{c} & \mathrm{d} & \mathrm{h} \\ \mathrm{a} & \mathrm{f} & \mathrm{g} \\ \mathrm{b} & \mathrm{e} & \mathrm{i} \end{pmatrix} \qquad E = \begin{pmatrix} \mathrm{e}_1 \\ \mathrm{e}_2 \\ \mathrm{e}_3 \end{pmatrix}
$$

Une autre des propriétés intéressantes des nimbers est la suivante : un nombre fini de nimbers engendre un sous-corps fini.

Ainsi, les poids des arêtes étant fixes, le vecteur des états ne peut prendre qu'un nombre fini de valeurs pour un état initial donné. Le réseau linéaire admet donc nécessairement des orbites de taille finie.

Dans cette partie, je m'intéresse donc aux polynômes caractéristiques des matrices de transition. En effet, puisque $\omega^{\omega^{\omega}}$ est algébriquement clos, le polynôme caractéristique est nécessairement scindé. En s'appuyant sur l'ordre des valeurs propres, on pourra en déduire les tailles possibles d'orbites finies.

On remarque tout d'abord qu'il existe un algorithme simple qui nous permettrait de déterminer les différentes racines. En effet, puisque ω^{ω^ω} est dénombrable, il est possible de tester chacun des éléments jusqu'à l'obtention d'une racine. Si l'on réalise des divisions du polynôme caractéristique par le polynôme $(X - r)$ avec r une racine, on peut ainsi déterminer les racines multiples. Comme toutes les racines appartiennent à ω^{ω^ω}, l'algorithme termine bien.

Cependant, il n'est pas raisonnable de travailler avec un tel algorithme. C'est pourquoi j'ai cherché à trouver d'autres alternatives pour les polynômes de degré 2 et à réduire l'ensemble de recherche dans des cas plus complexes.

3.1 Étude des polynômes du second degré

Les polynômes de degré deux correspondent aux réseaux linéaires à deux états. Puisque l'on se place dans un corps commutatif de caractéristique deux, c'est à dire $\forall x, x + x = 0$, on a la propriété suivante : $\forall x, y\ (x + y)^2 = x^2 + y^2$

J'ai tout d'abord travaillé sur la forme de mes polynômes afin de me ramener à des cas plus simples.

Soit P un polynôme de degré 2. On a deux cas à traiter selon le coefficient de degré 1 :

- $\exists a \neq 0, c$ tels que $P = a \cdot X^2 + c$
  On cherche x tel que $P(x) = 0 \quad \Leftrightarrow \quad a \cdot x^2 = c$
  $\Leftrightarrow \quad x^2 = c \cdot a^{-1}$
  Pour traiter ce cas-ci, il nous suffit donc de savoir déterminer la racine carré d'un nimber puisque l'on sait déjà réaliser algorithmiquement les différentes étapes des équivalences.

- $\exists a \neq 0, b \neq 0, c$ tels que $P = a \cdot X^2 + b \cdot X + c$
  On cherche x tel que $P(x) = 0 \quad \Leftrightarrow \quad a \cdot x^2 + b \cdot x = c$
  $\Leftrightarrow \quad x^2 + b \cdot a^{-1} \cdot x = c \cdot a^{-1}$
  $\Leftrightarrow \quad (b \cdot a^{-1})^2 \cdot y^2 + (b \cdot a^{-1})^2 \cdot y = c \cdot a^{-1}$
  En posant $y = b \cdot a^{-1} \cdot x$
  $\Leftrightarrow \quad y^2 + y = c \cdot a \cdot (b^{-1})^2$
  Dans ce second cas, il faut donc savoir résoudre une équation de la forme $x^2 + x = c$.

3.1.1 Calcul de racines carrés

On sait que chaque élément non nul x admet un inverse. Celui-ci est nécessairement une puissance de x.

En effet, le corps engendré par x étant fini,
Il existe $n, m,$ tels que $n < m$ et $x^n = x^m$ ce qui implique $x^n = x^n \cdot x^{m-n}$.
On a donc x^{m-n-1} inverse de x.

On peut même montrer que l'ordre d'un élément est toujours impair.

$$\forall x, \exists n \text{ tel que } x^{2n+1} = 1$$

On a donc $(x^{n+1})^2 = x^{2n+2} = x$

On sait donc déterminer une racine carrée d'un nimber c bien plus efficacement qu'en énumérant ω^{ω^ω}. On peut cependant améliorer cet algorithme en remarquant que si pour tout$x, y,\ (x + y)^2 = x^2 + y^2$, on en déduit que si x est racine carrée de X et y racine

carrée de Y, $x+y$ est racine carrée de $X+Y$. Ainsi puisque l'ordre de la somme de deux nimbers peut être le PPCM de l'ordre des deux nimbers, on réduit fortement les calculs en décomposant c en binaire.

De même, puisque $(x \cdot y)^2 = x^2 \cdot y^2$, on peut calculer séparément les racines carrées des termes d'un produit pour en déduire celle du produit.

Enfin, on remarque que toutes les racines carrées sont en fait des racines doubles. En effet, si $(X+r_1)\cdot(X+r_2) = X^2 + c, r_1 + r_2 = 0$ et donc$r_1 = r_2$

3.1.2 Résolution d'équations de la forme $x^2 + x = c$

Pour résoudre une telle équation, on cherche à réaliser des changements de variables nous permettant de diminuer c.

En effet, si $y + a$ est solution de $x^2 + x = c$, y est solution de $y^2 + y = a^2 + a + c$. On cherche donc à déterminer des valeurs de a permettant d'avoir $a^2 + a + c < c$.

On a deux cas à traiter :

- Si c est entier, puisque ω est quadratiquement clos, les solutions étant entières, a le sera aussi.
  On cherche ainsi à décomposer c en binaire afin que chacun des a choisis réduise c d'un terme de sa décomposition.
  Par exemple, si on cherche à déterminer x tel que $x^2 + x = 11$, comme $8 \leqslant 11 < 16$, on pose $y = x + 16$ et on a donc $y^2 + y = 24 + 16 + 11 = 3$. En réitérant le procédé, on obtient 22 comme solution de l'équation. On en déduit que si r_2 est l'autre racine du polynôme, on a $22 + r_2 = 1$ dont on peut immédiatement déduire $r_2 = 23$. On a ainsi identifié les deux racines du polynôme.

- Si c n'est pas entier, le procédé qui nous permet de déterminer a est plus complexe car il nécessite de résoudre des équations à plusieurs inconnues. L'idée reste cependant la même. Par exemple, si l'on cherche à déterminer les solutions de $x^2 + x = \omega^\omega + 3$, on utilise le tableau suivant :

X	X^2	$X^2 + X$
ω^ω	$\omega^{\omega\cdot 2}$	$\omega^{\omega\cdot 2} + \omega^\omega$
$\omega^{\omega\cdot 2}$	$\omega^{\omega\cdot 4}$	$\omega^{\omega\cdot 4} + \omega^{\omega\cdot 2}$
$\omega^{\omega\cdot 3}$	$\omega^\omega \cdot 4$	$\omega^{\omega\cdot 3} + \omega^\omega \cdot 4$
$\omega^{\omega\cdot 4}$	$\omega^{\omega\cdot 3} \cdot 4$	$\omega^{\omega\cdot 4} + \omega^{\omega\cdot 3} \cdot 4$

  Ainsi, si $a = \omega^{\omega\cdot 4}\cdot 5 + \omega^{\omega\cdot 3}\cdot 10 + \omega^{\omega\cdot 2}\cdot 6 + \omega^\omega \cdot 4$, $a^2 + a = \omega^\omega$. Il a fallu résoudre un système d'équations afin de déterminer chacun des coefficients devant les $\omega^{\omega\cdot n}$. De cette façon, on se ramène au cas $x^2 + x = 3$ que l'on sait résoudre par le point précédent. On trouve, par cette méthode, les deux solutions de l'équation : $\omega^{\omega\cdot 4}\cdot 5 + \omega^{\omega\cdot 3}\cdot 10 + \omega^{\omega\cdot 2}\cdot 6 + \omega^\omega \cdot 4 + 6$ et $\omega^{\omega\cdot 4}\cdot 5 + \omega^{\omega\cdot 3}\cdot 10 + \omega^{\omega\cdot 2}6 + \omega^\omega \cdot 4 + 7$.

On a ainsi trouvé des méthodes pour déterminer les racines de polynômes de degré 2. Cela a notamment été possible car ω^{ω^ω} est un corps de caractéristique 2.

Les degrés supérieurs sont bien plus complexes et je n'ai su produire des résultats satisfaisants que pour le cas des racines p-ièmes d'entiers.

3.2 Calcul de racines p-ième d'entiers

On peut d'abord remarquer que puisque ω n'est que quadratiquement clos, il n'y a pas de raisons de penser que les racines p-ièmes d'entiers soient entières.

Par exemple, 2 admet pour racines cubiques $\omega, \omega \cdot 2$ et $\omega \cdot 3$. Cependant, les valeur des α_p peuvent nous permettre de restreindre l'ensemble auquel appartiennent ces racines.

On peut aussi remarquer qu'une fois connues les p racines p-ièmes de l'unité, on peut déduire d'une racine les $p-1$ autres. Pour déterminer les racines p-ièmes de l'unité, il suffit d'en connaître une non triviale puis de l'élever à toutes les puissances comprises entre 2 et $p-1$. Ainsi, puisque p est premier, on obtient les $p-2$ autres racines non triviales.

Il y a deux cas à distinguer, les entiers p tels que α_p est entier et ceux tels que α_p est infini.

- **Si α_p est entier** Certains entiers, à commencer par α_p n'ont aucune racine p-ième dans ω. Seul un entier sur p admettra d'ailleurs des racines entières puisque, si α_p est entier, on a $[x_p]^p$ entier, mais aussi $[{x_p}^n]^p$ avec $n \in [\![2, p-1]\!]$
  Ainsi, toutes les racines p-ièmes d'entiers dans ce cas-ci seront de la forme $[{x_p}^n \cdot m]$ avec $n \in [\![0, p-1]\!]$.

- **Si α_p est infini** Dans ce cas-ci, on peut penser que les entiers admettent logiquement des racines p-ièmes entières par définition de α_p. En effet, lorsque l'on ajoute x_p à l'ensemble étudié, ceux-ci admettent déjà des racines p-ièmes. En réalité, cela nous indique seulement que les racines p-ièmes des entiers sont inférieurs à α_p. Pour démontrer que celles-ci sont entières, on peut raisonner par l'absurde :
  Soit r une racine p-ième non entière d'un entier.
  $\exists k, p', n, m, r_1, r_2$ tels que $r = (2^{\omega^k \cdot {p'}^n})^m \cdot r_1 + r_2$
  $r^p = (2^{\omega^k \cdot {p'}^n})^{(m \times p) \mod p'} \cdot [r_1]^p + r_3$
  Or p et p' étant deux nombres premiers distincts et $m < p$, on a $(m \times p) \mod p' \neq 0$.
  r^p ne peut donc pas être entier, puisque r_1, r_2 et r_3 sont strictement inférieurs à $2^{\omega^k \cdot {p'}^n}$.
  Ainsi, toutes les racines p-ièmes d'entiers dans un tel cas sont entières.

3.3 Calcul de la taille d'orbites

Puisque j'ai principalement travaillé sur la recherche de racines de polynômes, je me suis finalement moins intéressé aux réseaux linéaires en eux-mêmes. Je n'ai donc traité que peu de cas lorsque l'on revient à ceux-ci.

En effet, je me suis principalement intéressé au cas où la matrice de transition M était diagonalisable. Dans de tels cas, la taille des orbites est aisément calculable.

Ainsi, si $(r_i)_{1 \leqslant i \leqslant n}$ sont les n valeurs propres de la matrice de transition et $(o_i)_{1 \leqslant i \leqslant n}$ leurs ordres associés.

$$M \sim \begin{pmatrix} \mathrm{r}_1 & 0 & \dots & 0 \\ 0 & & & \vdots \\ \vdots & & & 0 \\ 0 & \dots & 0 & \mathrm{r}_n \end{pmatrix}$$

Ainsi, M^{o_1} admet un vecteur propre de valeur propre 1. M admet donc une orbite de taille o_1. De la même façon, M admet des orbites de taille $(o_i)_{2 \leqslant i \leqslant n}$. Cependant, M admet aussi des orbites de taille les PPCM des ordres de ses valeurs propres. En effet, un vecteur combinaison linéaire des vecteurs propres des deux premiers sous espaces propres aurait une orbite de taille PPCM(o_1, o_1).

Conclusion

Je me suis finalement, au cours de ce stage, principalement intéressé à la recherche de racines de polynômes à coefficients dans $\omega^{\omega^{\omega}}$. Pour cela, je me suis d'abord familiarisé avec les ordinaux puis les nimbers. Cela m'a pris beaucoup de temps, mais m'a aussi permis d'aborder avec suffisamment de recul la partie la plus intéressante de mon sujet de stage.

Cependant, je n'ai été capable de produire des résultats que minimes sur les réseaux linéaires en eux-mêmes. En effet, si le cas des polynômes de degré 2 est entièrement traité, il ne relève en fait que des réseaux à deux états. Pour des réseaux de taille supérieure, j'ai seulement su restreindre l'ensemble de recherche des racines des polynômes dans des cas très précis.

C'est pourquoi il pourrait être pertinent de chercher à étendre le travail réalisé sur les polynômes de degré 2 aux degrés supérieurs. Il pourrait aussi être intéressant d'étudier les réseaux linéaires en eux mêmes pour, par exemple, savoir déterminer efficacement les vecteurs propres associés aux valeurs propres déjà trouvées. Enfin, il serait pertinent d'étudier la dynamique des réseaux linéaires lorsque la matrice de transition n'est pas diagonalisable, mais seulement trigonalisable.

Ce stage, comme première réelle expérience de la recherche a été plus qu'enrichissant. Il m'a tout d'abord permis de travailler sur un sujet qui me semblait intéressant, mais que je ne maîtrisait absolument pas. Christophe Papazian, mon maître de stage, m'a souvent aidé à prendre du recul sur mes démarches pour continuer à progresser dans mon travail et a su aiguiller mes recherches.

Cette expérience a aussi été l'occasion d'échanger avec de nombreux doctorants ou post-doctorant au sein de mon laboratoire. Même si mon sujet de stage était plutôt éloigné des leurs, l'équipe m'a très bien accueilli.

J'ai aussi eu la chance de pouvoir assister à la journée de l'équipe, séminaire au cours duquel des membres de l'équipe présentaient leurs travaux de l'an passé. Cela m'a permis de réaliser l'étendue des domaines étudiés dans une seule équipe d'informatique.

Enfin, ce stage a aussi été l'occasion pour moi d'échanger avec d'autres stagiaires, que ce soit au sein du laboratoire I3S ou avec d'autres stagiaires de l'INRIA. Je me suis aussi intéressé à leurs sujets et il est arrivé que nous y réfléchissions à plusieurs.

Références

[1] Conway, John H, *On numbers and games*. AK Peters/CRC Press 2nd Edition, Chapter 6, 2000

[2] Sierpiński, Wacław, *Lecons sur les nombres transfinis*. Gauthier-Villars et cie, Volume 30, 1928

[3] Neumann, János, *Zur einführung der transfiniten Zahlen*. Acta Litterarum ac Scien-

tiarum Regiae Universitatis Hungaricae Francisco-Josephinae : Sectio Scientiarum Mathematicarum, Volume 1, page 199-208, 1922

[4] Lenstra, Hendrik Willem and others, *On the algebraic closure of two*, 1977

[5] Lenstra, Hendrik Willem and others, *Nim multiplication*, 1978

Annexes

Correction de la forme normale de Cantor

Soient α et β deux ordinaux non nuls sous forme normale de Cantor :

- $\alpha = \sum_{i=0}^{k} \omega^{\alpha_i} \cdot n_i$
- $\beta = \sum_{j=0}^{l} \omega^{\beta_j} \cdot m_j$

L'addition

- Si $\beta_0 > \alpha_0$
  Par associativité de l'addition, on a $\alpha + \beta = (\alpha + \omega^{\beta_0} \cdot m_0) + \sum_{j=1}^{l} \omega^{\beta_j} \cdot m_j$
  Or $b > a$, donc $\omega^a + \omega^b = \omega^b$
  On en déduit, plus généralement, pour tout $n, m \in \omega$, $n > 0$ et $m > 0$, $b > a$ implique que $\omega^a \cdot n + \omega^b \cdot m = \omega^b \cdot m$
  En écrivant α sous sa forme normale de Cantor, on a $\forall i \leqslant k, \alpha_i < \beta_0$ implique que $\omega^{\alpha_i} \cdot n_i + \omega^{\beta_0} \cdot m_0 = \omega^{\beta_0} \cdot m_0$
  Par associativité de l'addition, on en déduit $\alpha + \beta = \beta$

- Si $\beta_0 < \alpha_0$
  $\omega^{\alpha_0} \cdot n_0 + (\sum_{i=1}^{k} \omega^{\alpha_i} \cdot n_i + \beta)$ si $\beta_0 < \alpha_0$
  Ce résultat est vrai par associativité de l'addition sur les ordinaux

- Puisque l'on a $\alpha_0 = \beta_0,\ \forall i, 1 \leqslant i \leqslant k, \alpha_i < \beta_0 \Rightarrow \omega^{\alpha_i} \cdot n_i + \omega^{\beta_0} \cdot m_0 = \omega^{\beta_0} \cdot m_0$
  En effet, α étant sous forme normale, la suite $\{\alpha_i\}_{0 \leqslant i \leqslant k}$ est strictement décroissante.
  On en déduit :

$$\begin{aligned} \alpha + \beta &= \omega^{\alpha_0} \cdot n_0 + \omega^{\alpha_0} \cdot m_0 + \sum_{j=1}^{l} \omega^{\beta_j} \cdot m_j \\ &= \omega^{\alpha_0} \cdot (n_0 + m_o) + \sum_{j=1}^{l} \omega^{\beta_j} \cdot m_j \end{aligned}$$

La multiplication

- $\alpha \cdot m = \sum_{i=1}^{m} \alpha$
  En utilisant les propriétés de l'addition démontrées ci-dessus, on a donc, $\alpha \cdot m = \omega^{\alpha_0} \cdot m + \sum_{i=1}^{k} \omega^{\alpha_i} \cdot n_i$

- $\alpha \cdot \beta = \sum_{j=0}^{l} \alpha \cdot \omega^{\beta_j} \cdot m_j$ par distributivité à gauche de la multiplication par rapport à l'addition. On distingue deux cas :
  - **Si b est fini**
    $$\begin{aligned} \alpha \cdot \omega^b &= \alpha \cdot \omega \cdot \omega^{b-1} \\ &= \sup\{\alpha \cdot m + \alpha \text{ tel que } m \in \omega\} \cdot \omega^{b-1} \\ &= \sup\{\alpha \cdot (m+1) \text{ tel que } m \in \omega\} \cdot \omega^{b-1} \\ &= \sup\{\omega^{\alpha_0} \cdot m + \sum_{i=1}^{k} \omega^{\alpha_i} \cdot n_i \text{ tel que } m \in \omega\} \cdot \omega^{b-1} \\ &= \omega^{\alpha_0+1} \cdot \omega^{b-1} \\ &= \omega^{\alpha_0+b} \end{aligned}$$
  - **Si b est infini**
    $$\begin{aligned} \alpha \cdot \omega^b &= \alpha \cdot \omega \cdot \omega^{b} \\ &= \sup\{\alpha \cdot m + \alpha \text{ tel que } m \in \omega\} \cdot \omega^{b} \\ &= \sup\{\alpha \cdot (m+1) \text{ tel que } m \in \omega\} \cdot \omega^{b} \\ &= \sup\{\omega^{\alpha_0} \cdot m + \sum_{i=1}^{k} \omega^{\alpha_i} \cdot n_i \text{ tel que } m \in \omega\} \cdot \omega^{b} \\ &= \omega^{\alpha_0+1} \cdot \omega^{b} \\ &= \omega^{\alpha_0+b} \end{aligned}$$

  On déduit de ces deux cas : $\alpha \cdot \omega^b \cdot m = \omega^{\alpha_0+b} \cdot m$.

L'exponentiation

- La première règle se démontre simplement par induction transfinie.
- La démonstration de cette règle est similaire à la seconde règle de la multiplication. En effet, l'exponentiation est à la multiplication ce que la multiplication est à l'addition.
- Cette règle se déduit simplement des formules sur la multiplication en décomposant la puissance comme produit de $m-1$ ordinaux.
- Cette règle se déduit aussi simplement des formules sur la multiplication. Cette règle n'est pas très utile en pratique et une simple exponentiation rapide est plus pertinente.
- Cette règle se déduit simplement de $n^\omega)\omega$:
  $$\begin{aligned} n^{\omega^s} &= \mathrm{n}^{\omega \cdot \omega^{s-1}} \\ &= (n^\omega)^{\omega^{s-1}} \\ &= \omega^{\omega^{s-1}} \end{aligned}$$
- Cette règle se déduit simplement de $n^\omega)\omega$:
  $$\begin{aligned} n^{\omega^\gamma} &= \mathrm{n}^{\omega \cdot \omega^{\gamma}} \\ &= (n^\omega)^{\omega^{\gamma}} \\ &= \omega^{\omega^{\gamma}} \end{aligned}$$

Code des opérations sur les nimbers

Cet algorithme correspond simplement à l'implémentation des formules démontrées pour la mise sous forme normale de Cantor des ordinaux et celles démontrées par J. H. Conway et H. W. Lenstra pour les nimbers.

```
type ordinaux =
        Entier of int
        | Omega
        | Produit of ordinaux * ordinaux
        | Somme of ordinaux * ordinaux
        | Puissance of ordinaux * ordinaux
        | Somme_nim of ordinaux * ordinaux
        | Produit_nim of ordinaux * ordinaux
        | Puissance_nim of ordinaux * int;;

let afficher alpha =
        let rec aux alpha = match alpha with
        | Entier n -> string_of_int n
        | Omega -> "w"
        | Somme(beta,gamma) -> "["^(aux beta)^"+"^(aux gamma)^"]"
        | Produit(beta,gamma) -> "["^(aux beta)^"*"^(aux gamma)^"]"
        | Puissance(beta,gamma) -> "["^(aux beta)^"^"^(aux gamma)^"]"
        | Somme_nim(beta,gamma) -> "("^(aux beta)^"+"^(aux gamma)^")"
        | Produit_nim(beta,gamma) -> "("^(aux beta)^"*"^(aux gamma)^")"
        | Puissance_nim(beta,gamma) -> "("^(aux beta)^"^"^(aux (Entier gamma
          ↪))^")"
        in print_string (aux alpha);;

let rec inserer_somme alpha beta = match alpha with
        | Somme(gamma,delta) -> Somme(gamma,(inserer_somme delta beta))
        | _ -> Somme(alpha,beta);;

let rec ordonner_somme alpha = match alpha with
        | Somme(beta,gamma) ->
                begin
                let beta' = ordonner_somme beta in
                let gamma' = ordonner_somme gamma in
                inserer_somme beta' gamma'
                end
        | _ -> alpha;;

let rec inserer_produit alpha beta = match alpha with
        | Produit(gamma,delta) -> Produit(gamma,(inserer_produit delta beta)
          ↪)
        | _ -> Produit(alpha,beta);;

let rec ordonner_produit alpha = match alpha with
        | Produit(beta,gamma) ->
                begin
                let beta' = ordonner_produit beta in
                let gamma' = ordonner_produit gamma in
                inserer_produit beta' gamma'
                end
        | _ -> alpha;;

let premier_terme alpha = match alpha with
        | Somme(beta,gamma) ->
                begin
                match beta with
                | Entier n -> Entier 0, Entier n
                | Omega -> Entier 1, Entier 1
                | Produit(Omega,coefficient) -> Entier 1, coefficient
                | Produit(Puissance(nombre,exposant),Entier n) -> exposant,
                  ↪Entier n
                | Puissance(nombre,exposant) -> exposant, Entier 1
```

```
                | _ -> failwith "Vous avez essay  de calculer le premier
                    ↪terme d'un ordinal non sous forme normale de Cantor"
                end
        | Omega -> Entier 1, Entier 1
        | Entier n -> Entier 0, Entier n
        | Produit(Omega,coefficient) -> Entier 1, coefficient
        | Puissance(nombre,exposant) -> exposant, Entier 1
        | Produit(Puissance(nombre,exposant),coefficient) -> exposant,
          ↪coefficient
        | _ -> failwith "Vous avez essay  de trouver le premier terme d'un
          ↪ordinal non sous forme de somme";;

let reste alpha = match alpha with
        | Somme(beta,gamma) -> gamma
        | _ -> Entier 0;;

let premiers = ref ([|2;3;5;7;11;13;17;19;23;29;31;37;41;43|]) ;;

let alpha_p = ref ([|   Entier 1; Entier 2; Entier 4; Somme(Omega,Entier 1);
    ↪ Somme(Puissance(Omega,Omega),Entier 1); Somme(Omega,Entier 4); Entier
    ↪16;
Somme(Puissance(Entier 2,Produit(Omega,Entier 3)),Entier 4); Somme(Puissance
    ↪(Entier 2,Puissance(Omega,Entier 4)),Entier 1);
Somme(Puissance(Entier 2,Puissance(Omega,Entier 3)),Entier 4); Somme(
    ↪Puissance(Entier 2,Puissance(Omega,Entier 2)),Entier 1);
Somme(Puissance(Entier 2,Produit(Omega,Entier 3)),Entier 4); Somme(Puissance
    ↪(Entier 2,Puissance(Omega,Entier 2)),Entier 1);
Somme(Puissance(Entier 2,Puissance(Omega,Entier 3)),Entier 1);|]) ;;

let rec element_neutre alpha = match alpha with
        | Somme(beta,gamma) ->
                begin
                let beta' = element_neutre beta in
                let gamma' = element_neutre gamma in
                match gamma' with
                | Entier 0 -> beta'
                | _ -> Somme(beta',gamma')
                end
        | Produit(beta,gamma) ->
                begin
                let beta' = element_neutre beta in
                let gamma' = element_neutre gamma in
                match gamma' with
                | Entier 0 -> Entier 0
                | Entier 1 -> beta'
                | _ -> if beta' = Entier 1 then gamma' else Produit(beta',
                    ↪gamma')
                end
        | Puissance(beta,gamma) ->
                begin
                let beta' = element_neutre beta in
                let gamma' = element_neutre gamma in
                match gamma' with
                | Entier 0 -> Entier 1
                | Entier 1 -> beta'
                | _ -> Puissance(beta',gamma')
                end
        | _ -> alpha;;

let rec difference alpha beta =
        let exposant1, coefficient1 = premier_terme alpha in
```

```
        let exposant2, coefficient2 = premier_terme beta in
        match exposant1, exposant2 with
        | Entier n1,Entier n2 ->        if n1=n2
                then (if coefficient1=coefficient2
                        then (if (reste alpha = Entier  0 && reste beta =
                            ↪Entier 0) then 0 else difference (reste alpha)
                            ↪(reste beta))
                        else (if coefficient1>coefficient2 then 1 else -1) )
                else (if n1>n2 then 1 else -1)
        | Entier n1,_ -> -1
        | _,Entier n2 -> 1
        | _ -> let b = difference exposant1 exposant2 in
                if b=0
                then
                        begin
                        if coefficient1 = coefficient2
                        then difference (reste alpha) (reste beta)
                        else
                                begin
                                if coefficient1>coefficient2
                                then 1
                                else -1
                                end
                        end
                else
                        b;;

let rec puissance_precedente n = match n with
        | 0 -> Entier 1
        | _ -> Produit(Puissance(Entier 2,Produit(Puissance(Omega,Entier 0),
          ↪Puissance(Entier 2,Entier (n-1)))),puissance_precedente (n-1))
          ↪;;

let rec element_entier alpha = match alpha with
        | Entier 0 -> false
        | _ ->  let exposant, coefficient = premier_terme alpha in
                ((difference exposant (Entier 0))=0) ||  element_entier (
                    ↪reste alpha);;

let premier_facteur alpha = match alpha with
        | Puissance(Puissance(Entier 2,Produit(Puissance(Omega,Entier k),
          ↪Puissance(Entier p,Entier n))),Entier m)  -> k, p, n, m
        | Produit(Puissance(Puissance(Entier 2,Produit(Puissance(Omega,
          ↪Entier k),Puissance(Entier p,Entier n))),Entier m),beta')  -> k
          ↪, p, n, m
        | _ -> failwith "Le terme n'est pas d compos  en produit de
          ↪puissance de khi";;

let facteurs_restants alpha = match alpha with
        | Produit(beta,gamma) -> gamma
        | _  -> Entier 1;;

let rec distribuer_gauche alpha beta = match beta with
        | Somme(gamma,delta) -> Somme(simplifier (Produit(alpha,gamma)), (
          ↪distribuer_gauche alpha delta))
        | gamma -> (simplifier (Produit(alpha, gamma)))

and decomposer_puissance alpha beta =
        let exposant, coefficient = premier_terme beta in
        let a =         if difference exposant (Entier 0) = 1
```

```
                then simplifier (Puissance(simplifier (Puissance(alpha,
                    ↪Puissance(Omega,exposant))),coefficient))
                else simplifier (Puissance(alpha,coefficient))
        in
        let b = reste beta in
        if b = (Entier 0)
        then a
        else
                begin
                let c = decomposer_puissance alpha b in
                distribuer_gauche a c
                end

and expo_rapide x n =
        if n = 0
        then Entier 1
        else
                begin
                let y = expo_rapide x (n/2) in
                if n mod 2 = 0 then simplifier (Produit(y,y)) else
                    ↪simplifier (Produit(simplifier (Produit(y,y)), x))
                end

and simplifier alpha = match alpha with
        | Produit(beta,gamma) ->
                begin
                match beta, gamma with
                | Entier n, Entier n' -> Entier (n*n')
                | Entier 0, _ -> Entier 0
                | _, Entier 0 -> Entier 0
                | _, Entier 1 -> beta
                | _ ->
                        let exposant, coefficient = premier_terme beta in
                        let exposant', coefficient' = premier_terme gamma in
                        if difference exposant' (Entier 0)=1
                        then Produit(Puissance(Omega,simplifier (Somme(
                            ↪exposant,exposant'))),coefficient')
                        else Somme(Produit(Puissance(Omega,exposant),
                            ↪coefficient'),reste beta)
                end
        | Somme(beta,gamma) ->
                begin
                match beta, gamma with
                | Entier n, Entier n' -> Entier (n+n')
                | _, Entier 0 -> beta
                | _ ->
                        let exposant, coefficient = premier_terme beta in
                        let exposant', coefficient' = premier_terme gamma in
                        let dif = difference exposant exposant' in
                        if dif = 1
                        then Somme(Produit(Puissance(Omega,exposant),
                            ↪coefficient),simplifier (Somme(reste beta,gamma
                            ↪)))
                        else
                                begin
                                if dif =0
                                then Somme(Produit(Puissance(Omega,exposant)
                                    ↪,simplifier (Somme(coefficient,
                                    ↪coefficient'))),reste gamma)
                                else gamma
                                end
```

```
        end
| Puissance(beta,gamma) ->
        begin
        match gamma      with
        | Entier 0 -> Entier 1
        | Entier 1 -> beta
        | Entier n ->
                begin
                if element_entier beta
                then expo_rapide beta n
                else
                        begin
                        let exposant, coefficient = premier_terme
                            ↪beta in
                        let a = simplifier (Produit(exposant,Entier(
                            ↪n-1))) in
                        let b = simplifier (Puissance(Omega,a)) in
                        simplifier (Produit(b,beta))
                        end
                end
        | _ ->
                begin
                let exposant, coefficient = premier_terme beta in
                let exposant', coefficient' = premier_terme gamma in
                match exposant with
                | Entier 0 ->
                        begin
                        match exposant' with
                        | Entier n -> Puissance(Omega,Puissance(
                            ↪Omega,Entier (n-1)))
                        | _ -> Puissance(Omega,Puissance(Omega,
                            ↪exposant'))
                        end
                | _ -> Puissance(Omega,simplifier (Produit(exposant,
                    ↪Produit(Puissance(Omega,exposant'),coefficient
                    ↪'))))
                end
        end
| Somme_nim(beta,gamma) ->
        if beta = Entier 0
        then forme_normale gamma
        else

        begin

        if gamma = Entier 0
        then forme_normale beta
        else
                begin
                let exposant, coefficient = premier_terme beta in
                let exposant', coefficient' = premier_terme gamma in
                if coefficient = Entier 0
                then simplifier(Somme_nim(reste beta, gamma))
                else

                begin

                if coefficient' = Entier 0
                then simplifier (Somme_nim(beta, reste gamma))
                else
                        begin
```

```
                        match (difference exposant exposant') with
                        | 0 -> simplifier (Somme_nim(reste beta,
                            ↪reste gamma))
                        | 1 -> simplifier (Somme(forme_normale (
                            ↪Produit(Puissance(Entier 2,exposant),
                            ↪coefficient)),simplifier (Somme_nim(
                            ↪reste beta,gamma))))
                        | -1 -> simplifier (Somme(forme_normale (
                            ↪Produit(Puissance(Entier 2,exposant'),
                            ↪coefficient')),simplifier (Somme_nim(
                            ↪beta, reste gamma))))
                        end

                end

                end

        end

| Produit_nim(beta,gamma) ->
        if beta = Entier 1
        then gamma
        else

        begin

        if gamma = Entier 1
        then beta
        else
                begin
                let k, p, n, m = premier_facteur beta in
                let k', p', n', m' = premier_facteur gamma in
                if (p>p' || (p=p' && n>n'))
                then forme_normale (Produit(Puissance(Puissance(
                    ↪Entier 2,Produit(Puissance(Omega,Entier k),
                    ↪Puissance(Entier p,Entier n))),Entier m),
                    ↪simplifier (Produit_nim(facteurs_restants beta,
                    ↪gamma))))
                else

                begin

                if (p'>p || (p=p' && n'>n))
                then  forme_normale (Produit(Puissance(Puissance(
                    ↪Entier 2,Produit(Puissance(Omega,Entier k'),
                    ↪Puissance(Entier p',Entier n'))),Entier m'),
                    ↪simplifier (Produit_nim(beta, facteurs_restants
                    ↪ gamma))))
                else
                        begin
                        if (m+m'<p)
                        then forme_normale ( Produit(Puissance(
                            ↪Puissance(Entier 2,Produit(Puissance(
                            ↪Omega,Entier k'),Puissance(Entier p',
                            ↪Entier n'))),Entier (m+m')), simplifier
                            ↪ (Produit_nim(facteurs_restants beta,
                            ↪facteurs_restants gamma))))
                        else

                        begin
```

```
if k=0
then
        begin
        let a = simplifier (Produit_nim(
            ↪facteurs_restants beta,
            ↪facteurs_restants gamma)) in
        let b = forme_normale (Somme(
            ↪puissance_precedente n,
            ↪Puissance(Entier 2,Puissance(
            ↪Entier 2,Entier n)))) in
        let c  = forme_normale (Produit_nim(
            ↪Somme(puissance_precedente n,
            ↪Puissance(Entier 2,Puissance(
            ↪Entier 2,Entier n))),a)) in
        c
        end
else
        begin
        if n!=0
        then
                begin
                let a = Puissance(Entier 2,
                    ↪Produit(Puissance(Omega
                    ↪,Entier k),Puissance(
                    ↪Entier p, Entier (n-1))
                    ↪)) in
                let b = Puissance(Puissance(
                    ↪Entier 2,Produit(
                    ↪Puissance(Omega,Entier
                    ↪k),Puissance(Entier p,
                    ↪Entier (n)))),Entier ((
                    ↪m+m') mod p)) in
                forme_normale (Produit(b,
                    ↪Produit_nim(a,
                    ↪simplifier (Produit_nim
                    ↪(facteurs_restants beta
                    ↪, facteurs_restants
                    ↪gamma)))))
                end
        else
                begin
                let a = !alpha_p.(k) in
                let b = Puissance(Puissance(
                    ↪Entier 2,Produit(
                    ↪Puissance(Omega,Entier
                    ↪k),Puissance(Entier p,
                    ↪Entier (n)))),Entier ((
                    ↪m+m') mod p)) in
                forme_normale (Produit(b,
                    ↪Produit_nim(a,
                    ↪simplifier (Produit_nim
                    ↪(facteurs_restants beta
                    ↪, facteurs_restants
                    ↪gamma)))))
                end
        end

end
```

```
                        end

                        end

                end

        | _ -> alpha

and forme_normale alpha = match alpha with
        | Entier n -> Entier n
        | Omega -> Omega
        | Produit(beta,gamma) ->
                begin
                        let beta' = forme_normale beta in
                        let gamma' = forme_normale gamma in
                        distribuer_gauche beta' gamma'
                end
        | Somme(beta,gamma) ->
                begin
                        let beta' = forme_normale beta in
                        let gamma' = forme_normale gamma in
                        simplifier (Somme(beta',gamma'))
                end
        | Puissance(beta,gamma) ->
                begin
                        let beta' = forme_normale beta in
                        let gamma' = forme_normale gamma in
                        decomposer_puissance beta' gamma'
                end
        | Somme_nim(beta,gamma) ->
                begin
                        let beta' = forme_normale beta in
                        let gamma' = forme_normale gamma in
                        let beta'' = decomposition_binaire beta' in
                        let gamma'' = decomposition_binaire gamma' in
                        simplifier (Somme_nim(beta'',gamma''))
                end
        | Produit_nim(beta,gamma) ->
                begin
                let beta' = forme_normale beta in
                let gamma' = forme_normale gamma in
                if difference beta' (Entier 1) <=0 || difference gamma' (
                    ↪Entier 1) <= 0
                then simplifier (Produit(beta',gamma'))
                else
                        begin
                        let beta'' = decomposition_binaire beta' in
                        let gamma'' = decomposition_binaire gamma' in
                        let x = developper beta'' gamma'' in
                        x
                        end
                end
        | Puissance_nim(beta,n) ->
                if n=0
                then Entier 1
                else
                        begin
                        let x = forme_normale (Puissance_nim(beta,(n/2))) in
                        if n mod 2 = 0
                        then forme_normale (Produit_nim(x,x))
```

```
                        else forme_normale (Produit_nim(beta,Produit_nim(x,x
                            ↪)))
                        end

and developper alpha beta =
        let rec aux alpha beta =
                match beta with
                | Somme(gamma,delta) -> let exposant, coefficient =
                    ↪premier_terme gamma in
                        let gamma' = decomposition_premier exposant in
                        if coefficient = Entier 0
                        then aux alpha delta
                        else
                                begin
                                let resultat = simplifier (Produit_nim(alpha
                                    ↪,gamma')) in
                                let c = aux alpha delta in
                                forme_normale (Somme_nim(resultat,c))
                                end
                | gamma ->      let exposant, coefficient = premier_terme
                    ↪gamma in
                        let gamma' = decomposition_premier exposant in
                        if coefficient = Entier 0
                        then Entier 0
                        else
                                begin
                                let resultat = simplifier (Produit_nim(alpha
                                    ↪,gamma')) in
                                resultat
                                end
        in match alpha with
                | Somme(gamma,delta) ->         let exposant, coefficient =
                    ↪premier_terme gamma in
                        let gamma' = decomposition_premier exposant in
                        if coefficient = Entier 0
                        then developper delta beta
                        else forme_normale (Somme_nim(aux gamma' beta,
                            ↪developper delta beta))
                | gamma ->      let exposant, coefficient = premier_terme
                    ↪gamma in
                        if coefficient = Entier 0
                        then Entier 0
                        else
                                begin
                                let gamma' = decomposition_premier exposant
                                    ↪in
                                aux gamma' beta
                                end

and distribuer_binaire exposant dec = match dec with
        | Somme(Produit(Puissance(Entier 2, exposant'),coefficient),dec') ->
            ↪ if coefficient = Entier 0 then distribuer_binaire exposant dec
            ↪' else Somme( Produit(Puissance(Entier 2, forme_normale (Somme(
            ↪exposant,exposant'))),coefficient), distribuer_binaire exposant
            ↪ dec')
        | Produit(Puissance(Entier 2, exposant'),coefficient) -> Produit(
            ↪Puissance(Entier 2,(forme_normale (Somme(exposant,exposant'))))
            ↪,coefficient)
        | _ ->afficher dec; failwith "Vous avez essay  de distribuer en
            ↪binaire avec de mauvais param tres"
```

```
and distribuer_normale exposant dec = match dec with
        | Somme(Produit(Puissance(Entier n, exposant'),coefficient),dec') ->
                if coefficient = Entier 0
                then distribuer_normale exposant dec'
                else Produit(Puissance(Puissance(Entier 2,((Produit(
                    ↪Puissance(Omega,exposant),Puissance(Entier n, exposant
                    ↪'))))),coefficient), distribuer_normale exposant dec')
        | Produit(Puissance(Entier n, exposant'),coefficient) -> Puissance(
            ↪Puissance(Entier 2,((Produit(Puissance(Omega,exposant),
            ↪Puissance(Entier n, exposant')))),coefficient)
        | _ ->afficher dec; failwith "Vous avez essay  de distribuer en
            ↪produits de ??    ?khi avec de mauvais param tres"

and decomposition_binaire alpha =
        let rec aux alpha =
                let exposant, coefficient = premier_terme alpha in
                let dec = decomposition_entier coefficient 2 in
                let beta = reste alpha in match beta with
                        | Entier 0 -> distribuer_binaire (Produit(Omega,
                            ↪exposant)) dec
                        | beta -> Somme((distribuer_binaire ( (Produit(Omega
                            ↪,exposant))) dec),aux beta)
        in ordonner_somme ((aux alpha))

and decomposition_premier alpha =
        let rec aux alpha =
                let exposant, coefficient = premier_terme alpha in
                match exposant with
                        | Entier exposant ->
                                begin
                                let base = !premiers.(exposant) in
                                let dec = decomposition_entier coefficient
                                    ↪base in
                                let beta = reste alpha in
                                match beta with
                                        | Entier 0 -> distribuer_normale (
                                            ↪Entier exposant) dec
                                        | _ -> Produit(distribuer_normale (
                                            ↪Entier exposant) dec,aux beta)
                                end
                        | _ -> failwith "Vous essayez de r aliser des
                            ↪op rations sur des ordianaux n'appartenant pas
                            ↪    w^w^w"
        in ordonner_produit ((aux alpha))

and decomposition_entier n base =
        let rec aux calcul exposant n base = match n with
                | Entier m when m<base ->        if calcul = (Entier 0)
                        then Produit(Puissance(Entier base, Entier exposant)
                            ↪,Entier m)
                        else Somme(Produit(Puissance(Entier base, Entier
                            ↪exposant),Entier m),calcul)
                | Entier m ->    if calcul = Entier 0
                        then aux (Produit(Puissance(Entier base, Entier
                            ↪exposant),Entier (m mod base))) (exposant+1) (
                            ↪Entier (m/base)) base
                        else aux (Somme(Produit(Puissance(Entier base,
                            ↪Entier exposant),Entier (m mod base)),calcul))
                            ↪(exposant+1) (Entier (m/base)) base
                | _ -> failwith "Vous avez essay  de d composer autre
                    ↪chose qu'en entier en base enti re"
```

```
        in (aux (Entier 0) 0 n base)

and calcul alpha = afficher alpha; print_string " = "; afficher (
    ↪element_neutre (forme_normale alpha));;
```