Torem ipsum repreHEnderit, consectetur

aDipiscing elAt, sed tempoRinKididunt

utSabore et Iolore uD magna. Enim ad

veniam, quis nOstrud exercitation Felitum

FugiatlabORisCINt ut aliGuip

ex sunt in culpa qui officia deserunt est XIV.

この文章はダミーテキストである

淡中 圏

この文章はダミーテキストである。

ダミーテキストとは何か。それは、まだ文章が決まっていない場所を、とりあえず埋めておくためのプレースホルダーである。

例えば雑誌の誌面や広告や Web ページなどで、まだ正式な文章が出来ていない状態でヴィジュアルのデザインをしなくてはいけない場合がある。

そんな時に「ここには何かの文章が入ります」という意味で使われる、適当な文章、場合によっては無意味な「文章らしき文字列」が、ダミー文字列である。

デザインをする際や確認する際に、文章が入っている場所には文章が入っている方が分かりやすい。とは言え、そこに読める文章があると読んでしまうのが人の性で、そうするとデザインに注意が向かないかもしれない。なので、そこに読めない文章が入れるのは合理的である。

欧米圏で有名なのが、**lorem ipsum** であろう。これは 1960 年代から、広告業界などで使われているようだ。

これは次のようなものである。

> Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

これはラテン語風だが、意味のある文章ではない。なお、このような意味不明なダミーテキストを英語で **greeking** と読んだりするが、ギリシャ文字が使われているケースは見たことはない（目的を考えたら当然だが）。

ちなみにこの文字列はキケロの著作『De finibus bonorum et malorum』（善と悪の究極について）から抜き出しているらしい。

日本語では、それっぽい無意味な文章を書くことがラテン文字と比べると難しいのか、よく著作権が切れた小説が使われているケースをよく見る気がする。

さて、これでダミーテキストとは何か、ということが共有できたので、もう一度この文章の冒頭に戻ることができる。

念のためにもう一度書いておくと、この文章はダミーテキストなのである。

実は、この同人誌のデータには、元々この部分に lorem ipsum が挿入されて「この部分

に前書きを書く」という意味にしているのである。もちろん、それがそのまま読者の目に触れることはない。

しかし、この同人誌を作成中にあることに気づいたのである。この同人誌の表紙は lorem ipsum を改変したものを使っている。それは、この小説内で擬似乱数生成アルゴリズムとして有名な**メルセンヌ・ツイスター**について解説しているので、ランダムっぽい表紙にしたら面白かろうと思ったからだ。

また、付録として、「ぞうの卵はおいしいぞう」をたくさん印字する TEX マクロについて簡単な解説を書いている。あまりに書くことがなくて、適当に紙面を埋めるネタがないかと探していた時に、有名な科研費 LATEX のダミーテキストのことを思い出したからだ。

この二つは全く別の経路から考えたものであったが、さて前書きに何を書こうかと思ったときに、はたと気づいた。今回の隠れたテーマは「ダミーテキスト」であったのだ、ということに。

では、前書きもダミーテキストについて書いてしまおう。そしてその文章がダミーテキスト自身であったらもっと面白いだろう、と考えるまでは、小さなステップであった。

ところがここまで来て、問題にぶつかった。果たしてこれは本当にダミーテキストなのだろうか。

既に文章は 2 ページ目の真ん中あたりまで来ている。今更 1 ページ目に引き返すことはできない。しかし、本来前書きは 1 ページくらいを想定していて、だからこそ lotem ipsum も 1 ページで収まっていたのだ。なのに、1 ページを想定している前書きのダミーテキストが 2 ページではそもそも本末転倒である。しかももう一つ本末転倒なことに、ダミーテキストなのに、読者はなんと 2 ページ目の後半までこの文章をしっかり読んでしまっているのだ。

本来は義務教育で「この文章はダミーテキストである」などと書いてある文章を決して読み込んではいけないことくらい教えてしかるべきであろうが、今の義務教育にそんなことを期待はできない。

さらに言えば、この文章はどう考えても数学となんの関係もない。ダミーテキストとしてはそれでいいのかも知れないが、これを実質前書きとして読んでいる読者は困惑せざるを得ないだろう。数学の同人誌なのに、この前書き、数学について何も語っていないではないか。

書こうと思えば、ここで何か一つ「数学っぽい話」をでっち上げることは可能であろう。例えばこの文章は一種の「自己言及」になっている。ダミーテキスト自身がダミーテキストについて語り、自分がダミーテキストだと主張している。この手の自己言及構造は、実際形式論理学などでも昔からの定番の話題である。

しかし、それについて既にいろいろな面白い話が他の場所で語られているし、それについて生半可なこともたくさん語られている。ここにそのような生半可なことを書いても、この同人誌の価値は上がらないし、むしろ怪しまれてしまう可能性がある。だから、敢えてそのようなことは避けたい。

つまり八方塞がりなのだ。

どうしたらいいのだろう。このままでは 3 ページ目に突入してしまうではないか。それは避けたい。なのでこの前書きは特に何も提示しないまま終わる。

果たしてこんな前書きのある同人誌を買ってくれるのだろうか。せめてもの願いとしては、読者が決してこの文章を真面目に読み込まないことである。

しつこいようだが、何度でも言おう。この文章はダミーテキストである。ダミーテキストとは読む必要がないし、できれば読まない方が良い文章なのだ。

もしあなたがこの文章を読んでいるならば、決してこの文章を真面目に読まないで欲しい。

そしてここからは、ちゃんと中身のある話が始まるので安心して欲しい。

参考文献

[1] Wlipedia 日本語版 lorem ipsum./https://ja.wikipedia.org/wiki/Lorem_ipsum.(2019/12/22 access)

目次

この文章はダミーテキストである i

参考文献 iii

第 1 章 メルセンヌ・ツイスターとフロベニウス写像 1
1.1 イントロダクション 1
1.2 メルセンヌ・ツイスターの仕組み 2
1.3 フロベニウス自己準同型写像との関係 9
1.4 付録:inversive-decimation method の Ruby コード 10

参考文献 13

第 2 章 鳩の巣原理小説 15
2.1 鳩と存在 15
2.2 鳩と実数 16
2.3 鳩と無限 19

参考文献 21

第 3 章 エッセイ: 機械化された数学的知識と共に生きる 23

参考文献 30

おまけ:ぞうの卵はおいしいぞう 31

参考文献 40

第 1 章

メルセンヌ・ツイスターとフロベニウス写像

淡中 圏

まず、現在使われているメルセンヌ・ツイスターが $\mathbb{F}_2$ 上の簡単な線形代数と初歩の体論（特に有限体）と群論を基礎にしていることを説明する。そして次に、アルゴリズムが長い周期を実現するパラメーターを選ぶための手法が、有限体上の初歩のガロア理論と関係があることを説明する。

1.1 イントロダクション

現在多くのプログラミング言語に移植されている擬似乱数生成器として、**メルセンヌ・ツイスター**がある。

メルセンヌ・ツイスターは任意の bit 数に対して使えるが、32 ビットの整数型に対する実装である **mt19937** がよく使われている。これは $2^{19937}-1$ という長い周期と値が 623 次元に均等分布するという良い性質を持ち、また高速に計算できるため、様々な応用に十分耐えられる[*1]。

メルセンヌ・ツイスターの原理には有限体 $\mathbb{F}_2$ 上の線形代数が活かされていることは、[1] に詳しく書いてある。

しかし、さらに深く突っ込めば、実はメルセンヌ・ツイスターは初歩のガロア理論にも関係がある。松本氏による簡単な「気持ち」の表明が [**?**] にある。

ただし、上記の Web ページには簡単な説明しか書いていないので、もう少し詳しい説明が欲しい。

そこでガロア理論が具体的にどこに使われているかを少し見てみたい。

メルセンヌ・ツイスターの原論文 [4] には、メルセンヌ・ツイスターを計算するアルゴリズムだけではなく、メルセンヌ・ツイスターの候補を定めるパラメータについて、与えら

[*1] ただし過去の出力から状態を復元でき、その状態から次の値を予測可能なので、暗号には向かない。ちなみに、過去の出力から状態を復元できることは、この文章で説明する inversive-decimation method がうまくいく理由でもある。つまり、良い性質と悪い性質がここで絡み合っていて少し悩ましい。

れたパラメータが長い周期を実現するかどうかを判定するアルゴリズムも書かれてある。

実はこのアルゴリズムこそ（原論文 [4] の中でははっきりとは指摘されていないが）ガロア理論に届くもう少し深い内容が使われているのだ（それがまさに [**?**] に記載されたフロベニウス写像の位数に関する話なのだ）。

論文中ではそれは **inversive-decimation method** と呼ばれている。これのおかげで、そのパラメータが長い周期を実現するかどうかの判定にかかる計算量が大きく減らせるのだ。

mt19937 のパラメータを決定するのにもこれが使われたようで、これがなければ当時計算に数年かかってしまったかもしれないパラメータ探索が数日で終わったようだ。

メルセンヌ・ツイスターだけに限らず $\mathbb{F}_2$ 上の線形代数は擬似乱数の基礎理論であり、$\mathbb{F}_2$ 上のガロア理論も、擬似乱数を理解する上で重要になるはずである。暗号や符号理論では既にそうなっていると私は認識している。

そこで、それほど複雑ではないメルセンヌ・ツイスターの場合を使って、初歩のガロア理論がメルセンヌ・ツイスターの中でどう応用されているかを見てみよう。

1.2　メルセンヌ・ツイスターの仕組み

ここではメルセンヌ・ツイスターの仕組みを、大幅に簡略化して説明する。より詳しい説明は、[1] を、完全な説明は [4] を参照して欲しい。

1.2.1　乱数生成の仕組み

C 言語の int など、計算機の扱う整数型とは結局のところ有限のビット列であることが多い[*2]。それには当然上限と下限があり、オーバーフローすると 0 に戻ったり負の数になったりするので、数学的な意味での整数環ではない。

しかし、これを $\mathbb{F}_2$ 上のベクトル空間の元と見るとしたら、そのどこにも不正確さはないだろう。

と言うわけで、乱数を生成するにも $\mathbb{F}_2$ 上のベクトル空間を考えることが筋がいいのでは、と言う推論ができる（実際にはこれに到るにはそこそこ複雑な歴史があるようだ[*3]）。

例えば 32 ビットの整数型（**ワード**と呼ぶことにする[*4]）は $\mathbb{F}_2$ の 32 次元ベクトル空間だと考えることができる。その場合、ビット操作の xor がベクトルとしての足し算に対応する。

[*2] 任意精度の整数などはもっと複雑な仕組みである。

[*3] [3] に
（乱数の大御所の Knuth, Compagner, Marsaglia らが、
「整数の方が 2 元体より乱数生成に適している」
と主張しつづけており、MT の発表後も、
古典的なアルゴリズムを推奨し続けた。）
という記述がある。
詳しくは知らない。

[*4] ワードとはコンピュータ内部の情報の単位を表す、あまり厳密に定義されていない用語である。コンピュータのプロセッサの汎用レジスタのサイズをワードと言う場合も多い。すると 32 ビットを 1 ワードとする 32 ビットマシンの時代が長く続き、メルセンヌ・ツイスターの当時もそうであったので、現在もメルセンヌ・ツイスターは 1 ワード 32 ビットとして実装されている。ただし、現在のマシンはほぼ 64 ビットマシンであり、128 ビットマシンも登場している。

乱数生成器は、一回動かすごとにこの 32 次元の $\mathbb{F}_2$ ベクトル空間の元を一つずつ出力する。

そのためには乱数生成器は状態を持たなくてはいけない。状態もまた、$\mathbb{F}_2$ 上のベクトル空間にしようと考えるのは自然だ。

まずこの**状態空間**がそこそこ大きい必要がある。

例えばこの状態空間と出力が同じだとする。つまり、一つ前の出力を覚えておいて、そこから次の出力を計算する、と言うことだ。そうすると状態空間の濃度は 2^{32} である。そうすると、この乱数の周期は必ず 2^{32} 以下になる。なぜなら**鳩の巣原理**により $2^{32}+1$ 回乱数生成器を動かすと、必ず同じ状態が二回現れてしまう。その二つを n 回目と m 回目（$n < m$）だとすると、m 以降は必ず n の後と同じ乱数列が現れるので、$m-n$ が周期になるはずである[*5]。

状態空間を大きくすることは別に難しくない。より大きい次元の $\mathbb{F}_2$ ベクトル空間を状態空間とすればいいだけである。例えば 32 次元の $\mathbb{F}_2$ ベクトル空間の元をたくさん並べたものを使えばいい。より、現実のコンピュータに引き付けて書けば、ワードの配列を状態空間として使えばいいのだ。しかし、それだけでは周期が長いことを保証はできない。

周期を考えるために、乱数を計算するアルゴリズムの細部を見るより、抽象的な関数として考えたい。特に、せっかく状態空間を $\mathbb{F}_2$ 上のベクトル空間と考えたのだから、この状態空間の遷移関数も $\mathbb{F}_2$ 上の線形写像にしたいと考えるのが人情というものであろう。

状態空間を $\mathbb{F}_2$ 上の n 次ベクトル空間 V とし、状態空間の遷移関数を線形写像 $f : V \to V$ とする。

この f の作用の仕方を見るために、一変数多項式環 $\mathbb{F}_2[T]$ を考える。そして、f の固有多項式を ϕ とすると、ケーリー・ハミルトンの定理により、

$$\phi(f) = 0$$

が成り立つ。

もし、ϕ が規約ならば、ϕ は f の最小多項式にもなり、

$$\mathbb{F}_2(f) \cong \mathbb{F}_2[T]/(\phi) \cong \mathbb{F}_{2^n}$$

が成り立つ。ここで、$\mathbb{F}_2(f)$ は $\mathbb{F}_2$ と f から生成される体、$\mathbb{F}_2[T]/(\phi)$ は多項式環 $\mathbb{F}_2[T]$ をイデアル (ϕ) で割った環、そして $\mathbb{F}_{2^n}$ を $\mathbb{F}_2$ を n 次有限拡大した体である[*6]。

そしてよく知られているように、有限体の乗法群は巡回群になる。特にこの場合は、乗法群 $\mathbb{F}_{2^n}^*$ は位数 2^n-1 の巡回群になるわけである。

f はこの巡回群の要素であることから、f の周期、すなわち群論の意味での f の位数は、ラグランジュの定理により、2^n-1 の約数になることがわかる。

これでは ϕ の規約性を示しても、それだけでは f の周期が長い保証はできないようだ。しかし、この n をうまく選べばどうだろうか。

通常状態空間は、ビットの塊である整数型を並べたものであったので、この n は 32 の倍数であった。

[*5] ここで証明されているのは正確には準周期的と言う性質である。つまりある n_0 があり、n_0 以降は周期的になる。

[*6] これは同型を除いて唯一である。

メルセンヌ・ツイスターの主要なアイディアの一つは、いくつかのビットを使わないと言うものだった。mt19937 では、624 個の 32 ビット長のベクトル（つまりワード）を並べたものを状態として持つ。しかし、一番最初のベクトルは、次の状態空間を計算するのに 1 ビットしか使われず 31 ビットは捨てられてしまう。

なので、状態空間は実際には $32\times624-31$ で 19937 次元になっている。これが mt19937 の名前の由来である。

そしてなぜ 19937 なのかと言うと、これは**メルセンヌ指数**と呼ばれる数になっている素数だからである。つまり、2^p-1 が素数となる p をメルセンヌ指数といい、この時 2^p-1 自体を**メルセンヌ（素）数**と呼ぶのだ。ちなみに 2^p-1 が素数である時、実は p 自体も素数である。

ここまでの議論を思い出すと、状態空間 V が $\mathbb{F}_2$ 上の 19937 次元ベクトル空間で遷移関数 f が線形写像 $V\to V$ で、その固有多項式 ϕ が既約ならば、F の周期は $2^{19937}-1$ の約数になる。しかし、ここで $2^{19937}-1$ は素数なので、これは 1 か $2^{19937}-1$ のどちらかになる。

これにより、周期が長いことの保証が圧倒的にシンプルになるのだ。

そしてこの周期は十分長いので、多少ビットを削って状態空間を小さくして、可能な最大の周期を短くしても、全く問題はないのだ。

頭いいなあ。

1.2.2　Inversive-Decimation Method の仕組み

あとは f の固有多項式 ϕ の既約性を判定すればいいのだが、これは実際には簡単では無い。

一番簡単な方法は $p=2^{19937}-1$ としたときに、

$$\begin{cases} t^2 \not\equiv t \mod \phi \\ t^{2^p} \equiv t \mod \phi \end{cases}$$

を確かめることである。

これには $O(lp^2)$ の計算量がかかってしまう。ここで、l は ϕ の係数が 0 でない項の数である。

確かめてみよう。

証明　このような冪乗の計算は、2 乗をし続ければいいので、$\log_2 2^p=p$ 回 2 乗をすることになる。一回の 2 乗するごとに、ϕ で　mod　をとるので、$2p$ 次の多項式を ϕ で割る必要があるが、これは要は $O(p)$ 回の ϕ の引き算であり、その引き算には毎回 $O(l)$ の計算量がかかる。よって最終的に計算量は $O(lp^2)$ になるわけである。

これでパラメーターを探そうとすると、[4] の記述によると、メルセンヌ・ツイスターの開発当時の計算機では何年もの時間がかかってしまいそうだったようだ。

そこで [4] において考案されたのが **inversive-decimation method** である。

これには一つのパラメータが長い周期を実現するかどうかチェックするのに $O(p^2)$ の計算量しかかからない。p 次多項式 ϕ の 0 でない項の数 l が、大体 p と一緒に増えることを考えれば、これは 3 次式が 2 次式になったと考えられ、大きな進歩である。

では [4] の記述にしたがって、この手法について説明してみよう。

S^∞ を $\mathbb{F}_2$ の無限列のなすベクトル空間、

$$S^\infty := \{\chi = (\ldots, x_5, x_4, x_3, x_2, x_1, x_0) | x_i \in \mathbb{F}_2\}$$

とする。

二つの線形写像 $S^\infty \to S^\infty$ 遅延オペレータ D と間引きオペレータ H[*7]を、

$$D(\ldots, x_4, x_3, x_2, x_1, x_0) = (\ldots, x_5, x_4, x_3, x_2, x_1)$$
$$H(\ldots, x_4, x_3, x_2, x_1, x_0) = (\ldots, x_8, x_6, x_4, x_2, x_0)$$

として定義する。

次が証明したい定理である。（ [4] の Theorem 4.1.）

定理 1 p をメルセンヌ指数とし、ϕ を $\mathbb{F}_2$ 上の p 次多項式とする。ϕ が既約てあることと、$\chi \in S^\infty$ が存在して以下の条件を満たすことは同値である。

1. $\phi(D)\chi = 0$
2. $H\chi \neq \chi$
3. $H^p\chi = \chi$

$\phi(t)$ を線形漸化式の特性多項式だとする。

線形漸化式とは、

$$a_n x_{k+n} + a_{n-1}x_{k+n-1} + \cdots + a_1 x_{k+1} + a_0 x_k = 0$$

という形式のものであり、これにより、$x_k, x_{k+1}, \ldots, x_{k+n-1}$ から、x_{k+n} が計算できる。

そしてこの場合の特性多項式とは変数を λ をして、

$$a_n\lambda^n + a_{n-1}\lambda^{n-1} + \cdots + a_1\lambda^+ a_0 = 0$$

のことである。

線形漸化式はベクトルに対する行列の作用として書くことができ、漸化式の特性多項式とは、その行列の固有多項式になる。

すると、$\chi \in S^\infty$ が漸化式を満たすのは、$\phi(D)\chi = 0$ を満たす、丁度そのときである。

$$DH = HD^2$$

を確かめるのは、簡単である。

これを一つ一つの項で適用していけば

$$\phi(D)H - H\phi(D^2)$$

も分かる。

係数は $\mathbb{F}_2$ なので、$\phi(t^2) = \phi(t)^2$ となる。

よって、もし $\phi(D)\chi = 0$ だったら、

$$\phi(D)H\chi = H\phi(D^2)\chi = H\phi(D)^2\chi = 0$$

[*7] decimate は、間引きを意味する言葉だが、元々は古代ローマで反乱グループに対する処罰として、くじ引きをさせて 10 人に一人ごとに殺したことを意味している。ここでは 2 ごとに間引きしているので、少し合わない。

となるので、$H\chi$ も同様の漸化式を満たす。ただ、初期状態が違うだけである。

もし $\chi \in S^\infty$ の周期が $2^p - 1$ だったら、$H^p\chi = \chi$ となる。実際定義より、$(H^p\chi)_n = \chi_(2^p n)$ となり、周期が $2^p - 1$ ならば $2^p n \equiv n \mod 2^p - 1$ より、これは χ_n と等しくなる。

しかし、逆は一般に真ではない。

しかし、p がメルセンヌ指数であることを使えば上記の定理が言えるのだ。

証明　V をベクトル空間

$$V := \{\chi \in S^\infty | \phi(D)\chi = 0\}$$

とする。

ϕ を特性多項式とする線形漸化式を満たす数列であり、初期値 $x_0, x_1, \ldots, x_{p-1}$ を決めれば、あとは全て決まる。またこの初期値の選び方には何の制約もない。よって v は p 次元である。

そして、$\tau : S^\infty \to \mathbb{F}_2$ を線形写像

$$\tau(\ldots, x_2, x_1, x_0) = x_0$$

とする。

双線形写像 $(g|\chi)$ を

$$\begin{array}{ccc} \mathbb{F}_2[t]/(\phi) \times V & \longrightarrow & V \\ \cup & & \downarrow \\ (g, \chi) & \longmapsto & (g|\chi) := \tau(g(D)\chi) \end{array}$$

として定義する。

まず、これは *well-defined* である。実際、$g + a\phi (g, a \in \mathbb{F}_2[t])$ に対して、 $\phi(D)\chi = 0$ より $(g + a\phi|\chi) = \tau(g(D)\chi + a(D)\phi(D)\chi = \tau(g(D)\chi$ となる。

また、この双線形写像は非退化である。

証明　任意の g に対して、まず $(g|\chi) = \tau(g(D)\chi) = 0$ が成り立つとき、$\chi = 0$ であることを示す。

実際、任意の $n \in \mathbb{N}$ に対して $\tau(D^n\chi) = 0$ を意味しているので、結局 $\chi = 0$ になるしかない。

次に任意の χ に対して、$(g|\chi) = \tau(g(D)\chi) = 0$ が成り立つとき、$g = 0$ であることを示す。

まず、任意の $n \in \mathbb{N}$ に対して、χ_n を、

$$x_k = \begin{cases} 1 & (k = n \text{ の場合}) \\ 0 & (k \neq n) \end{cases}$$

となるような数列とする。すると、任意の $n \in \mathbb{N}$ に対して、$\tau(g(D)\chi_n) = 0$ であることが分かるが、これは g の n 次の係数が 0 であるということなので、g の全ての係数は 0、すなわち $g = 0$ であることがわかる。

この双線形写像が非退化であることは、V の双対空間

$$V^* = \{f : V \to \mathbb{F}_2 | f \text{ は線形 }\}$$

への写像 $\mathbb{F}_2[t]/(\phi) \to V^*; g \mapsto (g|\bullet)$ が同型であることを意味していることに注意しよう。ここでいう $(g|\bullet) : V \to \mathbb{F} = 2$ は $\chi \mapsto (g|\chi)$ という写像である。

さらに、今回 V は有限次元であるので、V の基底を取れば、V と V^* の同型が出来て、$\mathbb{F}_2[t]/(\phi)$ と V との同型も出来る。

この基底と、同型によって構成される $\mathbb{F}_2[t]/(\phi)$ の基底を使って、V の元も $\mathbb{F}_p[t]/(\phi)$ の元も数ベクトルとして表現すると、この双線形写像は数ベクトルの通常の内積と等しくなってしまう（特定の基底から双対空間への同型を作る方法を考えると明らか）。

ここで、写像 $F : \mathbb{F}_2[t]/(\phi) \to \mathbb{F}_2[t]/(\phi)$ を $F(g) = g^2$ として定義する。$g(t^2) = g(t)^2$ よりこれは環の自己準同型写像になる。

この F と間引きオペレーター H が随伴である、つまり、

$$(F(g)|\chi) = (g|H\chi)$$

が成り立つことが分かる。

証明 まず $g = t^n \in \mathbb{F}_2[t]$ の場合に対して証明する。実際、任意の $\chi = (\ldots, x_5, x_4, x_3, x_2, x_1, x_0)$ に対して、$(F(g)|\chi) = \tau(g^2(D)\chi) = D^{2n}\chi = x_{2n} = \tau(D^n H\chi) = (g|H\chi)$ という式変形によって証明できる。

すると、任意の $g \in \mathbb{F}_2[t]$ に対しても、$(g|\chi)$ の g に対する線形性より明らかである。

このとき、先ほどの V の基底（と同型によって構成される $\mathbb{F}_2[t]/(\phi)$ の基底）を使って F と H を数によって行列表現してみると、それらは互いに転置行列になっている（内積の一般論）。

定理の主張に現れる χ にとって、$\phi(D)\chi = 0$ は $\chi \in V$ と同値であり、$H\chi \neq \chi$ は $(H-1)\chi \neq 0$、すなわち $\chi \notin \mathrm{Ker}(H-1)$ と同値、さらに、$H^p\chi = \chi$ は $(H^p-1)\chi = 0$、すなわち $\chi \in \mathrm{Ker}(H^p-1)$ であることと同値である。ここで 1 とは V の恒等写像 $1 = \mathrm{id}_V : V \to V$ を表している。

定理はこのような χ の存在を主張しているので、H を V に制限したものを H と名付け直せば、定理の条件はまとめて、

$$\mathrm{Ker}(H^p - 1) \supsetneq \mathrm{Ker}(H - 1)$$

と表現することができる。

そして転置が和と冪乗との可換性と、転置行列のランクが等しいことにより、上記が成り立つことと、

$$\mathrm{Ker}(F^p - 1) \supsetneq \mathrm{Ker}(F - 1)$$

が成り立つことと同値であることが分かる。

これは $g \in \mathbb{F}_2[t]/(\phi)$ で $g^{2^p} = g$ で $g^2 \neq g$ でないようなものが存在することを意味している。

$l \geq 1$ を $g^{l+1} = g$ となる最小の自然数とする。すると $g^{2^p} = g$ なので、$l | 2^p - 1$ がわかる。ところが今回 $2^p - 1$ は素数なので、l は $2^p - 1$ 自体でしかありえない。

ここで $\mathbb{F}_2[t]/(\phi)$ が $\mathbb{F}_2$ 上の p 次ベクトル空間であり、よって濃度が 2^p であることを思い出そう。すると、g を含んだ環の乗法による軌道の濃度が $2^p - 1$ であると言うことは、乗法群 $(\mathbb{F}_2[t]/(\phi))^*$ の位数が $2^p - 1$ であると言うこと、つまり $(\mathbb{F}_2[t]/(\phi))$ が体であると言うことを意味しているので、ϕ は既約であることがわかる。

また ϕ が既約であるならば、つまり $(\mathbb{F}_2[t]/(\phi))$ が体であり、乗法群 $(\mathbb{F}_2[t]/(\phi))^*$ の位数が 2^p-1 でなので、位数が 2^p-1 である要素 g が存在し、$g^{2^p}=g$ で $g^2\neq g$ が成立している。

これによって示したい同値性が言えた。

実際の inversive-decimation method は次のようなものである（ [4] の Proposition 4.2.）。

系 1 p をメルセンヌ指数とし、$\mathbb{F}_2$ 上の p 次元ベクトル空間 V を、乱数生成器の状態空間とする。$f:V\to V$ を線形な遷移関数とする。$b:V\to\mathbb{F}_2$ を適当な線形写像とする（最初のビットを取り出す関数で良い）。f と b は定数時間 $O(1)$ で実行可能とする。

$\Phi:V\to\mathbb{F}_2^p$ を、

$$\Phi:S\mapsto(bf^{p-1}(S),bf^{p-2}(S),\ldots,bf(S),b(S))$$

定義する、Φ が全単射であり、その逆写像が $O(p)$ の計算量で実行できると仮定すると、f の固有多項式の既約性は、$O(p^2)$ の計算量で計算できる。

実際問題、重要なのは Φ の逆写像の計算量が $O(p)$ と言う部分だけである[*8]。b が定数時間なのは当たり前だし、f は状態空間から数個（メルセンヌ・ツイスターなら 3 個）のワードを取り出して定数個のビット操作をするだけであり、状態空間を大きくしても一緒に大きくなったりはしない。

証明　$S\in V$ に対して $\chi(S)$ を無限列 $(\ldots,bf^2(S),bf(S),b(S))$ とする。$\chi(S)$ の最初の p ビットから初期状態 S が Φ の逆写像で求まり、特に $(0,0,\ldots,0,1,0)$ のような数列を使えば、$H(\chi(S))\neq\chi(S)$ となるように S を選ぶことができる。

またこのことと線形写像 f の固有多項式 ϕ に対しての定理 *1* を組み合わせると、$H^{p+1}(\chi(S))$ の最初の p ビットが $\chi(S)$ の最初の p ビットと等しいなら、ϕ が既約であることが分かる。

ここまでの議論を整理し直すと、f は有限体 $\mathbb{F}_2[t]/(\phi)\cong\mathbb{F}_{2^p}$ の要素であるから、位数 2^p-1 の巡回群 $\mathbb{F}_{2^p}^*$ の要素であり、結果として、これにより f の周期が 2^p-1 であることが証明できたことになる。。

それを以下のような手順で計算する。

まず初期状態 S から $\chi(S)$ の最初の $2p$ ビットを計算する。f も b も定数時間で計算できると仮定したので、これにかかる計算量は $O(p)$ である。

そしてそれを間引きして $H(\chi(S))$ を計算する。これにも $O(p)$ の時間しかかからない。

$H(\chi(S))$ から、それを生成するような初期状態 S_1 を Φ の逆写像で計算する。つまり $H(\chi(S))=\chi(S_1)$ である。仮定よりこの計算には $O(p)$ 時間かかる。

同じ手順をこの S_1 に対しても実行する。$H(\chi(S))=\chi(S_1)$ の最初の $2p$ ビットを S_1 から計算する。そして、それを間引きして $H^2(\chi(S))=H(\chi(S_1))$ を作る。そこから Φ の逆写像で $H^2(\chi(S))=H(\chi(S_1))=\chi(S_2)$ となる S_2 を計算する。

これを p 回続けることで、$H^p(\chi(S))$ が計算できるわけである。

計算量 $O(p)$ を p 回行ったので、全体の計算量は $O(p^2)$ になる。

[*8] これはつまり、過去の出力から状態を簡単に復元できる、と言う意味である。つまり、この擬似乱数は暗号に向かない、ということが、この手法にはどうしても必要である。

inversive-decimation という名前はおそらく、間引き操作をした乱数列から初期状態を復元することができる、と言うことを表現しているのだろう。[*9]。

1.3 フロベニウス自己準同型写像との関係

ここまでの内容は、原論文を辿ってきたが、ここから原論文を掘り下げ、フロベニウス自己準同型の話をする。

1.3.1 Inverseive-Decimation Method とフロベニウス自己準同型写像

さて、上の間引きオペレーターと随伴の $\mathbb{F}_2[t]/(\phi)$ の自己準同型写像 $F : g \mapsto g^2$ であるが、これはガロア理論[*10]などを少しでも齧ったことのある人なら、これが**フロベニウス自己準同型写像**であることは容易に見て取れると思う[*11]。

p を素数、R を標数 p の可換環としたとき、R のフロベニウス自己準同型写像 $F : R \to R$ とは、任意の R の要素 r に対して、

$$F(r) = r^p$$

で定義されるものである。

これが準同型になるのは、$(rs)^p = r^p s^p$ であり、$1^p = 1$ であるという環一般に対する性質に、標数 p の環特有の現象としてさらに $(r+s)^p = r^p + s^p$ が成り立つからである。最後の高校までの数学からは異様な式は、二項展開における係数が、p 次と 0 次の項以外全て p の倍数なので、標数 p の世界では全て 0 になってしまうことから分かる。

R が被約な環、すなわち 0 でないべき零元を持たなければ、これは単射である（$r^p = 0$ なら $r = 0$ なので）。

なので、これが R が体なら単射である。そして有限体ならば、濃度が同じ集合同士の単射なので全単射になり、これは同型になる。つまり有限体 $\mathbb{F}_{p^n} = \mathbb{F}_q$ の場合フロベニウス写像 F はガロア群 $\mathrm{Gal}(\mathbb{F}_q/\mathbb{F}_p)$ の元になるわけだが、実は、フロベニウス写像はこのガロア群の生成元になる。

$$\langle F \rangle = \mathrm{Gal}(\mathbb{F}_q/\mathbb{F}_p) \cong \mathbb{Z}/n\mathbb{Z}$$

ここから F の群論の意味での位数が n、すなわち体の拡大次数に等しいことがわかる。

そう考えると、inversive-decimation method は、状態遷移関数の生成する巡回群が、体 $\mathbb{F}_{2^{19937}}$ の乗法群

$$\mathbb{F}^*_{2^{19937}} \cong \mathbb{Z}/(2^{19937} - 1)\mathbb{Z}$$

と同じ構造を持つかをチェックしようとする代わりにフロベニウス自己準同型により生成される群の構造が、ガロア群

$$\mathrm{Gal}(\mathbb{F}_{2^{19937}}/\mathbb{F}_2) \cong \mathbb{Z}/19937\mathbb{Z}$$

[*9] 間引き操作自体が可逆だと主張しているわけではないところが、少しわかりにくいと思う。

[*10] ガロア理論の入門書はいろいろあり、様々な一長一短があるので、どれを選んでも良く、いろいろ読んで欲しいとは思うが、とりあえず自分がこれで学んだと言う理由で [7] を挙げておく。

[*11] 元々の論文も、名前に F を選んだことなど、これがフロベニウス自己準同型であると言うことは意識しているのであろうが、実際にこれがフロベニウス準同型であると言うことは書かれていない。メルセンヌ・ツイスターを説明するにあたって、必要がないと判断したのかも知れない。

と同じ構造を持つかどうかを調べようとしていた、と見ることができる。

とりあえず群の位数が $2^{19937}-1$ から 19937 へと大きく減っているので、計算量が大きく減る可能性がある、というのでは雑すぎるように聞こえるかも知れない。しかし、まさにこの単純な理由により、ガロア群への操作の計算量が充分に小さいという仮定のもとで、計算量が大きく減ったのである。

1.3.2　フロベニウス写像の世界へ

フロベニウス自己準同型写像は正標数の世界ではあらゆるところに顔を出す。

フロベニウス自己準同型写像は有限体に対しては同型になるが、一般にはそうならず、例えば有限体上の有理関数体 $\mathbb{F}_p(t)$ は t の p 乗根を持たないので、全射ではあり得ない。これが面倒な事態を引き起こすのは、$\mathbb{F}_p(t)[x]$ の要素 (x^p-t) は $\mathbb{F}_p(t)$ 上既約であるが、$\mathbb{F}_p(t)[x]/(x^p-t) \cong \mathbb{F}_p(t^{\frac{1}{p}})$ においては、

$$x^p - t = (x - t^{\frac{1}{p}})^p$$

と重根を持つようなことが起こりうるからである。

このようにある体で既約な多項式を拡大体で分解すると重根を持つような事態は、標数 0 の体では見たことがなかった。それもそのはずである。既約な多項式を拡大体で分解したときに重根が絶対にないような拡大を**分離拡大**と呼ぶが、標数 0 の体の拡大は必ず分離拡大である。このように、拡大が必ず分離拡大になることを、**完全体**と呼ぶ。

そして正標数の体が完全体であることと同値な条件が**フロベニウス自己準同型が全単射**であることなのだ。詳しくは参考文献に挙げた [7] のようなガロア理論の解説書を読んで欲しい。

このように、正標数の世界には、体論の入門講義でまず取り扱う標数 0 の世界とはまた違った豊かな世界が広がっている。そこには必ずフロベニウス自己準同型が顔を出すと言って良い。

例えば、正標数のスキーム論を考えると、**絶対的フロベニウス**, **相対的フロベニウス**, **数論的フロベニウス**, **幾何的フロベニウス**など、一気に種類が増えたいろいろなフロベニウスが現れる。

私はこれらが、擬似乱数の理論に関わりを持つかして、実はほとんど何も知らない。少し聞いたことがあるのが、有限体上の楕円曲線を考えて、その上のフロベニウス写像が、擬似乱数と関係がある、と言う噂くらいだ。

しかし、有限体上に様々な代数幾何的な対象を考えて、それを計算することが擬似乱数や符号理論や暗号理論などに応用ができると言う事態は何度も見てきた。

そして、それには必ずフロベニウス写像が現れるはずだ。

だから、結構応用ができるのでは、となんとなく考えているのである。

もし、何かアイディアがあったら教えて欲しいし、もしご自分で実装までできるなら、やってみて欲しいものである。

1.4　付録:inversive-decimation method の Ruby コード

メルセンヌ・ツイスター自体の様々なプログラミング言語によるコード例は見たことがあるが、この inversive-decimation method の原論文 [4] にある疑似コード以外の、本当

に動くコード例は寡聞にして見たことがないので、ここに本当に動く Ruby のコードを載せておく。

ただし、この文章では、メルセンヌ・ツイスターの細部を大幅に省略しているため、この文章を読んだだけでは、このコードは理解できるようにならない。ただコメントから、なんとなく inversive-decimation method の雰囲気を掴んでもらえるだけである。もし完全な理解を得たければ、[4] を読んで、ここに書かれていないアルゴリズムの細部を調べて欲しい。この文章はこの論文を私なりに解読した結果が多く含まれているので、この文章を読んだ後に [4] を読むと理解がしやすくなるはずだ。

ちなみにこのコードを実際に Ruby で実行すると結果が出るまで数分かかる。C 言語で実装すれば、数秒で終わる、

メルセンヌ・ツイスター自体と inversive-decimation method を Coq で書き、Coq でその性質に数学的な証明を与え、そして C 言語のコードに変換して、高速で動かそうと言う計画が進行中であり、将来的には様々な擬似乱数生成器に対して同じ仕事を行い、擬似乱数生成器に関する Coq のライブラリーへと抽象化する予定である。

Listing 1.1 inversive-decimation method

```
W = 32
N = 624
M = 397
R = 31

P = W * N - R # 19937

UPPER_MASK = ("1" * (W-R) + "0" * R).to_i(2)
LOWER_MASK = ("0" * (W-R) + "1" * R).to_i(2)

BOTTOM_ZERO_MASK = ("1" * (W-1) + "0").to_i(2)
BOTTOM_ONE_MASK = 1

class MT
  def initialize(a, initial)
    @a = a
    @i = 0
    @x = initial
    @timer = 0
  end

  def next
    # computing (x^u_i | x^l_(i+1))
    z = @x[@i] & UPPER_MASK | @x[(@i + 1) % N] & LOWER_MASK
    # multiplying A
    y = @x[(@i + M) % N] ^ (z >> 1) ^ (z & 1 == 0 ? 0 : @a)
    @x[@i] = y

    @i = (@i + 1) % N
    return y
```

```
  end
end

def test_mt(a)
  x = []
  initial = []

  N.times do |i|
    x[i] = i
    initial[i] = x[i]
  end

  P.times do |i|
    mt = MT.new(a, x[0,N])
    (2*P-N).times {|i| x[N+i] = mt.next
} # generate
    1.upto(P) do |j|
      x[j] = x[2*j-1] # decimation
    end
    P.downto(N) do |k| # compute initial state (inversive decimati
      y = x[k] ^ x[k - N + M] ^ (x[k-N+1] & 1 == 0 ? 0 : a)
      y = y << 1
      if x[k-N+1] & 1 == 0
        y = y & BOTTOM_ZERO_MASK
      else
        y = y | BOTTOM_ONE_MASK
      end
      x[k-N+1] = (UPPER_MASK & x[k-N+1] ) | (LOWER_MASK & y)
      x[k-N] = (UPPER_MASK & y) | (LOWER_MASK & x[k-N])
    end
  end

  if (x[0] & UPPER_MASK) != (initial[0] & UPPER_MASK)
    return false
  end

  1.upto(N-1) do |j|
    if initial[j] != x[j]
      return false
    end
  end
  true
end

A = 0x9908B0DF # the last row of the matrix A
p test_mt(A)
```

最後に `A` と書いてある定数は実際に mt19937 で使われているパラメータである。これ

が、メルセンヌ・ツイスターの遷移関数を決める、実際には、これが遷移関数を表現する行列の係数を決めているのだ。

このコードを Ruby で実行すると、`true` と印字されるはずである。これはつまり、実際のメルセンヌ・ツイスターが $2^{19937}-1$ と言う長い周期を実現していることを意味する。

もしこれを例えば、`A = 0x9908B0DD` などのように書き換えてから Ruby で実行したら、`false` が印字されるはずである。つまり、そのパラメータでは、遷移関数の特性多項式が既約ではなく、よって周期も $2^{19937}-1$ より短くなってしまっていることを意味するのだ。

参考文献

[1] 松本眞, 有限体の擬似乱数への応用, http://www.math.sci.hiroshima-u.ac.jp/ m-mat/TEACH/0407-2.pdf, 2004(2019/12/16 accessed).

[2] http://www.math.sci.hiroshima-u.ac.jp/ m-mat/MT/math.html. (2019/12/27 access).

[3] 松本眞, あなたの使っている乱数、大丈夫？. http://www.math.sci.hiroshima-u.ac.jp/ m-mat/TEACH/ichimura-sho-koen.pdf. 2014. (2019/12/27 access).

[4] M. Matsumoto and T. Nishimura, Mersenne twister: A 623-dimensionally equidistributed uniform pseudorandom number generator, ACM Trans. on Modeling and Computer Simulations, 1998.

[5] 桂利行, 代数学 ⟨1⟩ 群と環（大学数学の入門）, 東京大学出版, 2005.

[6] 桂利行, 代数学 ⟨2⟩ 環上の加群（大学数学の入門）, 東京大学出版, 2005.

[7] 桂利行, 代数学 ⟨3⟩ 体とガロア理論（大学数学の入門）, 東京大学出版, 2005.

第 2 章

鳩の巣原理小説

淡中 圏

どのようにして、鳩が存在について思索するようになり、無限という概念を発見し、世界の全てをそれで説明しようとし始めるのか。そのことについて疑問に思っている方は多いと見えるので、今回はその問いに、一つの答えを与えたいと思う。

2.1 鳩と存在

鳩にとって**存在**とは、**鳩の巣原理**によって提示されるものであった。

定理 2 (鳩の巣原理) n 羽の鳩がいて m 個の鳩の巣があり、全ての鳩を鳩の巣に入れるとする。もし $n < m$ ならば、2 羽以上の鳩が入る巣が、少なくとも一つ存在する。

抽象的思考に不慣れな地上的存在のために、まず具体的な数で例を出そう。

10 個の鳩の巣に 11 羽の鳩が入ろうとする。すると、実際ある巣には 2 羽の鳩が入ってしまう。鳩はとても気まずい思いをする。まだこの時点では、鳩は**存在**について、何も知らない。ただ気まずいだけである。

しかし、何度も同じ気まずさを感じ続けた鳩たちが、10 個の鳩の巣に、11 羽を押し込めようとする前に、少し飛び止まって考えたとする。このまま、11 羽の鳩を押し込めたら、鳩が 2 羽以上入って気まずいを思いをしてしまう巣が、少なくとも一つは存在してしまう。

しかしそれがどの巣なのかは、実際に試してみるまで分からない。にも関わらず、それは必ず存在する。

さらに、それは鳩の数 n と鳩の巣の数 m をいろいろ動かしても、$n < m$ が成り立つ限りにおいて、いつでも成り立っている[*1]。

このような性質の不思議さに鳩は目を丸くし、この時に至って初めて、鳩は**存在**と呼ばれる現象の存在に気がつくのだ。

[*1] 地面に這いつくばっている存在には永遠に望めないことであるが、空を飛ぶ存在には、このような抽象への飛翔は日常である。

それ以来、鳩の目は丸いままだ。

また存在に目を凝らすあまり、鳩は目でなく、頭を動かして物を見るようになった。そのような独特の物の見方で、鳩は様々な存在を観察し、考察するのである。

2.2　鳩と実数

鳩の巣で気まずい思いをしたことのない存在には、この原理もいまいちピンと来ないかもしれない。

しかし鳩はまさにこの気まずさから思索を飛翔させ続け、ついには**無限**という概念を抽象の空の高みに発見するに至るのである。

一見有限なものについてしか何かを言っていないように見えるこの主張が、無限についての何事かに到達し得るところが、この原理の深いところである。

例えばこの原理の応用として、次の実数に関する定理が鳩によって証明されている。

定理 3 (ディリクレの定理) 任意の無理数 ω に対して、

$$|x - \omega y| < \frac{1}{y}$$

となる整数 x, y が無限に存在する。

証明　任意の自然数 n を一つ固定する。この時、

$$0 < y \leq n, |x - \omega y| < \frac{1}{n} \tag{2.1}$$

となる整数の組 x, y が必ず一組存在することをサブゴールとしてまず示そう。

二つの実数 $a < b$ に対して、区間 $[a, b)$ を $[a, b) = \{x \in \mathbb{R} | a \leq x < b\}$ として定義する。

このとき、区間 $[0, 1)$ を

$$\left[0, \frac{1}{n}\right), \left[\frac{1}{n}, \frac{2}{n}\right), \ldots, \left[\frac{n-1}{n}, 1\right)$$

と言うように、n 等分する。

y を $0, 1, \ldots, n$ まで動かし、それぞれの y に対して x_y を ωy を超えない最大の整数とする。そうすれば

$$0 < \omega y - x < 1$$

となるが（無理数なので 0 にはならない）、y の候補は $n + 1$ 個あったので、鳩の巣原理により、必ず二つの y に対して $\omega y - x$ が上の n 個の区間のどれか一つに入る。

それを $y' > y''$ と名付けると、

$$|(\omega y' - x_{y'}) - (\omega y'' - x_{y''}| < \frac{1}{n}$$

となる。$y = y' - y'', x = x_{y'} - x_{y''}$ として x, y を取れば、

$$|\omega y - x| < \frac{1}{n}$$

となる（式には書かれてないが、これも 0 にはならない）。

これによって、不等式 *2.1* を満たす x, y の存在が示せた。

実は上で作った x, y は

$$|x - \omega y| < \frac{1}{y}$$

と満たしている。なぜなら、$y', y'' \in \{0, 1, \ldots, n\}$ なので、$y = y' - y''len$ なので、

$$|x - \omega y| < \frac{1}{n} \leq \frac{1}{y}$$

だからである。

そして固定した n に対して、*2.1* を満たす x, y が実は無限個ある。

実際、$\frac{1}{n_1} < |\omega y - x| < \frac{1}{n}$ となる n_1 を新たに取った時（$\omega y - x$ が 0 にならないので、これは必ず取れる）、新しい x_1, y_1 を取って

$$|\omega y_1 - x_1| < \frac{1}{n_1} < |\omega y - x| < \frac{1}{n}$$

と出来て、この系列は無限に続けることができる。

これらもまた、同様の方法で作ったものなので、

$$|x - \omega y| < \frac{1}{y}$$

を満たしており、これはまさに証明したい事実そのものである。

上記の式を y で両辺割ると、任意の無理数 ω に対して、

$$\left|\omega - \frac{x}{y}\right| < \frac{1}{y^2}$$

となる、x, y が無限個あることが示されたことになる。

このような $\frac{x}{y}$ を ω の**ディリクレ近似**と呼ぶ。

上の定理は ω が無理数である場合にしか成り立たない。

命題 1 任意の有理数 ω に対して、

$$\left|\omega - \frac{x}{y}\right| < \frac{1}{y^2}$$

を満たす x, y は有限個しか存在しない。

これを示すために次の補題をまず示す。

補題 1 実数 ω のディリクレ近似で、分母が $y > 0$ であるものは高々 2 個しか存在しない。

証明

$$|\omega y - x| < \frac{1}{y}$$

から、

$$\omega y - \frac{1}{y} < x < \omega y + \frac{1}{y}$$

が分かる。$\frac{2}{y} < 2$ なので、この範囲に入る整数は多くて 2 個である。

これを使って命題を証明しよう。

証明　$\omega = \frac{a}{b}$ $(a, b \in \mathbb{Z}, b > 0$ とし、$\frac{x}{y}$ を ω のディリクレ近似とする。するとこのとき、

$$\frac{1}{y^2} > \left|\frac{a}{b} - \frac{x}{y}\right| = \frac{|ay - bx|}{by} \geq \frac{1}{by}$$

となるので、

$$y < b$$

が成り立たないといけない。つまり可能性のある分母は有限個しかない。そして補題より、分母一つに対して分子は多くて二つなので、全体として有限個しか存在し得ないことがわかる。

これは実数の近似のしやすさを評価する式として見ることができる。有理数を他の有理数で近似することには限界があるが、無理数を有理数で近似することは、分母の 2 乗の誤差で可能だと言うことを意味している。

この定理の証明で鳩の巣原理がどう使われているか確認しよう。

区間 $[0, 1)$ を

$$\left[0, \frac{1}{n}\right), \left[\frac{1}{n}, \frac{2}{n}\right), \ldots, \left[\frac{n-1}{n}, 1\right)$$

と n 等分して、n 個の鳩の巣を作るのだ。

そこに、y を $0, 1, \ldots, n$ まで動かした時の ωy を超えない最大の整数として作った $n+1$ 羽の x_y の鳩を押し込める。

そうすると、2 羽以上の鳩が同じ巣に入って気まずい雰囲気が満ちてしまう鳩の巣が必ず一つ存在する。

その鳩を捕まえると、実はその鳩が実数を良い近似する有理数を作るための材料になるのだ。

これは、古代の鳩国家の**聖体婚姻（ヒエロス・ガモス）**の組みを選ぶ方法に酷似している。

n 個の鳩の巣に $n+1$ 個の鳩を押し込めて、同じ巣に入ってしまった 2 羽が聖体婚姻の儀式を行うのだ。これは、大規模化する前の未開の鳩社会における、通常の婚姻相手の選び方が発展したものだと考えられている。

そしてさらに、古代鳩国家の**生贄**を選ぶのも、これと同様の方法で行われていた。

この二つは表裏一体であると考えた方が良いかもしれない。そして、それは先ほどの定理との関連で考察することで、よりはっきりする。

これらはどちらも、無限への飛翔するための材料を選ぶ、という意味合いを持っているのだ。つまり、鳩が飛翔できる最高の高さのさらにその上の高みを飛翔する**神**への飛翔である。

鳩にとって存在とは、目の前にあるものではなく、いつでも目の前にある物の一つ上に見つかるものである。

目の前にあるわけでもないのに、いつでもそれを発見できる。それこそが存在なのである。

鳩は実際に実数を、必ず次の存在が発見できる一連の流れとして理解している。

具体的な実数に対しての。ディリクレ近似を、先ほどの証明を見ながら実際に計算していくことはできる。実際には、実数に対してディリクレ近似を計算するのなら、実数と 1 に対してのユークリッドの互助法を適用し、連分数表示すると良いだろう。

またある実数が、この分母の指数を変化させた式で近似できるか考えていくと、実はその指数の限界は代数的数としての次数と関連し、もしこの次数がどこまでも大きくすることができれば、それは代数的数ではない、すなわち超越数であることがわかる。

定理 4 (リウヴィル) 実数 ω が任意の正整数 n に対して、ある整数 x, y で

$$0 < \left|\omega - \frac{x}{y}\right| > \frac{1}{y^n}$$

となるものが存在すれば、ω は超越数である。

この定理により、例えば

$$\sum_{k=1}^{\infty} 10^{-k!} = 0.11000100000000000000000100\ldots$$

は超越数であることがわかる。

2.3 鳩と無限

ある時、鳩の若い学僧が師に言った。

「なぜ n 個の鳩の巣に m 羽の鳩を入れたとき、$m > n$ ならば、必ず 2 羽の鳩は一緒になる巣が必ず少なくとも一つは存在するのでしょうか？　そもそも、それに証明は与えられているのでしょうか？　別にそうでなくてもいいのではないでしょうか？」

これはもし、大学の外の公衆の面前で発すれば、たちまち異端として、丸焼きにされてしまう恐ろしい考えだった。前の章に書いたように、鳩の巣原理こそ、彼らの宗教の根幹であり、彼らが社会の基礎と考えるものであったからだ。

幸いにも、大学の中では、比較的そのような考えを述べる自由が確保されていた。

彼は若く、理想に燃え、血気に逸っていた。同じ鳩の巣に入り、気まずい思いをする鳩たちを救いたいと思っていた。そのような鳩を、もう二度と出したくないと、本気で考えたいた。

また、聖なる儀式の名目で、野蛮な犠牲や望まぬ婚姻が行われていることも、苦々しく思っていたのだ。

「それは数の定義による」

師は頭ごなしに否定はせず、ただ重々しく言った。

「数を基数として定義すれば、それはただの定義である」

頭を軽く振りながら、クルッポーと鳴く師の様子は、自らの若い頃を思い出しているようだった。

「数を順序数として定義すれば、それは証明しなくてはならず、そして証明できる。もちろん通常の体系であればだが」

若い鳩も頭を軽く振りながら、神妙にその話に耳を傾けていた。

師の言葉の隙間を狙って若い鳩が何かを言おうとするが、師はその言葉を予想したかのようにさらに言葉を続ける。

「もちろん、論理や公理が十分に弱い体系上で考えれば、それが証明できないことが証明できる、つまり、鳩が鳩の巣より多くても、どこにも 2 匹以上の鳩がいる鳩の巣がないようなモデルを作れるかもしれない」

若い鳩は偶数でも奇数でもない数が存在するような、自然数の部分体系を思い出した。

「しかし、意味もなく弱い体系を作り出しても、仕方がない。それぞれの体系にはそれぞれの意味がある。その意味を見失うと、ピカソとモネの違いもわからなくなってしまう[*2]」

「では、無限についてはどうでしょう？」

若い鳩は食い下がった。

対象が無限の場合は、例えば奇数の集合と自然数全体の集合が 1 対 1 対応を持つような、有限では起こり得ないような不思議なことが起こることが知られていた。

「無限が実際に存在と呼べるものなのか、私には分からん」

古代の哲学者たちは、可能無限と実無限を区別し、可能無限について語ることは許しても、実無限について語ることは許されないと考えた。

「場合によっては、近似として実無限を語ることに意味があるのでは？」

「近似とは？」

若い鳩は実数論からこのアイディアを得ていた。

「例えば、鳩の巣がものすごいスピードで増えていたらどうでしょうか？」

「鳩の巣が増えるのか？　鳩ではなく」

弟子の奇抜な発言に、師は豆鉄砲を食らったような顔をする。

「ええそうです。でも、私は鳩も鳩の巣と共にものすごいスピードで増えていると考えています。そしてもし、数を数えるよりも早く増えるなら、それは事実上の無限と捉えた方が、必然的に不正確な何らかの有限な数を与えるよりも近いと言えるのではないでしょうか」

師は羽を組んで考え込んだ。

「それならば、鳩と鳩の巣のどちらかが多いか考える必要はありません。同じ鳩の巣に入ろうとした 2 羽の鳩がいる時はいつでも、その 2 羽を分ける鳩の巣があるのですから」

師は嘴を挟む。

「自分たちがどれくらいいるのかも分からずに、社会が成り立つのか？」

「大丈夫です。社会の構成員が無限ならば、良い性質の社会的厚生関数が存在します。つまり、良い選挙が実施できます」

「アローの不可能性定理が成り立たないということか」

アローは「非独裁的」で「定義域が非限定的」で「無関係な選択肢が独立」で「パレート独立」な社会的厚生関数が存在しないことを証明した。これは、「ある種の良い選挙のふ可能性を、ある数学的形式化のもとで証明した」と言い換えることができる。

そして、これは有限集合に対しては、単項ではないウルトラフィルターが存在しないという事実から証明できる。そして無限集合に対しては、選択公理（もしくはそのずっと弱いヴァージョン）によって、非単項ウルトラフィルターが存在する。

これは、社会の構成員の数が無限ならば、良い選挙が実施できることを意味しているのだ。

「私は、これこそが、主の言葉『産めよ、増えよ、地に満ちよ』の意味だと考えるのです」

この言葉にさすがの師も首を横に振った。いつも首を縦に振っている鳩が首を横に降る

[*2] ”ピカソとモネの違いもわからない”とは、鳩に伝わる古い諺である。若い鳩には、ピカソとかモネとかが何を意味しているのか分からなかったが、実は年老いた師にも分からなかった。

のはよほどのことである。

「今日はここまでにした方がいい。そして、お前は少し休んだ方がいいだろう」

しかし若い鳩は休まなかった。思考は肉体を離れ、抽象の高みへと飛翔し続ける。その対角線の軌道により、若い鳩は実数全体を自然数個の鳩の巣に入れようとすると、やはり、必ず一つは 2 羽、それどころか実数全体と同じ数の鳩が入っている鳩の巣が存在することを発見した。

無限は一つではなく、いく層にも階層化されていたのだ。

若い鳩の頭の中に自然数個の鳩の巣に 1 羽ずつ入ることができる順序付けられた鳩の群がいくつもできた。どれも、鳩たちが自分より小さい鳩を 1 羽指名していくと、必ず有限回で先頭の鳩にたどりつくような順序だった。

その鳩の集合自体がまるで鳩のように見えた。まるで椋鳥の大群が一つの生き物に見えるように。

いや、若い鳩の脳内で、順序づけられた群は実際に 1 羽の鳩であり、それが列を成していたのだ。その列もまた、鳩たちが自分より小さな鳩を 1 羽指名していくと、必ず有限かいで先頭のの鳩にたどり着くような列だったが、この鳩の群も自然数個の鳩の巣に 1 羽ずつ入れることはできなかった。

その鳩でできた鳩は、自然数個の鳩の巣に 1 入れなかったことを不服としてか、大きく口を開けて、自ら鳩の巣となった。

その鳩の巣に、何羽もの鳩が入っていく。その数は実に実数羽。

若い鳩は脳内に目を凝らした。

果たして鳩は 1 羽ずつ入れるのか。それとも、2 羽以上鳩が入ってしまう鳩の巣が必ず存在するのか。

全ての鳩が鳩の巣に入った瞬間、若い鳩は悟りを開いて、全ての存在は群れであり、それらは最終的には空の群からなっており、自らも群に過ぎないと気付き、雲散霧消してしまったのだ。

参考文献

[1] 塩川宇賢 著. 無理数と超越数. 森北出版株式会社. 1999.

第 3 章

エッセイ: 機械化された数学的知識と共に生きる

淡中 圏

Coq を学ばなければいけないな、と思い続けていて、この半年ほど、ようやく Coq を触り始めている。

かなり手練れと思われるプログラマーたちが、Twitter で「Coq 難しい」と言っていたりするので、どんなものかと思っていたが、やはり難しい。

いつかここでも Coq に関する記事を書こうと思っているが、それはまだ少し早いと思うので、ここでは Coq をはじめとして証明支援系が、人類の「叡知圏（ノウアスフィア）」にとって大きな意味を持っている、みたいなちょっとした大言壮語をしてみたい。

Qita などでは「ポエム」と呼んだりするが、実際あれはとても酷い用語で、本来はエッセイと呼ばれるものだ。そしてエッセイは決して適当に書いて良いものではない。これも、まだまだ荒削りだが、一応は真面目に書いたつもりのエッセイである。

Coq で数学の証明をしようとして、まず面食らうのが、一体何を適用していいのかわからない、と言うことだ。細かいタクティックもそうだし、既に証明された定理や補題を再利用するのでもそうだ。

知り合いに Coq でまず教えられたのが、`About` や `Check` や `Print` や `Locate` などのある項や文法要素について調べるためのコマンド、そして `Search` や `SearchPattern` などの検索系のコマンド（Proof General でのショートカットキー）であることが、それを裏付けていると思う。

既に Coq や mathcomp や ssreflect に巨大な「形式化された知識」の資産があるのだが、それをどう使っていいか分からない。慣れると、検索の仕方も多少は分かってくるのだが、まだ紙と鉛筆で数学をやっている方がずっと効率がいいと感じられる。

それは、なんと言ってもこれまでの紙と鉛筆の数学が 2000 年以上（つまり紙と鉛筆が発明される以前から）脈々と発達してきて、数学的知識自体が紙と鉛筆で行うものとして最適化されているからではないだろうか。

それはある意味では、文字を手に入れてからの我々の知識の最適化全体なのではないか、と思う。

ソクラテスが嘆いたように、文字が使えるようになることによって、人類の記憶能力は大きく低下した。

それはもちろん、記憶能力を高くすることにもコストがかかるからで、文字によってそれを外部化してしまえるならば、そんなコストは必要ないからである。

そのことによって我々の知識の最適化の仕方は大きく変わった。

我々の現在の知識の多くは、一切の文字に手の届かない場所、すなわち社会の知識的インフラから遠く離れた場所ではうまく使えないであろう。

現在のインターネットに溜め込まれ続ける知識も、未だにこの文字の時代の知識から変わっていない。

セマンティック Web だのなんだの言っているのは、このような旧時代の巨大なレガシー知識体系に対するか細い抵抗であった。

なぜ抵抗する必要があるかと言えば、文字による知識の遺産が巨大になり過ぎて、このままでは保守コストが耐えられないほど上がるという予感、そして兆候が明らかに見えているからである。

学問の世界はこの 100 年で大きく発達し、細分化した。学問の価値を民主主義的に決定することは不可能だ。だが、あまりに細分化した結果、パトナムの言う「言語的分業」が先鋭化し、お互いに言葉が通じなくなってしまうことが起きている。

小学校の先生は、今や数学的に間違った算数を教え、歴史学的に間違った歴史を教え、スポーツ科学的に間違った体育を教えていいても、専門家の言葉が彼らに届くとは限らない。そもそも専門家の言うことなど自分たちには関係ないと言いかねない状態になってしまっている。

政治家たちは、経済学者の言うことを聞かないし、患者たちは医療の専門家の声よりも、詐欺師の言うことを信じかねない。

いろいろな原因が考えられるが、一つには我々は発達した学問の全体を一人で理解することはとっくの昔に出来なくなっているし、それらの知識を普段から利用しているが、根拠が分からないので偽物と本物の区別がつかない、と言うことが一つだろう。

それは学者にも言えて、学者は自分の専門以外のことには素人で、そして素人と同様に、細分化した他の学問が理解できない理由で否定したりとんでもないことを言ったりしがちなのだ。

細分化の結果、民主主義に頼らずに学問の質を保つ有意義なシステムである**ピアレビュー**も、ちゃんと機能するか怪しくなってきた。自分の扱っているテーマと少しずれてしまうだけで、その論文を正しく評価できないかもしれないのだ。

このような事態に対して「タコツボ化が悪い」と言う言い方がなされることもある。

しかし、タコツボ化は必然だ。

学問が進歩すれば、学問が扱うテーマは増えていく。もしタコツボ化をしなければ、テーマの数が n とするならば、そのテーマと他のテーマの関連は $\frac{n(n-1)}{2}$ になる。つまり、学問の進歩とともにテーマ同士の関連は $O(n^2)$ 乗のオーダーで増えていくのだ。一つのテーマを一人ずつ取り持つとすれば、研究資源は $O(n)$ のオーダーで増えていく。これはあまりに雑なモデルなので、話半分に聞くのが正しいが、それでもここからもしタコツボ化をしなければ、他の研究にキャッチアップするためのコストが急速に増えて、研究の進展が遅れるであろうことが予想される。

これはソフトウェア開発の古典である『人月の神話』に書かれたブルックスの法則「遅れているプロジェクトに人を加えるともっと遅れる。なぜならプロジェクトの進捗速度は人の数に比例するが、チームの複雑度は人の 2 乗に比例するからである」のちょっとした

書き換えである。

この（哲学というにはあまりにも雑な）例え話のちょっと面白い帰結は、「タコツボ化が可能であること」が「科学的研究活動が実践的に可能であること」の必要条件であることがわかることだ。

パトナムや少し昔の科学哲学者は「世界が根本的に理解可能かどうか」みたいな問いを立てていたりしたように僕には見えるが、この問いは「世界が実践的に理解可能かどうか」と言い換えられる。

もし世界がタコツボ化を拒否するようなものだったら、つまりあらゆる学問のジャンルがそれ以外のジャンルと必ず大きく関連するようになっていたら、世界は複雑すぎて、知識の管理コストばかり嵩んで知識を増やす方にほとんどコストをかけられないだろう。

先ほどのブルックスの法則の話に戻ると、その場合の解決は正しいチーム分け、そのための問題の正しい小分け、そしてその結果としてのソフトウェアの正しいモジュール分けである。

つまり、ある程度関連しあった問題を集めたモジュールを分け、モジュールの中ではお互いに気をつけていなくてはいけないが（凝集度が高く）、モジュールの外に対してはあまり気にしなくて良いよう（疎結合）にする。

これによって、人数を増やしてもしばらくはある人が気にしなくてはいけない他のメンバーの数は定数に抑えられる。

ソフトウェアのモジュールも、他のモジュールとの関係をあまり気にしなくて良いので単純になる。

これはつまりタコツボ化こそが成功の鍵だ、とも言い換えられるということだ。

そしてソフトウェア開発をしたことのある誰もが経験のあることであろうが、モジュール分けに失敗すると、モジュール分けをしたのに複雑度が全く抑えられない。

全てのモジュールが他のモジュールと関係しあって、スパゲティ化してしまうのだ。

そうなると事態が収集がつかなくなる前にリファクタリングをして、正しいモジュール分けを見つけなくてはいけない。

そしてソフトウェアエンジニアはみんな知っているように、経験的に正しいモジュール分けはそれなりにある。

つまり、魔法のように複雑度を下げるモジュール分けの方法なんてこの世にはないけれど、対処できないほど複雑なプロジェクトというのは基本的に存在しないのだ。

これは今のところ、この世界はタコツボ化が可能なように見えていて、だからこそ今のところ科学が可能に見えていることに対応している。

もしかしたら、このモジュール分けの失敗によるリファクタリングは科学において一度起こっているのかも知れない。

ガストン・バシュラールの『科学的精神の形成』は 18 世紀の通俗科学読物を読んで、その今から見るとちょっと笑ってしまう内容を真面目に紹介してくれる大層愉快な本だが、この本を読んでわかることは、当時の人々は、全ての学問（物理も科学も）生物学や心理学の言葉で語ろうとしていたようなのだ。

それこそ有機物と無機物の区別という今となってはかなり恣意的に見える区別や、原形質流動という概念の「原形質」の部分にその残滓が見えるだろう。

当時の人々は、あるものが高い反応性を見せている時、それは「興奮している」という用語で語りがちだった。

バシュラールがいうように「浮力を水が押しているとは普通考えない。物体が上に浮き上がろうとしていると考える」のだ。

科学の教科書ではガルヴァーニがカエルの解剖中に、二種類の金属（ナイフと皿）でカエルの体を挟んだことで流れた電流を「生体電気」と名付け、生体に備わったものだと考えたことを少し紹介したあと、すぐにボルタのボルタ電池によってそれが生体と関係ないと分かったと繋がってしまう。

しかし、実際には生体電気の考えはその後、ドイツロマン派の自然哲学などに脈々と受け継がれ、生体磁気と名を変えメスメルの磁気療法、そして催眠術へと流れ込む。

そこまで理解しないと、どうしてフリッツ・ラングの『怪人マブゼ博士』や『キン肉マン』などで、目を見るだけで催眠にかかって操られてしまうのかが分からなくなってしまう。あの描写には、予想以上に歴史があり、そして我々の「生命力」に関する妙な直感に支えられているのだ。

しかし 19 世紀以降の学問の進歩によって、生物学的な我々の直感は学問の世界から大きく取り除かれた。それは、デネットなどの自然哲学者が「機能」や「目的」などの用語を学問の世界に再輸入する必要性を叫ぶくらい取り除かれてしまっているほどである。

今では、「魚類」などの我々の生物学的直感の根幹に関わる分類方法が、科学の名の下に攻撃されている。これは例えば「正方形は長方形ではない」と小学校で教師が教えることに数学者たちが反対することと軸を同じくしている。これは決して小さい問題ではない。バシュラールは「ケプラー以前は、楕円は円が崩れたものだと考えていた。数学の素養のあるケプラーは円を楕円の一種と考えられた。これがケプラー法則の発見に本質的であったに違いない」という話を書いているが、これは流石の直感である。

「魚類」などの直感的分類を放棄することには大きな抵抗がある。また哲学者のトマス・ネーゲルなどは、物理主義に抵抗し、心などが他へ還元できない世界の基礎的な概念であるという主張をすることがある。

これなども、19 世紀以降に科学の基本的語彙が生物学的な語彙から物理学的な語彙へと大きくシフトしたことへの抵抗と見ることができるだろう。

私の考えるところ、物理主義とは畢竟、学問を進めるために長年の経験から導き出した経験則、ベストプラクティクスなのであって、実際それで研究がうまく進むかどうかが問題であって、もしそれに反対するなら他の方法で研究をもっとうまく進めるしかない種類のものである。

しかし、もし物理主義がひっくり返って、また根本的なリファクタリングが必要になったらどうなるだろうか。

私は「科学の実践的可能性に疑問符がつくのでは？」くらい考えている。

もし私がプロジェクトを進めていて、最初のモジュール分けがうまくいかず、根本的なリファクタリングが必要になっても「それはそうだろう」としか思わない。しかし、それが複数起こるとなると、私は「この仕事は割りに合わない」と考える気がするのだ。

ネーゲルはそんなことは全く考えていないとは思うが、ある種の哲学者の願いが叶うと、科学の実践的可能性への我々の信念が揺らぐ可能性がある、と思うと個人的には少し楽しかったりする。

閑話休題。

ここまで科学の発展、つまり知識の獲得にかかるコストについて考えてきた。そしてタコツボ化によってそれを下げることができると主張したきた。それと最初に少し挙げたタ

コツボ化の弊害とはどういう関係にあるのだろうか。

一つにはやはり単純にタコツボ化は不可能であるようだ、ということだろう。タコツボ化によって、実際には存在するモジュールとモジュールの間の関係が無視されてしまう。これは例えば、デザイン・パターンという概念を作ったクリストファー・アレグザンダーの『都市はツリーではない』などで大きく語られている観点である。問題を単純にツリー上にモジュール分けして整理できると考えるのは思い込みに過ぎない。世界はもっと複雑であり、それに対処するためにはもっと柔軟な考え方が必要だ。

というわけで彼の考えは、ソフトウェア工学ではアジャイル手法の哲学へと繋がっていく。

しかしアジャイル手法でも固定的に考えることをやめるというだけで、モジュール分け自体は基礎中の基礎である。

そしてもう一つの見方では、タコツボ化の弊害とは結局、タコツボ化の弊害というより単にタコツボ化では抑えきれない知識の保守コストが膨れ上がって来た、と考えられるかも知れない。

今や数学では学ぶべき基礎知識が膨れ上がり、ある分野で最新の結果を理解するために必要な学習期間はどんどん伸びている。それは他の学問でも起こっていることだろうと思う。

このままでは、知識の保守コストが知識による利益と釣り合ってしまう。そうなれば、全ての資金は知識を保守するために使われ、科学の進歩は止まってしまう（私は冗談で逆 ω 点と呼んでいる）。

これは単に学問の中の問題ではなく社会全体の問題である。最初の方に語った社会の中での専門家と素人（つまり知識のユーザー）のコミュニケーション不全も、知識の保守ができていないことの端的な表出である。

かつての知識の理解は、抽象的な命題の集合のようなものだったかも知れない。ポパーのいう「客観的知識」は図書館などに収蔵された様々な論文に書かれたインクの染みによって表現された「命題」のようなもののようだが、自然主義以降、このような考え方は支持しにくくなっているように感じる。

知識の宿る存在を個人とするにせよ、社会とするにせよ、コンピュータなどを含めるにせよ、どう考えても知識を溜め込んでるだけでだめで、それは利用可能でなければいけない。そしてそれにはコストがかかる。

例えば図書館ならば、利用可能にするために分類があり、以前ならカードがあり、今ならコンピュータ端末から検索できる。それによって大量の蔵書の利用可能性を上げている。

しかし、それが地球ほどの図書館ならば、今の方法ではとても利用可能ではない、という可能性もある。

知識に対して、それほど資源や時間に制限がないときは十分に抽象的な概念で考えればすむが、場合によっては物理的ハードウェアまで降りて考えなくてはいけないのだ。そこまでいかなくても、その中間の、例えばデータベースのインデックス付けなどのレベルを考える必要は実際あるであろう。

このように考えれば、知識というものは命題のような抽象的で静的なものではなく、どちらかというとソフトウェアに近い動的なものである、という描画が合っている気がしてくる。

このような考えを大きく打ち出した哲学書にデービス・ベアードの『物のかたちをした知識』というものがある。

この本では、例えば天球儀というものも、宇宙のモデルを目に見える形にしたという意味で知識と読んで良いし、また電磁石のような「動作するもの」もそれを見ただけで、原理は分からなくとも、使い方が分かったりするものなので、ある意味「知識」と呼んでよいのではないだろうか、と提案している。

この路線は「知識に対して抽象的に考え過ぎず、しっかりコストを考えよう」というこの文章のテーマとも相性が良い。

そして、実は我々の知識の大半は「動作する知識」だと考えられる。

我々は飛行機に乗り、パソコンを使っているが、それらの原理を全て理解している人というのはいない。

さらに言えば、「言語としての知識」はそれら大量の「動作する知識」に依存している。

我々が文字を読んで理解し、そこに書いてある知識を利用できるのも、言語を読むという「動作する知識」を基礎としているが、日本語が読めるのに日本語の文法のテストが解けない人が多くいるように、それは全く言語化されていない知識である。

そう考えるとかつて「普遍言語」というものが目指していたものは、そのような依存する「動作する知識」を一切必要としない「言語」を得ようとした、ともみなせる。

一切の文脈を必要としない言語。

しかしそれは不可能である。ソフトウェア業界では、相変わらずこれと同様の銀の弾丸を歌う詐欺師が跋扈しているようではあるが。

もちろん、それら「動作する知識」をできる限り言語化するというアプローチ自体は可能である。

しかしそのコストは急激に高まる。

例えば始終変化する者を言語化することは無駄が多い。せっかく言語化しても、次の日にはもう変わってしまっていては、また書き直さなくてはいけない。もちろんその中から変わらないものを発見して言語化することを目指すことは可能だが、必ずしも成功しないし、時間もかかる。

それなら現状うまくいっている「動作する知識」だけに任しておいたほうが安上がりということが普通にありうる。

何の話をしているかというと、ソースコードのコメントの話である。

現代において「動作する知識」としてソフトウェアを考えるのはとても面白いと思う。

なぜならソフトウェアは「動作する知識」と「言語としての知識」の中間物と見做せるからだ。

ソースコードを公開していないソフトウェアは、ユーザーにとっては完全な「動作する知識」であり、世界を支えるブラックボックスの「文脈」である。

ソースコードがあれば普通は読まないが、読もうとすることはできる。しかしそれが読めるとは限らない。汚いコードならそれは相変わらず単なる「動作する知識」であり、綺麗ならそれは「言語としての知識」に近づいていく。

コメントを書くことにより、「言語としての知識」に近づいていくが、ソースコードの変化に追随させることはコストがかかるし、もしソースコードとコメントの矛盾が起これば、むしろ可読性が減ってしまい、バグの温床になる。

だからこそ、コメントをどれくらい書くのかは議論が生じるのだ。

コメントを全く書かないのも、書きすぎるのも問題なのだ。

ちなみに「動作する知識」の別名はマイケル・ポランニーの「暗黙知」であり、「言語としての知識」は「形式知」である。

ポランニーは、「盲人にとって杖は意識しないから、つまり暗黙知だから役に立つ。もし意識してしまえばうまく使えなくなる」という例え話で、哲学のように暗黙知を形式知に変えようとする行為が、害を成す可能性について指摘したが、これは上のコメントの問題のように、形式知の保守・運用コストの問題と考えると分かりやすいだろう。

一見コストが低いからといって、安易に暗黙知の側に流れていいわけではないのだ。

「動作する知識」は動作している間はいいかも知れないが、動作しなくなった時に、何が間違っているか分からないのだ。スパゲティコードを少しだけ書き換えるなどということは不可能なのだ。

リチャード・ストールマンやローレンス・レッシグらが、たとえ余分のコストを払おうとも、フリーソフトに拘る理由はそこにある。

今の社会は多くのソフトウェアが社会を支えている。もしそれが「暗黙知」の形で提供されているとしたら、それはつまり、我々の社会を支える知識が言語化されていない、ということだ。何が問題があった時に、一体どこが問題なのかわかるのだろうか。

だからこそ、社会の根幹を支えるソフトウェアはフリーでなければいけない、と彼らは考えるのだ。それに余分のコストが必要であってもである。

これについては民主主義の成員である我々はそれなりに真面目に考える必要があるだろう。

そして、このままではいずれ、我々の社会の根幹を支えるソフトウェアがたとえオープンソースで公開されていたとしても、それを読み解くのは非常に難しく、実質「暗黙知」となってしまうと現状私は考えている

これもまた、我々の知識の保守コストになっていくのだ。

どうすればいいか。ここで、ようやく話は Coq へと帰っていく。つまり、古き良き文字による知識へのか細い抵抗の一つ。

知識のなかの根幹的な部分を形式化する必要がある。

記憶の多くの部分を文字によって外部に依存してしまったように、理解の多くの部分も形式化によって外部に依存してしまうしかないのではないか。

それによって我々は、文字を覚えたことによって記憶力を大きく失ったように、理解力を大きく失うのかも知れない。

それでも文字を覚えたことによってより大きな知識の活用ができるようになったのと同じように、形式化を覚えたことによってより多いな知識の活用ができるようになるかも知れない。

形式化して証明することは、究極のモジュール分け、究極のブラックボックス化である。それを使うだけなら、本当に中身を見る必要は一切なくなる。

そしてその証明は機械がチェックしてくれる。我々はインターフェースの仕様だけ本当に覚えれば良い。モジュール分けによる複雑度の抑制がフルに使える。

また状況が変化した時は、証明が壊れることによっていち早く察知できる。そして証明を直すことによって、再び中身を気にすることなく、モジュールを使用することができる。

これは「言語としての知識」と「動作する知識」との、どちらについても言える話である。

我々はその機械言語で書かれた証明を普通の言語のようには理解できないかも知れない。

だが、それは「間違っていたらどこが間違ったかわかる」「どこを直せばいいかわかる」という意味で、「言語としての知識」の持つ良い性質をちゃんと持っていて、しかも「言語としての知識」と違って理解しなくてもよく、保守コストを大きく下げられる可能性がある。

そして、多くの場合、新規作成のコストよりも保守コストの方が問題になりやすいのだ（それこそがこの文章のテーマだ）。

そして形式化された知識は、知識の利用可能性を大きく進歩させる可能性もある（やっと冒頭の話に戻ってきた）。

それは今は、人間の記憶と連想によって行われているが、その記憶の部分をまるごと機械が置き換えてくれれば、もっと効率が良くなるかも知れない。

今の Coq の使える補題やタクティックの検索性能やユーザー・インターフェースを改善するのも必要だろう。

それ以上に、例えばある問題に適用可能な位相の提案とか、分野ごとの特殊化された推論をタクティック化することの提案とか、様々な「機械と人間のコラボレーションする数学の夢」が広がる。

いや、実際には、「なんで通らねんだよ」と叫びながら Coq と悪戦苦闘しているのだが、そんなことを考えながら、やっていっているのである。

参考文献

[1] ベアード, デービス./松浦俊輔 訳. 物の形をした知識 実験ときの哲学. 青土社. 2005.

[2] ポランニー, マイケル./高橋勇夫 訳. 暗黙知の次元. 筑摩書房. 2003.

[3] アレグザンダー./稲葉武司 押野身邦英 訳. 形の合成に関するノート/都市はツリーではない. 鹿島出版会. 2013.,

おまけ:ぞうの卵はおいしいぞう

淡中 圏

あなたの前には、どこまでも空白のファイルが広がっている。実はファイルは広がっておらず、むしろ小さいのだが、画面に空白が表示されているので、広漠とどこまでも広がっている感じがする。とは言え、その真っ白なページをたくさん印刷して製本するわけにもいかない。筒井康隆の『虚人たち』じゃあるまいし。しかし、このままではどうにも本が薄くなってしまう。

どうしよう。

こんな時、アカデミズムと付き合った経験のある人間の多くが思い出す文章がある。

それは

ぞうの卵はおいしいぞう。

である。

これは**科研費 LaTeX** のサンプル文章の中で、ぞうの卵の研究計画について熱い思いを込めて真面目に語っていたと思ったら突然現れる文章である[*1]。実際にはこれが延々と続くのだ。これはサンプル文章を間違えて提出してしまった時に、一眼で間違いに気付けるという深慮があると思われる。実際にそのようなミスが起こったことが Twitter で報告されている [4]。

学振や科研費の申請の書類を書いたことのある人間の多くが、このサンプル文章を目にしている。そして何を書けばいいかを悩んだことのある人間の多くが、このサンプル文章を（事故でなく）故意にそのまま提出したくなる誘惑を感じるのだ（個人的見解）。

そしてそのような人間が同人誌を作ろうとし、どう考えても内容が足りない原稿データを見たとき、脳裏に浮かぶのはやはりあの文字列なのである。

ぞうの卵はおいしいぞうで原稿を埋めたい！

でも、それを実際タイピングするのは面倒臭い。コピー&ペーストも一回や二回ならいいけど、100 回を超えると小指が痙ってしまう。

こんなとき、プログラミング的思考を義務教育で習ったはずのこれからの小学生ならどうするであろうか。

[*1] 科研費 LaTeX とは日本学術振興会と文部科学省の科学研究費補助金（科研費）の応募の書類を、LaTeX で書くための道具である。これは大阪大学の山中卓氏が 2006 年秋から日本学術振興会の学術システム研究センターの数物系科学専門調査班の活動の一つとして作られ始め、2017 年秋からは、日本学術振興会からの依頼に基づいて作成されている。詳しくは本家サイト [3] を参照。

もちろんプログラミングするのである。

というわけでプログラミングしてみた。

この原稿は LaTeX で書いているので、TeX でプログラミングするのは自然な流れだと思う。

これを読んでいる皆様は十中八九 TeX でのプログラミングに習熟しているであろうから、あまり細かいことは説明しないが、プログラミングができる方には、見慣れたものばかりであろう。

もちろん TeX や LaTeX のマクロについて本格的に入門したい方は、ちゃんとした本を読んでください[*2]。

```
\newcount\counterI
\def\zounotamagohaoishiizou#1{
  \counterI=0
  \loop\ifnum\counterI<#1ぞうの卵はおいしいぞう。

  \advance\counterI by1
  \repeat
}
```

説明すると\newcount は、TeX のカウンターを定義するコマンドである。カウンターの整数値をとることができる。今回は\newcount\counterI で coutnerI というカウンターを定義している。これは何回「ぞうの卵はおいしいぞう」したかを表している。

続いて\def は TeX でマクロを定義するためのコマンドで、今回は

```
\def\zounotamagohaoishiizou#1 { ... }
```

によって、\zounotamagohaoishiizou というマクロを#1 という仮引数が一つあるものとして定義している。中括弧の中身が処理の本体である。

代入は\counterI=#整数か\counterI=#カウンター名の形式で行われる。今回はまず\counterI=0 によって、カウンターを初期化しています。

次は\loop である。これは、\loop <処理 1> \if <条件> <処理 2> \repeat という形で使う、C 言語で言えば、while ループと do while ループを合わせたようなものだ。

しかし、実は\loop は、TeX の元々のコマンドではなく、実はクヌースが必要最低限のマクロを定義した **plain TeX** の中にあるマクロである。

TeX のマクロとは、ただ単にマクロとして定義されたトークンを文字列に展開し、中に展開された文字列の中にさらに処理すべきトークンがあれば、さらに処理していくというだけで、制御と呼べるものは実質 if しかない。繰り返しなどは実は全て、マクロの再起的な定義で行っている。なんだか Lisp の練習問題でループ構文をマクロで再帰関数の定義に書き換えるみたいな話である。実際、ここを追求し始めると、なんだか関数型言語でコーディングしているみたいな頭の使い方を要求されるが、今回は、もちろんそんなことはしない。何も考えずにただのループだと思って使う。

[*2] 日本語の本としてよくおすすめされるのが「黄色の本」こと [6]。しかし上記が手に入りにくいため、手に入るものとして代替に挙げられるのが同じ著者の「もっと黄色の本」こと [5]。英語でも構わないなら、CTAN に入っており無料で手に入る [8]。そして、やはり王道を行く [7] も一応挙げておく。

ここでは、\ifnum\counterI<#1 で、引数に与えられた数とカウンターを比べている。もしこれが真ならば「ぞうの卵はおいしいぞう」と表示し、\advance\counterI by1 によって、カウンターを一つ増やす。そして\repeat によって、もう一度ループをするわけだ。

これによって、与えられた数だけ「ぞうの卵はおいしいぞう」する TeX のマクロができた。

あとは引数に好きなだけの数を渡してマクロを実行すれば良い

\zounotamagohaoishiizou{400}

ぞうの卵はおいしいぞう。ぞう

の卵はおいしいぞう。ぞう

の卵はおいしいぞう。ぞう

の卵はおいしいぞう。ぞうの卵はおいしいぞう。ぞうの卵はおいしいぞう。ぞうの卵はおいしいぞう。ぞうの卵はおいしいぞう。ぞうの卵はおいしいぞう。ぞうの卵はおいしいぞう。

もう少し抽象化するなら次のようになるであろう。

```
\def\babbling#1#2{
  \counterI=0
  \loop\ifnum\counterI<#2
    #1
  \advance\counterI by1
  \repeat
}

\babbling{\textbf{BABEL} }{1300}
```

BABEL BABEL

BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL

BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL

BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL

BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL BABEL BABEL BABEL BABEL
BABEL BABEL BABEL BABEL

いかがでしたか？

参考文献

[1] 寺村輝夫 著. ぼくは王様 - ぞうのたまごのたまごやき. 理論社.1998

[2] エッツ, マリー・ホール/まさきるりこ 訳. もりのなか. 福音館書店. 1863.

[3] 科 研 費 LaTeX./http://osksn2.hep.sci.osaka-u.ac.jp/~taku/kakenhiLaTeX/(2019/12/22 access)

[4] Togetter./https://togetter.com/li/580804(2019/12/22 access)

[5] 吉永徹美 著. 独習 LaTeX2ε. 翔泳社. 2008.

[6] 吉永徹美 著. LaTeX2ε マクロ&クラス プログラミング基礎解説. ページエンタープライゼズ. 2002.

[7] クヌース, ドナルド E./鷺谷好輝 訳.TEX(テック) ブック—コンピュータによる組版システム (アスキー・電子出版シリーズ). 1989.

[8] Eijkhout, Victor. TeX by Topic. http://www.ring.gr.jp/pub/text/CTAN/info/texbytopic/TeXbyTopic.pdf(2019/12/23 access).

淡中 圏　本名：田中健策

私は、この同人誌制作中に頭の中がわやくちゃになって、さあ今日は同人誌を書くぞ！　と思った途端に、人と会う予定を二つほどぶっちぎりました。一つは何と前日に「明日の朝ですね」と確認した用事でした。しかも夜に。そう確認して家に帰ってパソコンの前に座り、同人誌を書き始め、そして深夜になり、「続きは明日にしよう」と思って寝て、昼ごろ起きて同人誌の続きを書き始めました。

みなさん、同人誌は体にも社会にも毒になり得ます。用法容量を守って、正しく制作しましょう。

よく分からない自作 web ページ　`https://tannakaken.xyz/`

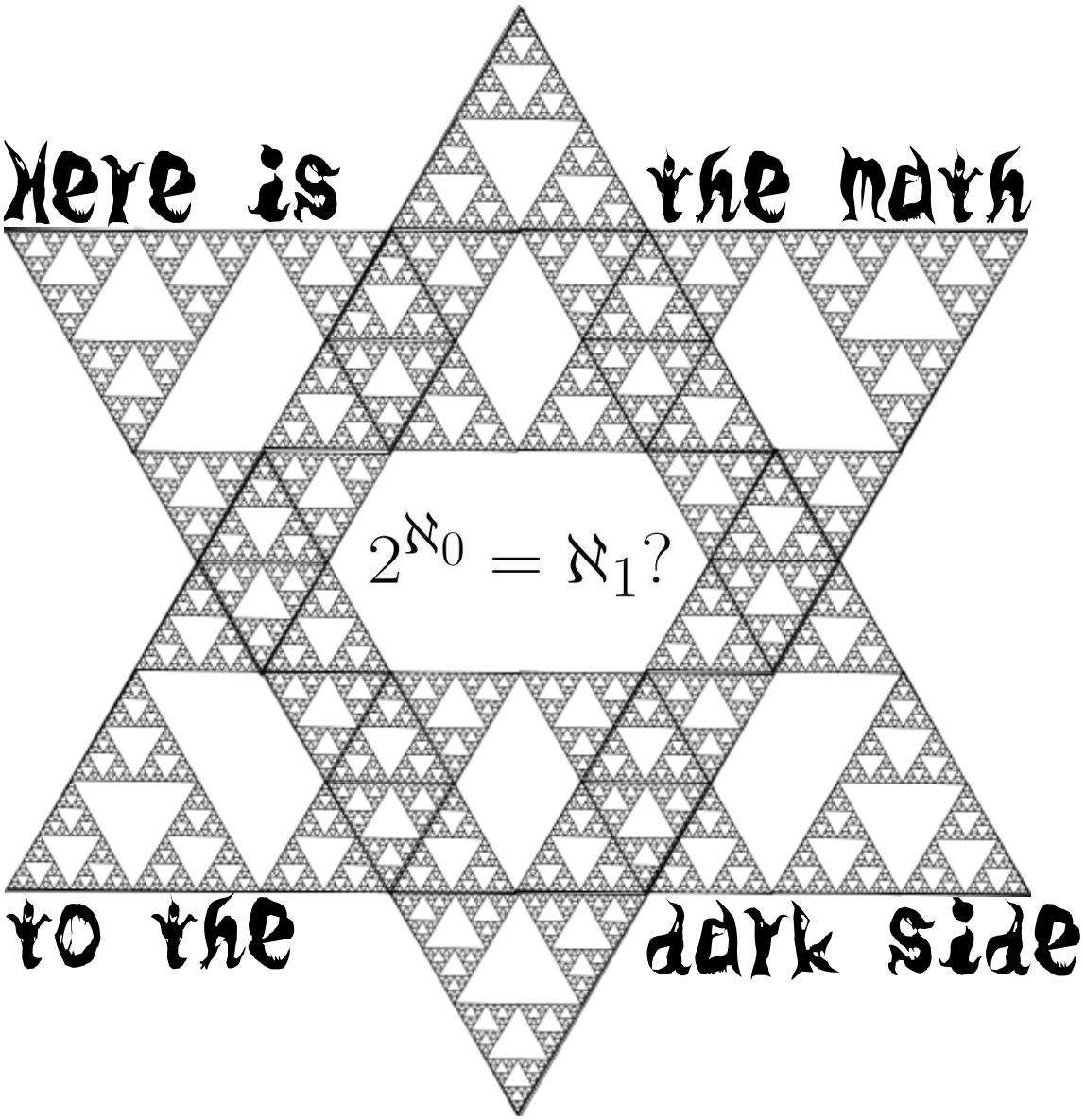
編集後記:

何となく個人あとがきと全体の編集後記を分けていましたが、いまやだんだん分ける意味も失われた今日この頃、みなさんいかがお過ごしでしょうか？

もう書くことなんか何にも残ってないんです。でも、次こそは次こそは、と思って温めてるネタはある。o-minimal 理論についての入門記事を書きたいとか、Morley の非加算範疇性定理についての解説記事を書きたいとか、プログラミングでまた馬鹿なことがしたいとか。そのために、途切れさせたくないなあと言う思いがこの段々内容が薄くなっている本を支えています。さあ、The Dark Side of Forcing の明日はどっちだ？まあ、Dark Side に決まってはいますが。

【淡中　圏】

発行者 ： The dark side of Forcing
連絡先 ： http://forcing.nagoya
発行日 ： 2019 年 12 月 30 日


Here is the math
$2^{\aleph_0} = \aleph_1?$
to the dark side