巨大数論

第2版

フィッシュ 著

はじめに

私たちが日常生活で使う大きな数は、たとえば世界の人口が 75 億人であるとか、日本の GDP が 500 兆円以上であるといったように、100 兆=10 の 14 乗程度までです。その上の単位である京（けい）を知っていても、さらにその上の垓（がい）といった単位を使うことはめったにありません。日本語の数の単位は、一、十、百、千、万、億、兆、京、垓、秭（じょ）、穣（じょう）、溝（こう）、澗（かん）…と 10 の 68 乗の無量大数まで続きます。

科学の世界では、たとえば 1 モルに含まれる要素粒子の数を表すアボガドロ数という数は、約 6023 垓になります。物理的に意味のある非常に大きな定数としては、エディントン数という数があります。これは全宇宙にこの数の陽子があるとしたもので、2 の 256 乗の 136 倍で、10 の 79 乗のオーダーになり、無量大数までの数の単位の体系では命名できなくなります。「天文学的数」という表現がありますが、天文学の世界でも、エディントン数よりも大きい数を扱うことはまずありません。

ところが、それよりもさらに大きな数が仏教の経典には書かれているとのことです。経典に書かれている不可説不可説転という数の大きさを計算すると、10 の 37 澗（かん）乗程度というとてつもない大きな数になります。「不可説不可説転メートル」がどれくらい長い距離なのか、あるいは「不可説不可説転秒」がどれくらい長い時間なのか、想像をすることすら不可能です。さらに、古代ギリシャでは、アルキメデスが「宇宙を砂粒で埋め尽くすためには何粒の砂粒が必要か」という計算をするために、10 の 80000000000000000(8 京) 乗までの数の単位を考えました。ただし、計算に実際に使われたのは 10 の 63 乗までででしたので、とにかく大きな数まで名付けておきたかったのでしょう。そのような太古の昔からそれだけ大きな

数が考えられていた、ということからも分かるように、人々は古くから大きな数に魅了されてきました。

20 世紀に入ると、数学者が著書に書いたグーゴルやグーゴルプレックスという大きな数が紹介され、やがてプロの数学者が書いた書物を読んだ数学愛好家の中に、巨大数の魅力にとりつかれて独自に拡張した巨大数を考える人たちが出てきました。巨大数には、プロの数学者が数学の専門的な問題を解くときに副産物として出てきたいわゆる「正統派巨大数」（グラハム数やスキューズ数）と、単純に大きな数を目指した「お遊び系巨大数」があります。お遊び系の巨大数は、プロの数学者が趣味で開発したもの（コンウェイのチェーン表記やスタインハウス・モーザー表記など）もありますが、多くはアマチュアの数学愛好家が定義したものです。

このような巨大数を考えるアマチュアの数学愛好家は、当初はそれぞれが孤立して独自の考え方で巨大数を定義していました。やがて、インターネットでロバート・ムナフォの Large Numbers というサイトを発端に、巨大数に関する情報が整理・集約されてきました。巨大数の議論はインターネット上で白熱し、特に 21 世紀に入ってから、様々なアマチュアの数学愛好家たちが「どれだけ大きな数を考えることができるか？」という議論をしてきました。中でも、「現代巨大数論の父」と言われるジョナサン・バウアーズは、BEAF という巨大数を生み出す難解なシステムを開発しただけではなく、その BEAF によって定義される巨大数に、たくさんの面白い名前をつけました。巨大数愛好家の中には、BEAF の定義はよく分からないけど、そういった面白い名前の巨大数を眺めて楽しむ、といったような人たちもたくさん出てきました。そのようなカジュアルな楽しみ方から、より効率よく巨大数を生み出すためにはどうすれば良いかという理論的な探求をする人まで、いろいろな人たちが巨大数を楽しむコミュニティが形成されています。

日本では、2002 年からインターネットの掲示板「巨大数探索スレッド」で、巨大数探索が繰り広げられてきました。その中では数学の新しい概念を少しずつ取り入れながら議論が発展し、多変数アッカーマン関数、ふぃっしゅ数、バシク行列などが生まれました。本書の著者は、ここに「ふぃっ

しゅ数」という「お遊び系巨大数」を書き込んで、巨大数に関する議論が活発になるきっかけを作った者です。

2013 年には巨大数をテーマにした漫画『寿司　虚空編』の連載が開始され、2016 年には岩波書店から『巨大数』という書籍が刊行され、数学のイベント「ロマンティック数学ナイト」における『寿司　虚空編』作者による巨大数解説の様子が日本テレビの番組で紹介されるなど、近年特に巨大数への注目が集まっています。そのきっかけとなったのが、2007 年にネットで公開しながら執筆を開始して、2013 年に初版を公開した本書『巨大数論』です。ホームページから PDF ファイルをダウンロードして読むことができる手軽さから、巨大数に興味を持つ人たちに広く読まれてきました。

そしてこのたび初版から大幅に加筆・更新したこの第 2 版を発行することとしました。著者名は「ふぃっしゅっしゅ」から「フィッシュ」に変えました。紙の本で読みたいという声も寄せられているため、単行本（ペーパーバック）と PDF 版の同時公開とします。

本書では、章が進むにつれて次第に大きな巨大数を取り扱います。章が 1 つ進むごとに、前の章で取り扱っていた数は「無視できるほど小さな数」となってしまうほど飛躍的かつ巨大数の構成方法に関する本質的な変化がそこにはあります。

第 1 章の「巨大数入門」では、いわゆる指数表記を重ねていくことで、その大きさの程度を表記できるような大きさの数を取り扱います。無量大数や不可説不可説転、グーゴル、グーゴルプレックス、スキューズ数といったような有名な巨大数は、ほとんどがこの章に入ります。第 2 章では指数表記を重ねる回数を数える「テトレーション」という演算をはじめとするハイパー演算を表記するためのクヌースの矢印表記を導入します。第 3 章では、有名なグラハム数やコンウェイのチェーン表記を紹介します。第 3 章まではすでに一般によく知られている内容ですが、第 4 章からは、巨大数論独自の混沌とした世界に入っていきます。そして、巨大数を作り、その解釈をするために数学基礎論、数理論理学、証明論、計算機科学のような数学の分野から、再帰理論、順序数、逆数学、述語論理、計算可能性理論のような理論を導入します。これらの理論は、そもそもは巨大数とは無関係に発展してきたものですが、多くの巨大数愛好家たちが巨大数を作って

評価するという目的で議論を重ねて、色々な数学の成果を取り入れてきたものを、巨大数論という体系として取りまとめたものが本書です。巨大数論の立場からは、それらの理論がまるで巨大数を作るために発展してきたかのように利用することになります。

このように数学の幅広い分野の専門的な内容を含むため、本書の内容は次第にハードになっていきます。より多くの人にスマホや電子書籍リーダーで気軽に読んでいただけるように、第 1 章を独立させて『巨大数入門』[1] という Kindle 版の電子書籍として出版しています。

本書『巨大数論』第 2 版は、紙の単行本を Amazon で販売し、単行本と同じ内容の電子書籍（**PDF** 版）をホームページ[2] で無料公開しています。また、クリエイティブ・コモンズ 表示 - 継承 4.0 国際[3]の条件で自由に印刷・複製や再配布をしていただくことができます。紙の本でも電子書籍でもプリントアウトでも、皆様が読みやすい方法でお読みください。

本書には著者が巨大数の探求をはじめてから 15 年間の多くの人たちとの議論が凝縮されています。そして、ネットに公開した PDF 版に多くのご指摘をいただいたことで改良を重ねてきました。特に、Limit of Empty さん（巨大数研究 Wiki 設立者）、ミカヅキモさん（数学者）、MUPI さん、noan さんには、丁寧に読んでいただき多くのご指摘をいただきました。ここに記して感謝の意を表します。

2017 年 6 月 29 日

本書は多くの方にご愛読いただいたおかげで、インプレスグループ創設 25 周年企画 POD 個人出版アワードで審査員特別賞（窓の杜賞）を受賞しました[4]。読者の皆様に感謝します。いくつかの誤記を修正した 2 版 2 刷を発行します。修正点はホームページをご参照ください。

2018 年 10 月 13 日

フィッシュ

[1] 巨大数入門 Kindle 版 https://www.amazon.co.jp/dp/B01N4KCIJQ
[2] 巨大数論（本書）のホームページ http://gyafun.jp/ln/
[3] CC BY-SA 4.0 https://creativecommons.org/licenses/by-sa/4.0/deed.ja
[4] 「POD 個人出版アワード」受賞作品決定！ http://www.impressrd.jp/news/180320/NP

目 次

はじめに **3**

第 **1** 章　巨大数入門 **11**

1.1　クラス 2 の巨大数 11
1.1.1　数のクラス分け 11
1.1.2　無量大数 13
1.1.3　グーゴル 17
1.1.4　シャノン数 21
1.1.5　数の比較 25
1.2　クラス 3 の巨大数 27
1.2.1　アルキメデスが考えた数の単位 28
1.2.2　不可説不可説転 31
1.2.3　グーゴルプレックス 32
1.2.4　ハイパー E 表記 33
1.2.5　プロマキシマ 34
1.2.6　超過剰数 36
1.3　クラス 4 の巨大数 37
1.3.1　グーゴルデュプレックス 38
1.3.2　スキューズ数 39
1.4　クラス 5 以上の巨大数 40
1.4.1　宇宙論で使われた最大の数 40
1.4.2　ベントレー数 41
1.5　巨大数に関する参考資料 44

第 2 章 原始再帰関数 **45**
2.1 クヌースの矢印表記 45
2.1.1 矢印表記 45
2.1.2 指数 46
2.1.3 テトレーション 49
2.1.4 ハイパー演算子 49
2.1.5 連続関数化 54
2.2 グッドスタイン数列 57
2.3 原始再帰関数 61

第 3 章 2 重再帰関数 **65**
3.1 アッカーマン関数 65
3.2 2 重再帰関数 68
3.3 モーザー数 72
3.4 グラハム数 75
3.5 コンウェイのチェーン表記 76

第 4 章 多重再帰関数 **81**
4.1 多変数アッカーマン関数 81
4.2 多重再帰に見えてそうでない関数 84
4.3 拡張チェーン表記 86
4.4 ふぃっしゅ数 87
4.4.1 ふぃっしゅ数バージョン 1 88
4.4.2 ふぃっしゅ数バージョン 2 100
4.4.3 ふぃっしゅ数バージョン 3 103
4.5 バード数から配列表記へ 106

第 5 章 順序数 **115**
5.1 順序数 116
5.1.1 整列集合と順序型 116
5.1.2 超限順序数とフォン・ノイマンの順序数表記 120
5.1.3 極限順序数の基本列 122
5.1.4 カントールの標準形 124

5.2 順序数階層 125
5.2.1 ハーディ階層 126
5.2.2 急増加関数 130
5.2.3 緩増加関数 138
5.3 順序数解析 139
5.3.1 形式体系 140
5.3.2 一階述語論理 140
5.3.3 算術の理論体系 143
5.3.4 順序数解析 145

第 6 章 ペアノ算術の限界 147
6.1 $F[\epsilon_0](n)$ の計算 147
6.2 ドル関数の角括弧表記 149
6.3 ふぃっしゅ数バージョン 5 151
6.4 ヒドラゲーム 154
6.5 原始数列数 160
6.6 多重リストアッカーマン関数 167
6.7 バードのネスト配列表記 169
6.8 BEAF のテトレーション配列 170
6.9 巨大数生成プログラム 175

第 7 章 再帰関数 177
7.1 ヴェブレン関数 177
7.2 順序数崩壊関数 181
7.2.1 非可算順序数と共終数 181
7.2.2 フェファーマンの θ 関数 183
7.2.3 ブーフホルツの ψ 関数 187
7.2.4 マドールの ψ 関数 187
7.2.5 ワイヤーマンの ϑ 関数 192
7.2.6 順序数崩壊関数の拡張 193
7.3 2 階算術 194
7.3.1 逆数学 194
7.3.2 2 階算術 194

7.3.3 論理式の階層 196
7.3.4 2 階算術の部分体系と証明論的順序数 197
7.3.5 タラノフスキーの C 表記 200
7.4 様々な巨大数と関数 200
7.4.1 ふぃっしゅ数バージョン 6 201
7.4.2 ペア数列数 205
7.4.3 サブキュービックグラフ数 215
7.4.4 ブーフホルツのヒドラ 216
7.4.5 BAN 219
7.4.6 BEAF 220
7.4.7 バシク行列システム 222
7.4.8 強配列表記 224
7.4.9 ローダー数 226
7.4.10 超越整数と巨大基数による拡張 229
7.4.11 有限約束ゲーム 232
7.4.12 欲張りクリーク列 234

第 8 章 計算不可能な関数 239
8.1 ビジービーバー関数 239
8.2 神託機械 243
8.3 クサイ関数 244
8.4 ふぃっしゅ数バージョン 4 245
8.5 ラヨ数 248
8.6 ふぃっしゅ数バージョン 7 251
8.7 ビッグフット 255
8.8 サスクワッチ 257

おわりに 259

巨大数年表 263

第 1 章
巨大数入門

本章では、数の大きさをクラス分けすることによって、段階的に巨大数を見ていきます。本章で解説をする巨大数が、本書の著者を含めて普通の人間がその大きさについて想像をめぐらせて「これは大きな数だなぁ」と気が遠くなって驚くような巨大数の限界だと思います。ところが、それは巨大数論では入門レベルということになります。

1.1 クラス 2 の巨大数

1.1.1 数のクラス分け

ロバート・ムナフォ (Robert Munafo) は、1996 年に開設したホームページ[1] で、様々な巨大数について紹介し、理論的に考察しています。その説明に従って、段階的に大きな数について記します。ムナフォは、数の大きさをクラス分けして、クラス 0 の数、クラス 1 の数、クラス 2 の数、クラス 3 の数…と分類しました。日常生活で使う数は、クラス 2 の数までです。さらに無量大数やグーゴルといった有名な巨大数も、クラス 2 になります。そこで本節ではクラス 2 の数までについて見てみます。

クラス 0 の数 (0–6): クラス 0 は、カウフマンが提唱した subitizing という概念で[2] 、即座に認識できる数です（図 1.1)。たとえば、1 本の鉛筆

[1] Large Numbers http://www.mrob.com/pub/math/largenum.html

[2] Kaufman, E. L. et al. (1949). The discrimination of visual number. *American Journal of Psychology* 62 (4): 498–525. doi:10.2307/1418556

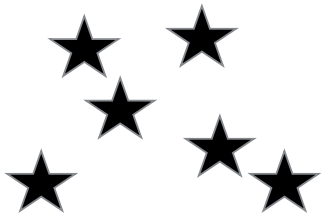

図 1.1: 一瞬で認識できる数。

があれば、それは 1 本であると一瞬で認識できます。2 本であっても、一瞬で認識できます。5 本の鉛筆はどうでしょうか。目をつぶって、目を一瞬あけてすぐに閉じたときに、見た鉛筆の本数を目を閉じた状態で言えるかとなれば、5 本くらいであれば誰でも言えるでしょう。このように、いちいち数えなくても即座に認識可能な数は、最大 5 から 9 程度であると言われていますが、ムナフォは低めに見積もって 6 であるとして、0 から 6 までの数をクラス 0 としました。

クラス 1 の数 (6–10^6): クラス 1 は、人間の目でおおまかな数が認識できる数です。この上限を、ムナフォは 100 万であるとしました。たとえば、大きな紙に 100 万個の点を印刷して、その 1 つ 1 つの点を見分けながら、残りのすべての点が視野に入るようにします。ムナフォの実験で本人はそれができましたが、人によってはもっと大きな数でも認識できるかもしれません。クラス 1 の数であれば、正確な数は分からなくても、大体の大きさが認識できます。たとえば、部屋の中に 85 人の人がいれば、70〜100 人程度かな、と分かります。もっとたくさんの人がいるときには、何千人とか何万人といった表現をします。このように、正確な数は分からなくても、だいたいの数を認識できる限界をクラス 1 として、ムナフォはその上限を 100 万としました。

クラス 2 の数 (10^6–10^{10^6}): クラス 2 は 10 進数で記録できる数です。クラス 0 の 6 からクラス 1 の $10^6 = 1000000$ へと増やしたように、分かりやすくクラス 2 は $10^{10^6} = 10^{1000000}$ までとされました。ここで指数を 2 つ

以上重ねたときには、右上から順番に計算する、ということに気をつける必要があります。つまり $10^{10^6} = 10^{(10^6)}$ です。左から順番に計算すると、$(10^{10})^6 = 10^{10\times 6} = 10^{60}$ となって、指数が掛け算になってしまいますから、右から計算した場合と比べてとても小さい数になります。クラス 2 は、100 万桁までの数で記録できる数ということです。クラス分けの境界は、全体として大ざっぱなものです。子供に「できるだけ大きな数を書いてごらん」と言って紙を渡せば、多くは「999999…」と書き続けるでしょう（111111… と書く方が楽ですが）。根性のある子供でも 100 万桁はなかなか書き続けられないと思います。何桁記録できるのかは、どこにどのように記録するのかにもよります。パソコンにデータを保存するのは簡単なので、100 万桁よりもずっと大きな桁数を保存できます。ここでは紙と鉛筆でものすごく頑張れば 10 進数表記で書ける数、と解釈すると、100 万桁という上限がちょうどいい感覚になりそうです。

1.1.2 無量大数

まずは日本での大きな数の数え方について記します。それからその由来となった中国とインドにさかのぼって調べたところを記します。

私達が日常生活で扱う数は、クラス 2 の数までです。したがって日常生活で使う数に関する言葉も、だいたいこの範囲の言葉になります。そこでまずは、日本語での数の数え方から見ていきます。それからその源泉をたどって中国、インドの数について見ます。

数の位は、一、十、百、千、万と 10 倍ずつになり、その先は 1 万倍ずつ、億 $= 10^8$、兆 $= 10^{12}$、京（けい）$= 10^{16}$、垓（がい）$= 10^{20}$、秭（じょ）$= 10^{24}$、穰（じょう）$= 10^{28}$、溝（こう）$= 10^{32}$、澗（かん）$= 10^{36}$、正（せい）$= 10^{40}$、載（さい）$= 10^{44}$、極（ごく）$= 10^{48}$、恒河沙（ごうがしゃ）$= 10^{52}$、阿僧祇（あそうぎ）$= 10^{56}$、那由他（なゆた）$= 10^{60}$、不可思議（ふかしぎ）$= 10^{64}$、無量大数[3]（むりょうたいすう）$= 10^{68}$ となります。垓以上の単位を日常生活で使うことはめったになく、特に意識しなければ知らない人も多いでしょう。著者は子供の頃から数が好きだったので、父

[3]巨大数研究 Wiki - 無量大数 http://ja.googology.wikia.com/wiki/無量大数

に教えてもらって無量大数までの桁を覚えて喜んでいました。当時は無量大数は 10^{88} であると覚えていて、これは恒河沙から上は 10^8 ずつ増える万万進という方式に基づいていたためですが、現在は万進に統一されていますので、正しくは 10^{68} になります。その根拠として、1627 年に初版が出た吉田光由の『塵劫記』では、寛永 8 年版 (1631) で無量大数が 10^{68} または 10^{88} として紹介され、寛永 11 年版 (1634) では 10^{68} のみに統一されているため、現在では 10^{68} に統一されていると考えることができます。

【定義】無量大数

無量大数 $= 10^{68}$

さて、高杉親知の「無量大数の彼方へ」[4] というサイトによれば、昔の中国では、後漢の時代に載までの単位が記されていて、億以上の単位については、「下数」「中数」「上数」の 3 つの体系があり、さらに「中数」には「万進」「万万進」という体系があって、合計 4 つの体系それぞれで、同じ言葉でも表す数が違っていました。下数では、次の単位では 10 倍になるため、万=10^4、億=10^5、兆=10^6 と少しずつ上がり、載=10^{14} になります。中数では、万進では 1 万倍ずつ、万万進では 1 億倍ずつ上がります。したがって万進の兆は現在の日本と同じ 10^{12} になるのに対して、万万進では「万億」が 10^{12} になります。そして万万進の兆は 10^{16} で、万進の京と同じになります。載は万進では 10^{44}、万万進では 10^{80} になります。中国では、万進と万万進が統一されずに近代化が進んだので混乱が生じましたが、日本では、万進の表記に統一されたので混乱が生じないようになりました。

下数よりも中数よりも、飛躍的に数の増加が大きいのが**上数**です。上数では単位を 2 乗すると次の単位になります。したがって、億=$10^8 = 10^{2^3}$、万億=10^{12}、千万億=10^{15}、兆=$10^{2^4} = 10^{16}$、万兆=10^{20}、億兆=10^{24}、万億兆=10^{28}、千万億兆=10^{31}、京=$10^{2^5} = 10^{32}$、億京=10^{40}、兆京=10^{48}、億兆京=10^{56}、万億兆京=10^{60}、垓=$10^{2^6} = 10^{64}$、千万億兆京垓=10^{127}、秭 = $10^{2^7} = 10^{128}$、穣=$10^{2^8} = 10^{256}$、溝=$10^{2^9} = 10^{512}$、澗=$10^{2^{10}} = 10^{1024}$、正 =$10^{2^{11}} = 10^{2048}$、載=$10^{2^{12}} = 10^{4096}$、千万億兆京垓秭穣溝澗正載=10^{8191} と、現在の定義である無量大数とは比べ物にならないほどの大きな数にな

[4] 無量大数の彼方へ http://www.sf.airnet.ne.jp/~ts/language/largenumber.html

ります。それでも指数の部分が 100 万に達していないので、まだクラス 2 です。

載よりも上の単位である極から上は、元の朱世傑（しゅせいけつ、生没年不詳）による『算学啓蒙』(1299 年) において登場しました。ここで極以外の単位はすべて仏教から取られたものです。恒河沙は「ガンジス川の砂」の意味で、それだけ大きな数ということです。阿僧祇は「数えることができない」、那由他は「極めて大きな数量」、不可思議「思い測ったり言葉で言い表したりすることができない」無量大数は、『算学啓蒙』では「無量数」として登場し、「量ることのできない数」ということです。

この中で恒河沙以外は「数え切れないほどの大きな数」と理解すれば良いのですが、恒河沙については、ガンジス川の砂の数という具体的な数になっているので、どの程度の大きさなのか見積もってみます。今は恒河沙と言えば日本では 10^{52} という意味になり、中国でも下数、中数、上数によって数は変わります。仏教の中ではそういう数を意味するものではなく、ガンジス川の砂の数という意味ですから、今の日本で恒河沙を意味する 10^{52} と比較することは無意味です。

まずはガンジス川の大きさと砂粒の大きさを考える必要があります。ガンジス川の全長は 2500 km あまりです。川幅は数 km なので 4 km 程度と考えて、面積を 10000 km^2 とします。深さをどこまで取るのかは分かりませんが、仮に 1 m の深さまで考えると、体積は 10 km^3 になります。問題は、砂の大きさです。砂は粒子の直径が 0.05–2 mm のもので、それよりも小さい直径のものはシルトあるいは粘土と言われます。当時のインドにはそのような概念はなかったのでしょうから、全部ひっくるめて砂と呼ばれていたことでしょう。そうすると、現在では非常に小さくて粘土と呼ばれているような粒子のものも、砂に含めて考えるのかどうか、これによって値はだいぶ変わってきます。ここでは暫定的に直径が 0.1 mm であると考えます。次に充填率を考える必要がありますが、充填率が低くて 50 % であったとしても、充填率が 100 % であると仮定して計算したときの 2 倍程度の誤差なので、これは許容誤差であると考えて充填率 100 % でざっくりと計算すると、10 $\mathrm{km}^3 = 10^{10}\,\mathrm{m}^3$ の中に 0.001 $\mathrm{mm}^3 = 10^{-12}\,\mathrm{m}^3$ の粒子を隙間無

く埋めた時の粒子の個数なので、10^{22} となります。今の日本の数え方だと、100 垓程度であると見積もられます。

仏教では、とても長い時間を表すときに劫（こう）という時間の単位が使われます。「未来永劫」「億劫」「五劫のすり切れ」などの言葉は、ここから来ています。大谷大学の木村宣彰教授によれば[5] 、この「劫」には 2 通りの意味があり、1 つ目は「磐石劫」で、四十里四方の大石を、いわゆる天人の羽衣で百年に一度払い、その大きな石が摩滅して無くなってもなお「一劫」の時間は終わらないと譬えています。2 つ目は「芥子劫」で、方四十里の城に小さな芥子粒を満たして百年に一度、一粒ずつ取り去り、その芥子がすべて無くなってもなお尽きないほどの長い時間が一劫であるとのことです。

この時間がどの程度のものなのか「カガクの時間」というサイトでは、次のように計算しています。「磐石劫」では[6] 、「1 里=500m」「原子=1 辺 0.2nm の立方体」「1 回なでると、1 平方メートルの範囲の原子が 1 層はがれる」という仮定の下、4 秭 $= 4 \times 10^{24}$ 年という計算結果を出しました。2 つ目の「芥子劫」については[7] 、「1 里=500m」、「けし粒=1 辺 0.5mm の立方体」、という仮定の下、6 秭 4000 垓 $= 6.4 \times 10^{24}$ 年という計算結果を出しました。磐石劫でも芥子劫でも同じ秭の単位になるのは面白いところです。宇宙の年齢は 138 億年程度であるとされていますので[8] 、それと比べて 100 兆倍ほどの気の遠くなるほど長い時間です。恒河沙という数と劫という時間の単位を組み合わせれば、恒河沙劫というとても長い時間が表現できますので、そういう表現があるか検索してみたところ、『首楞厳経』という経典に恒河沙劫という表現が出てきて、親鸞が『尊号真像銘文』で引用していることが分かりました[9] 。恒河沙劫は、$10^{22} \times 10^{24} = 10^{46}$ 年=100 載年のオーダーになります。

---

[5] http://www.otani.ac.jp/yomu_page/b_yougo/nab3mq0000000r5r.html

[6] 天女が岩をなでたなら http://d.hatena.ne.jp/inyoko/20111015/How_long_is_kalpa2

[7] けし粒はいつなくなる http://d.hatena.ne.jp/inyoko/20111008/How_long_is_kalpa

[8] Universe as an Infant: Fatter Than Expected and Kind of Lumpy (New York Times) https://nyti.ms/YrtRRV

[9] WikiArc: ノート:首楞厳経 https://bit.ly/1b4Myz9 (labo.wikidharma.org)

1.1.3 グーゴル

ここまで、主にインド、中国由来の日本の数の数え方について見てきました。次に英語を中心としたヨーロッパの言語の話に移ります。

大きな数を表す言葉として、上記の億、兆…といった単位の他には、キロ (k, kilo-, 10^3)、メガ (M, Mega-, 10^6)、ギガ (G, Giga-10^9)、テラ (T, Tera-10^{12}) といった単位の頭につける接頭辞もあります。単位の頭にこれらの接頭辞をつけることで、キロメートル、キログラム、メガバイト、Mbps (メガビットパーセカンド)、メガピクセル、ギガバイト、テラバイトなどという単位ができます。

なお、ハードディスク等の記憶容量を表すときには、1 キロが 1000 ではなく $2^{10} = 1024$ を意味することがあります。これはコンピュータは 2 進数を基本としているので、1000 よりも 1024 の方が切りの良い数だからです。国際電気標準会議 (IEC) は、2 進数に基づいた単位につける 2 進接頭辞を定めています。たとえば 2^{10} はキロバイナリー (kilobinary) を略してキビ (kibi) となって Ki と書き、1000 バイトを意味する KB と区別します。同様に 2^{20} はメビ (Mi)、2^{30} はギビ (Gi)、2^{40} はテビ (Ti) となって、テラは $10^{12} = 1,000,000,000,000$ ですが、テビは $2^{40} = 1,099,511,627,776$ なので、その差は 10 % ほどになります。このように 2 進と 10 進の接頭辞で差異があって混乱するために、両者を明確に区別するために定められた 2 進接頭辞ですが、まだ十分に普及していません。

さてテラの上の接頭辞は、国際単位系の SI で、ペタ (P, Peta-10^{15})、エクサ (E, Exa-10^{18})、ゼッタ (Z, Zetta-10^{21})、ヨッタ (Y, Yotta-10^{24}) と定められています。現実に使う大きさとしては、杼=10^{24} 程度が上限であると考えられているということでしょう。

現実世界の現象をあらわす数としては、どの程度の数が最大となるのでしょうか。「天文学的数」という表現があるように、天文学では広大な宇宙のことを取り扱うため、非常に大きな数が使われます。「ガンジス川の砂の数」と言ったときには、とても広いガンジス川にある、とても小さい砂粒の数なので、非常に大きな数になりました。広い方の限界として宇宙の大きさ、小さい方の限界としては、素粒子の大きさを考えることになるでしょ

う。その両方の限界を取ると、宇宙に存在する素粒子の数はいくつか？ という疑問になります。

イギリスの天文学者アーサー・スタンレー・エディントン (Sir Arthur Stanley Eddington, 1882–1944) は、1938 年にケンブリッジ大学のトリニティ・カレッジでの講義 (Tarner lecture) で、宇宙に存在する全陽子の数が正確に 136×2^{256} 個あり、全電子数も同じ数だけある、としました。これが、**エディントン数**[10] (Eddington number) と呼ばれています。10 進数に直すと、1.574×10^{79} 程度です。現在では、陽子の数はもっと多いとされています。陽子よりも小さい素粒子の単位で考えると、その数は 10^{80}–10^{85} 程度であるとされています。実際、現実には 10 進数で 100 桁以上の数を扱うことはなさそうです。

英語では、million (100 万) や billion (10 億) のように、「〜リオン」という接尾語が使われています。このように、ヨーロッパ系の言語で「〜リオン」という接尾語で数の名称を体系的かつ非常に大きな桁まで名付けた最初の著作は、フランスの数学者ニコラ・シュケ (Nicolas Chuquet, 1450 年頃–1500 年頃) が書いた算術の 3 つの分野（有理数、無理数、方程式）に関する『数の学の三部』(Le triparty en la science des nombres) という中世フランス語で書かれた本です。この本はシュケの生きている間には出版されず、1520 年にエスティエンヌ・ド・ラ・ロシュ (Estienne de La Roche, 1470–1530) が『算術』(l'Arismetique) という本でシュケの出典を記さずにコピーしました。1870 年に、アリスティード・マール (Aristide Marre, 1823–1918) がシュケの著作の原稿を発見して、1880 年に刊行しました。その原稿には、ロシュの手書きメモがあったようです。シュケの本では大きな数を 6 桁ずつに区切って、そのグループを million, byllion, tryllion, quadrillion, quyillion, sixlion, septyllion, ottyllion, nonyllion のように好きなだけ名付けることができる、としています。この数え方だと、nonyllion が 10^{54} になります。

現代の英語の大きな数の表記は、このシュケのシステムが元となっていますが、million よりも上の数について、シュケが考えた 6 桁ごとに名前をつけるシステムだけではなく、3 桁ごとに名前がつけられるシステムがあります。つまり million は 100 万を意味しますが、その上の billion は 10 億

[10] http://ja.googology.wikia.com/wiki/エディントン数

をあらわす場合と 1 兆をあらわす場合があります。現在は、billion が 10 億をあらわすショートスケールが主流になっています。かつてイギリスでは billion が 1 兆をあらわすロングスケールが使われていましたが、現在はイギリスでもショートスケールが基本です。ただし、ヨーロッパ大陸や南米の英語以外の言語圏では、ロングスケールが使われているところもあります。たとえば、シュケの母語であるフランス語では今でも billion が 1 兆で trillion が百京のロングスケールのままです。ショートスケールとロングスケールの関係は、ちょうど万進と万万進の関係のようなもので、ショートスケールが千進、ロングスケールが千千進と言えば分かりやすいと思います。つまり million から先、billion, trillion, quadrillion, … と進んで行くごとに、ショートスケールでは千倍、ロングスケールでは千千倍、すなわち 100 万倍になります。

標準的な英語の辞書に定義されている大きな数は、ショートスケールでは million $= 10^6$, billion $= 10^9$, trillion $= 10^{12}$, quadrillion $= 10^{15}$, quintillion $= 10^{18}$, sextillion $= 10^{21}$, septillion $= 10^{24}$, octillion $= 10^{27}$, nonillion $= 10^{30}$, decillion $= 10^{33}$, undecillion $= 10^{36}$, duodecillion $= 10^{39}$, tredecillion $= 10^{42}$, quattuordecillion $= 10^{45}$, quindecillion $= 10^{48}$, sexdecillion (sedecillion) $= 10^{51}$, septendecillion $= 10^{54}$, octodecillion $= 10^{57}$, novemdecillion (novendecillion) $= 10^{60}$, そして、ビギンティリオン[11] (vigintillion) $= 10^{63}$ (ロングスケールでは 10^{120}) までが連続的に定義されていて、次はセンティリオン[12] (centillion) $= 10^{303}$(ロングスケールでは 10^{600}) まで飛ぶのが一般的です。

また、それとは別にグーゴル[13] (googol) $= 10^{100}$ とグーゴルプレックス (googolplex) $= 10^{10^{100}}$ も英語の辞書に載っている標準的な数です。

【定義】グーゴル

グーゴル $= 10^{100}$

グーゴルとグーゴルプレックスの語源は数学者エドワード・カスナー (Edward Kasner, 1878–1955) らの『数学と想像力』(Mathematics and the

[11] http://ja.googology.wikia.com/wiki/ビギンティリオン
[12] http://ja.googology.wikia.com/wiki/センティリオン
[13] http://ja.googology.wikia.com/wiki/グーゴル

Imagination) [14] という本に、次のように書かれています。

> 知恵の言葉は科学者よりむしろ子供たちから発せられる。「グーゴル」という名前は子供（カスナー博士の 9 歳の甥）によって発明された。彼は、とても大きい数、つまり 1 の後に 0 が 100 個続く数の名前を考えて欲しいと頼まれた。彼は、この数が無限ではないことをはっきりと分かっていたので、名前がなければなければならないことも分かっていた。「グーゴル」という名前を提案すると同時に、さらに大きい数である「グーゴルプレックス」を名付けた。

カスナーの甥であるミルトン・シロッタ (Milton Sirotta) が 9 歳の時に考えた数で、ミルトンは 1911 年生まれなので 1920 年にグーゴルが生まれたことになります。

なお、グーゴルプレックスはクラス 3 の数で改めて説明します。

検索エンジンの名前、そして IT 企業の名前である Google は、googol に由来しています。Google が生まれたのはスタンフォード大学で、当初 Google の共同創業者であるラリー・ページ (Lawrence Edward “Larry” Page) とセルゲイ・ブリン (Sergey Mikhailovich Brin) は BackRub と呼んでいました。1997 年 9 月に、ラリーがスタンフォード大学の研究室で当時大学院生だったショーン・アンダーソン (Sean Anderson) と検索エンジンの新しい名前について話をしていたときの様子が、次のように記録されています。これは、2004 年にスタンフォード大学の大学院生だったデイビッド・コラー (David Koller) がショーンから聞いた話です[15] 。

> ショーンとラリーは、研究室でホワイトボードを使いながら良い名前について考えていました。非常に膨大なデータをインデックスする、ということに関係のある名前がいいと考えていました。ショーンが非常に大きな数である「グーゴルプレックス」はどうかと口頭で提案したところ、ラリーは「グーゴル」がいい

[14] Kasner, E. and Newman, J. R. (1940) Mathematics and the Imagination. Simon & Schuster, New York.

[15] http://graphics.stanford.edu/~dk/google_name_origin.html

> と答えました。ショーンはコンピュータの端末に座っていたので、インターネットのドメイン名登録データベースを検索して、その名前が登録可能かどうか調べました。そのときに、ショーンは googol の綴りを正確に覚えていなかったので google.com という名前を探すというミスをしてしまいました。ラリーはその名前が気に入って、直後に google.com を登録する手続きをしました。

つまり、googol のスペルミスから google.com が登録されたというのが Google の由来となっているのです。

1.1.4 シャノン数

シャノン数[16](Shannon number) は、10^{120} です。これは、1950 年に情報理論の父として知られるアメリカの数学者、電気工学者のクロード・シャノン (Claude Elwood Shannon, 1916–2001) が書いた論文[17] に出てくる数です。この論文は、チェスの指し手を読むコンピュータを開発することの意義について考察したものです。

当時は、1949 年に EDSAC という最初のノイマン型コンピュータが作られた頃でした。シャノンは、コンピュータにチェスをさせることは「おそらくまったく実用的な重要性はないが、理論的な関心であり、この問題に対する満足な回答が得られれば、同じような他の同様の性質を持つ問題を解決する役に立つだろう」としています。

この論文では、まずチェスというゲームについて記述してから、チェスというゲームを完全に解析して完全ゲームをさせることができるだろうか、と疑問を発して、それは原理的には可能であるが現実的には無理だろうとしています。その理由を説明するために、完全解析をするには何手読む必要があるかを見積もっています。

まず通常のチェスの局面では可能な駒の動かし方は 30 通り程度であり、多くのマスター（とてもチェスが強い人）のゲームを解析すると、ゲームの

[16]巨大数研究 Wiki - シャノン数 http://ja.googology.wikia.com/wiki/シャノン数

[17]Shannon, C. (1950) Programming a computer for playing chess. *Philosophical Magazine* 41 (314)

最初から終わりまで、あまりこの数が変化しないという結果を示して、チェスの 1 手、つまり先手の白が 1 つ駒を動かして、後手の黒が 1 つ駒を動かすセット（将棋では 2 手と数えられる）は、およそ 10^3 通りの可能性があるとしました。そして、通常のゲームは投了するまでに 40 手ほどであると見積もりました。コンピュータは投了までではなくチェックメイトまで読むので、本当はもっと手数が長くなることを考えると、控えめの見積もりだとしています。

このような控えめな見積もりでも、初期配置から 10^{120} 通りの変化を読む必要があり、1 つの変化を 1 マイクロ秒で読めるコンピュータができたとしても、初手を読むためには 10^{90} 年以上かかってしまいます。チェスには 10^{120} 通りの変化があるというこの計算から、10^{120} をシャノン数と言います。

【定義】シャノン数

シャノン数 $= 10^{120}$

シャノンはまた、チェスのあらゆる可能な局面を記憶する「辞書」を作る方法についても考察し、チェスの全局面は $\frac{64!}{32!(8!)^2(2!)^6}$ 通り（64 のます目に、白と黒で合計 32 個の駒を並べる順列、ただしそのうち 8 つ同じ種類の駒があるものが 2 種類（ポーンの白と黒）、2 つ同じ種類の駒があるものが 6 種類（ルーク、ビショップ、ナイトそれぞれ白と黒）がある）あり、概算で 10^{43} 通りの局面があるため、それもまた非現実的だとしています。このように、チェスの完全解析をコンピュータにさせることは非現実的であるとして、コンピュータにチェスをさせるための戦略についてシャノンは論文で考察しています。

なお、シャノン数はシャノン自身が「控えめな見積もり」としていますが、相当に控えめな見積もりです。実際にチェスを完全解析するために「手当たり次第に」読ませたとすれば、コンピュータは人間のように「試合を終わらせようとする手」だけではなく、「お互いにまったく攻めようとせずに自陣で駒を動かし続けるような試合」も数多く出て、そのような手順をすべて読むことが「完全解析」なので、手数は 40 手どころではなく果てしなく続きます。イギリスの数学者ゴッドフレイ・ハーディ (Godfrey Harold

Hardy, 1877–1947) は「宇宙の陽子の数は 10^{80} である。チェスのゲームの可能な数は、それよりもはるかに大きく、おそらく $10^{10^{50}}$ である」としています[18] 。これはおそらく「同一局面が 3 回で引き分け」のルールを採用した場合で、10^{43} 通りの局面が 3 回繰り返さないところまで読むと考えると $(10^3)^{10^{43}\times 2} = 10^{6\times 10^{43}}$ 程度となるだろうと思います。

ロバート・ムナフォは、チェスの 50 手ルールと 75 手ルールを採用し、白黒双方が協力して 75 手ルールで強制的に引き分けとなるまで引き分けを宣言しないとしたときに、最長手数は 8848 手となると計算して、チェスのゲームの可能な数はおよそ $26^{8848} \approx 10^{12500}$ 通りになるとしています。

なお、完全解析の難しさを考えるときには、このような「可能なゲームの数」を考えるよりも「可能な局面の数」である 10^{43} を使う方が適切だろうと思います。もちろん可能な局面の数を保存しておくメモリを仮定することは非現実的ではありますが、すでに非現実的な時間を想定しているわけですから、メモリについて非現実的な仮定に躊躇をする理由がありません。読み終わった全ての局面を保存しておくことが可能であるとして完全解析の難しさを評価するのが妥当でしょう。

その後コンピュータは発展し、1997 年 5 月には、IBM が開発したチェス専用のスーパーコンピュータ「ディープ・ブルー」が、当時のチェス世界チャンピオン、ガルリ・カスパロフに 6 戦中 2 勝 1 敗 3 引き分けで勝利しました。一方、1997 年の時点では、将棋のプログラムはまだ弱く、とてもプロ棋士の相手にはなりませんでした。将棋はチェスよりも複雑なので、そう簡単にチェスのようには強くならないと思われていました。

将棋の実現可能局面数はチェスよりもはるかに多く、10^{62}–10^{70} 程度とされています[19]。また、将棋の世界でもシャノン数と同じ計算方法、つまり一局の平均手数が約 115 手、各局面での可能な指し手の数が約 80 通りであるという計算から、$80^{115} \approx 10^{220}$ 通りのパターンがある、というような説明がよくされています（これは、チェスと同じ理由であまり適切ではない

[18] Hardy, G. H. (1940) Ramanujan: twelve lectures on subjects suggested by his life and work. Cambridge University Press.

[19] 篠田正人 (2008) 将棋における実現可能局面数について, IPSJ Symposium Series 2008(11) 116–119.

と思いますが、シャノン数と比較するという目的にはかなっています)。この数が、後に「不可説不可説転」で説明をする予定の、仏教の経典である華厳経に出てくる阿伽羅[20]（あから）という数の見積もりである 10^{224} に近いことから、情報処理学会では「あから」というコンピュータ将棋が開発されました。なお、情報処理学会のホームページには「阿伽羅（あから）は 10 の 224 乗という数を表し、将棋の局面の数がこの数に近いことに因んで命名されました。」[21]と説明されていますが (2016 年 6 月 25 日現在)、「将棋の局面の数」という表現であれば、10^{70} 以下という見積もりになるだろうと思います。

さてこのように、将棋はチェスよりもずっと複雑だから当面はコンピュータが人間に勝つことはない、と 20 世紀末には多くの人が考えていたものの、その後コンピュータはみるみる強くなり、2010 年には「あから」が女流棋士の清水市代を敗り、2012 年からはプロ棋士とコンピュータソフトが戦う「電王戦」が開始され、2013 年にはプロ棋士 5 人とコンピュータソフト 5 種類の団体戦で、プロ棋士が 1 勝 3 敗 1 分となり、現役のプロ棋士が敗れました。

将棋よりもさらに局面が広くて複雑であるとされる囲碁についても、2016 年 3 月に、グーグルが開発した囲碁プログラム「アルファ碁」(AlphaGo) がイ・セドル九段に 4 勝 1 敗で勝ちました。囲碁の全局面数は、シャノン数と同じ考え方で 10^{360} 程度とされていることが多いですが、それは適切ではありません。盤面の状態は $3^{19^2} \approx 1.74 \times 10^{172}$ 通りありますが、その中で実現可能な局面の数は 2016 年 1 月 20 日に正確な値が計算されました[22]。その値はおよそ $2.081681994 \times 10^{170}$ であり、この近似値は 2006 年にすでに知られていました。

シャノン数をきっかけとして、チェス、将棋、囲碁のプログラムについて話しましたが、これらのプログラムは最強の人間を負かすほどに強くなったとはいえ、「完全解析」ができる時は来ないでしょう。それは「全局面数」が桁違いに大きいためですが、巨大数論の世界では「高々クラス 2 程度の

[20]巨大数研究 Wiki - 阿伽羅 http://ja.googology.wikia.com/wiki/阿伽羅
[21]コンピュータ将棋「あから」 https://www.ipsj.or.jp/event/shogi.html
[22]Number of legal Go positions https://tromp.github.io/go/legal.html

数」です。

ロバート・ムナフォは、6 才の時には知っていた最大の数が million (100 万) で、7 才の時に母親にその上の単位を聞いて、billion (10 億) と trillion (兆) の単位を知り、さらに 8 才の時に学校の図書館で読んだ本でビギンティリオン (10^{63}) という数を覚えて嬉しくなり、校庭の砂に

1 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000

と書いて、いじめられたという過去があるようです。

コンウェイとガイの『数の本』[23] では、ビギンティリオンの先を、ショートスケールでは 10^{3003} まで、ロングスケールでは 10^{6000} まで、つまりミリニリオン (millinillion) = 10^{3003} あるいはミリリオン[24] までラテン語を元に名付ける体系が提案されています。

日常生活で使うのは、クラス 2 の中でも非常に小さい数までですので、数に名前がつけられるのも、たいていはここまでです。

1.1.5　数の比較

それではここで、表 1.1 にクラス 2 の巨大数を、小さいものから大きいものへと順番にまとめます。そして、このような巨大数の大きさをどのように比較するのかを考えます。

クラス 2 までの数は、数の大小比較が簡単にできます。10 進数で書かれていれば、まずは桁数を比較して、桁数が同じであれば一番上の桁から順番に大きさを比較すれば良いのです。数が 3.23×10^{21} のような指数表記で書かれていれば、桁数を数える必要もないので楽です。ただし、10 進数表記あるいは 10 進数の指数表記同士で書かれていない数を比較するときには、計算をする必要が出てきます。計算は手計算でもできますが、電卓を使えば簡単です。電卓には、スマホのアプリやパソコンのソフト等、色々あり

[23] Conway, J. H. and Guy, R. K. (1996) The book of numbers. Copernicus. 訳書『数の本』丸善出版.

[24] http://ja.googology.wikia.com/wiki/ミリリオン

表 1.1: クラス 2 の巨大数（下の数ほど大きい）

名称	値
恒河沙劫の年数の見積もり	10^{46}
無量大数	10^{68}
エディントン数	$\approx 1.57 \times 10^{79}$
グーゴル	10^{100}
シャノン数	10^{120}
センティリオン	10^{303}
ミリニリオン	10^{3003}
上数の載	10^{4096}
クラス 2 の数の上限	$10^{1000000} = 10^{10^6}$

ます。性能によって、表示できる最大の数（指数表示の時の最大の指数)、表示される桁数、内部で計算されている桁数、等が変わります。

カシオが運営している高精度計算サイト[25] は、強力です。「生活の計算」「数学・物理」「専門的な計算」それぞれ様々な実用的な計算をすることができて、実用的にも便利ですし、なんといっても「フリー計算」の機能が良いので、本書のような趣味の計算をするときにも便利に使えます。様々な関数が扱え、桁数は 50 桁までの精度が得られ、さらに精度保証の機能がついています。コンピュータで計算をするときには、計算機の内部で保持している変数の桁数に限界があるので、限界以上の桁は丸めて計算をするため、計算結果に誤差が出ます。そこで、計算結果を丸めるときに、必ず「最小の値」と「最大の値」を同時に計算することで、最大でどの程度の誤差が生じるかを正確に知る手法が、精度保証つき計算です。

この高精度計算サイトでエディントン数を 50 桁まで計算すると、

$$1.5747724136275002577605653961181555468044717914527 \times 10^{79}$$

と計算できます。しかしエディントン数は 80 桁あるので、最後の 30 桁が

[25]高精度計算サイト (CASIO) http://keisan.casio.jp/

計算できていません。そこで桁数に制限のない任意精度計算機[26] [27] を使えば、エディントン数を正確に計算できます。計算結果は、

1574772413 6275002577 6056539611 8155546804 4717914527 1167093662 3142507618 5631031296

となります。このような計算機を使えば、クラス 2 の数はだいたい大小比較ができます。

手元の電卓ではエラーが出てしまうような大きな数を計算するときには、対数 (log) を使うと、大きさを知ることができます。対数には、底を 10 とする常用対数と、底を $e \approx 2.71828$ （ネイピア数）とする自然対数があるので、常用対数で計算したときにはその数 a に対する 10^a を、自然対数で計算したときには $\exp(a) = e^a$ を計算することになります。ここでは、常用対数を使います。エディントン数 136×2^{256} の常用対数の大きさを

$$\begin{aligned} \log(136 \times 2^{256}) &\approx \log(136) + 256\log(2) \\ &\approx 79.197217798 \end{aligned}$$

とまず計算してから、エディントン数のおおよその大きさを

$$\begin{aligned} 10^{79.197217798} &= 10^{0.197217798+79} \\ &= 10^{0.197217798} \times 10^{79} \\ &\approx 1.57477241 \times 10^{79} \end{aligned}$$

と計算すれば良いのです。指数が 100 万までのクラス 2 の数であれば、対数を取れば 100 万までの数になりますので、対数が計算できる関数電卓であれば計算できます。

1.2 クラス 3 の巨大数

クラス 3 の数 (10^{10^6}–$10^{10^{10^6}}$): クラス 2 を超えると、10 進数で数を書いて大きさを比較するのが無理になります。ここでもクラス 2 までと同様、

[26] Arbitrary Precision Calculator http://apfloat.appspot.com/
[27] http://www.javascripter.net/math/calculators/100digitbigintcalculator.htm

クラス 2 の上限である 10^{10^6} を 10 の指数に持ってきて、$10^{10^{10^6}}$ をクラス 3 の上限とします。ここでクラスの定義をフォーマルな形で記しておきます。

【定義】自然数のクラス

数列 $c(n)$ を $c(0)=6, c(n+1)=10^{c(n)}$ で定義する。

クラス n の自然数とは、
(i) $n=0$ の時は、$c(0)$ 以下の自然数である。
(ii) $n>0$ の時は、$c(n-1)$ よりも大きく、$c(n)$ 以下の自然数である。

1.2.1　アルキメデスが考えた数の単位

古代ギリシャの数学者、科学者であるアルキメデス (Archimedes; 紀元前 287 年? – 紀元前 212 年) の著作である『砂粒を数えるもの』(The Sand Reckoner; Ψ α μ μ ΄ι τ η ς) [28] では、当時の宇宙に関する知識から、宇宙を埋め尽くすのに必要な砂粒の数を概算しました。

そのために、まずは大きな数をあらわすための数の体系を作りました。当時のギリシャでは、1 万をあらわす myriad (μ υ ρ ι ΄ας) という単語があり、この単語を使って 10^8 までの数を名付けることができました。

アルキメデスは、10^8 までの数を「第 1 級の数」として、10^8 を「第 2 級の数の単位」としました。そして、第 2 級の数の単位を使って、10^8 単位までの数を数えることにより、10^{16} までの数を 10^8 ごとに数えることができます。第 2 級の数の単位を使って 10^8 とあらわされる数、すなわち 10^{16} を第 3 級の数の単位とします。

このようにして、第 10^8 級の数の単位を使って 10^8 まで数えることで、$(10^8)^{(10^8)}=10^{8\cdot 10^8}$ まで数えることができます。ここまでの数を「第 1 期の数」として、次に第 2 期の数を考えます。第 1 期の最後の数 $10^{8\cdot 10^8}$ を「第 2 期第 1 級の数の単位」として、また 10^8 まで数えたときの最後の数を「第 2 期第 2 級の数の単位」とします。そして「第 2 期第 10^8 級の数」を使って 10^8

[28] http://ja.googology.wikia.com/wiki/砂粒を数えるもの

まで数えて、それを第 3 期第 1 級の数の単位とします。$\left(10^{8\cdot 10^8}\right)^2 = 10^{16\cdot 10^8}$ になります。

このようにして、第 10^8 期第 10^8 級の数の単位で 10^8 まで数えると、

$$\left((10^8)^{(10^8)}\right)^{(10^8)} = 10^{8\times 10^{16}}$$

まで数えることができます。これは、クラス 3 の巨大数です。紀元前にこのような実用からは程遠い大きさの数が考えられて、その記録が残っているというのは驚くべきことで「最古の巨大数」と言って良いと思います。

アルキメデスは、続いて宇宙を埋め尽くすのに必要な砂の数を次のように見積もりました。

1. ケシの実の直径は 40 分の 1 寸（直訳すると「指」なので、インチと同様に親指の幅をあらわす「寸」で訳しておく）よりも小さくはないので、直径 1 寸の球に入るケシの実の数は $40^3 = 64000$ 個よりは少ない。

2. ケシの実 1 粒の体積が砂 1 万粒よりは小さいので、直径 1 寸の球に入る砂の数は、64000 の 1 万倍よりは少ない。これは現在の数え方では 6 億 4000 万であるが、アルキメデスの単位で「第 2 級の数で 6 単位と第 1 級の数で 4000 万個」よりは小さいと計算した。そしてこれは、第 2 級の数で 10 単位 (10^9) より少ない。

3. 直径 100 寸の球に入る砂の数は、第 2 級の数で 10^7、つまり 10^{15} 個よりも少ない。

4. 1 スタディオンは 1 万寸よりも短い。スタディオンは古代ギリシアおよびローマで使われていた長さの単位で、複数形はスタディア。時代や地域によって差があるようだが、およそ 180 m 程度。

5. ここまでの仮定を整理すると、ケシの実の直径は 180 m の 1 万分の 1 の 40 分の 1、すなわち 0.45 mm よりも大きく、砂の直径はその $10000^{1/3}$ 分の 1、つまり 0.02 mm よりも大きい、という仮定をしてい

ることに相当する。ちなみに、前節でガンジス川の砂の数を見積もったときには、砂の直径を暫定的に 0.1 mm としたが、0.02 mm とすれば砂の数は $5^3 = 125$ 倍、すなわち 2 桁ほど大きく見積もられることとなる。

6. 直径 1 スタディオンの球に入る砂の数は、直径 100 寸の球に入る砂の数（第 2 級の数で 10^7 よりも少ない）の 100 万倍になる。これを計算すると第 3 級の数で 10 万個、つまり 10^{21} 個よりも少ないこととなる。

7. 太陽から地球までの距離を半径とする球を「世界」として、世界の直径を見積もる。地球の周囲は 300 万スタディアよりも小さく、直径は 100 万スタディアよりも小さい。そして、太陽の大きさの見積もりと太陽の直径が地球から見える視角から、世界の直径は地球の直径の 1 万倍よりも小さく、したがって 10^{10} スタディア（18 億 km）よりも小さいと見積もられる。現代の科学では、太陽から地球までの距離の 2 倍は約 3 億 km であることが知られているので、これは正しい。

8. 直径 10^{10} スタディアの球である「世界」に入る砂の数は、第 7 級の数で 1000 単位、つまり 10^{51} 個以下である。

9. 次に、古代ギリシャの天文学者で数学者のアリスタルコス (Aristarchus, 紀元前 310 年–紀元前 230 年頃) が考えた星が存在する「宇宙」の大きさを見積もる。ここで、地球の直径と世界の直径の比が、世界の直径と宇宙の直径の比に等しいという仮定を使う。すると宇宙の直径は世界の直径の 1 万倍よりは小さいこととなる。これは 2 光年よりも小さいという仮定であり、現代の天文学からすると小さすぎる仮定となる。この宇宙に入る砂の数は、第 8 級の数で 1000 万、すなわち 10^{63} 個以下である。

Harrison (2000) [29] は、10^{63} 個の砂に含まれる核子の数はおよそ 10^{80} 個であり、エディントン数と一致するのは驚くべき偶然だとしています。

[29] Harrison, E. R. (2000) Cosmology - The Science of the Universe. Cambridge University Press. pp. 481–482.

1.2.2 不可説不可説転

ここでまた、仏教で使われている数の大きさについての話になります。インドの数の単位では、本来は下数の単位が使われていましたが、後の仏典では上数として使われることがあり、華厳経第 45 巻、阿僧祇品第三十には、

> 100 洛叉（らくしゃ=10 万）を 1 倶胝（くてい）とする。倶胝倶胝を 1 阿庾多（あゆた）とする。阿庾多阿庾多を 1 那由他（なゆた）とする。那由他那由他を 1 頻波羅（びんばら） とする。（中略）不可説転不可説転を 1 不可説不可説とする。このまた不可説不可説（倍）を 1 不可説不可説転（ふかせつふかせつてん）とする。

と書かれています。「倶胝倶胝を 1 阿庾多とす」とは、倶胝×倶胝=阿庾多ということなので、10^7 を意味する倶胝以上は、中国の上数と同じように 2 乗すると次の単位になっています。したがって、倶胝（くてい）$=10^7$、阿庾多（あゆた）$=10^{7\times 2} = 10^{14}$、那由他（なゆた）$=10^{7^{\times 2^2}} = 10^{28}$、頻波羅（びんばら）$=10^{7^{\times 2^3}} = 10^{56}$ と、2 乗することで指数が 2 倍になり、指数の中の 2 の指数が 1 増えます。つまり、倶胝の n 個上の単位は $10^{7^{\times 2^n}}$ となります。不可説不可説転[30]（ふかせつふかせつてん）は、倶胝の 122 個上の単位なので、$10^{7\times 2^{122}}$ になります。

【定義】不可説不可説転

不可説不可説転 $= 10^{7\times 2^{122}}$

数学史家の鈴木真治によると[31]、華厳経の漢訳完本としては、全 60 巻の旧訳（普経）と、全 80 巻の新訳（唐経）があり、上記の定義は旧訳を元にしていて、華厳経の旧訳と新訳で数が微妙に異なっているとしています。数学者の末綱は、新訳を元に「不可説転転 $= 10^{5\times 2^{120}}$」と計算しています[32]。トーマス・クリアリー (Thomas Cleary) が華厳経の新訳を英語に訳し[33]、

[30] http://ja.googology.wikia.com/wiki/不可説不可説転
[31] 鈴木真治 (2016)『巨大数』岩波書店
[32] 末綱恕一 (1957)『華厳経の世界』春秋社
[33] Cleary, T. (1993) The flower ornament scripture: a translation of the Avatamsaka Sutra. Shambhala, Colorado, USA.

その訳では unspeakable $= 10^{10\times 2^{120}}$, square untold $= 10^{10\times 2^{123}}$ となります[34]。

1.2.3　グーゴルプレックス

グーゴルプレックス[35] (googolplex) は数の単位で、10 のグーゴル乗 (10^{googol})、つまり $10^{10^{100}}$ です。

【定義】グーゴルプレックス

グーゴルプレックス $= 10^{10^{100}}$

グーゴルの説明で引用したように、グーゴルプレックスはカスナーの甥のミルトン・シロッタが 9 歳の時に名付けましたが、それは 1 の後に 0 を疲れるまで書いた数、という定義でした。カスナーは、人が疲れる時間には違いがあるので、それでは正確な定義にはならないとして、より正確な定義として 1 の後に 1 グーゴル個の 0 を書いた数としました。1 グーゴル個の 0 を書くためには、宇宙の全物質をインクに変えてもまだ足りません。観測可能な宇宙に存在する素粒子の数は 10^{80} から 10^{85} 程度なので、素粒子 1 つが「0」を表すものとして、全宇宙の素粒子を集めてもまだグーゴルプレックスを 10 進数で表記することはできないからです。巨大数のたとえとして「全宇宙の物質全てをインクに変えても 10 進数で表記できないほど大きい」というたとえがよく使われますが、このたとえはグーゴルプレックスのようなクラス 3 の数をたとえるときには良いのですが、クラス 4 以上の数をたとえるときには、たとえが小さすぎて不適切であると言えます。

クラス 3 の数は、通常の計算で比較するのが困難です。不可説不可説転まで大きくなると、指数表記をしたときに指数の部分が 7×2^{122} という大きな数になりますので、通常の電卓では計算範囲を超えます。たとえば、前節で紹介した高精度計算サイトを使って計算しても、計算結果は∞と表示されるだけで、計算の上限を超えてしまっています。

[34] http://iteror.org/big/book/ch1/ch1_7.html
[35] http://ja.googology.wikia.com/wiki/グーゴルプレックス

このような大きさの数を比較するためには、対数を使うことができます。不可説不可説転とグーゴルプレックスの比較であれば、すでに両者とも 10^x の形になっているので、簡単に比較できます。両者の常用対数を取った、7×2^{122} と 10^{100} を比較すれば良いのです。

$$7 \times 2^{122} = 372183838819776444413065976878496481 28 \approx 3.72 \times 10^{37}$$

と計算されるので、これよりも 10^{100} の方が大きく、グーゴルプレックスは不可説不可説転よりも大きいことが分かります。

1.2.4 ハイパー E 表記

指数が重なると読みにくくなってくるので、アメリカのアマチュア数学者・哲学者であるスビス・サイビアン (Sbiis Saibian) [36] が考案したハイパー **E** 表記[37] (hyper-E notation) を使うと便利です。

ハイパー E 表記は、1 つ以上の正の整数から成る数列 a_n の引数をハイペリオン記号 # で区切ったものです。これを $E(b)a_1\#a_2\#...\#a_n$ と表記し、b を基数と呼びます。基数が省略されたときは 10 がデフォルトで、よく省略されます。次のような定義です。

【定義】ハイパー E 表記

$$\begin{aligned} E(b)x &= b^x \\ E(b)a_1\#a_2\cdots\#a_n\#1 &= E(b)a_1\#a_2\cdots\#a_n \\ E(b)a_1\#a_2\cdots\#a_n &= E(b)a_1\cdots\#a_{n-2}\#(E(b)a_1\cdots\#a_n - 1) \end{aligned}$$

1. 1 番目の式は、ハイペリオン記号がまったくない場合には b^x となるということです。

2. 2 番目の式は、最後のハイペリオン記号の後の数が 1 であれば、そのハイペリオン記号と数を取り除くことができるということです。

[36] http://ja.googology.wikia.com/wiki/Sbiis_Saibian
[37] http://ja.googology.wikia.com/wiki/ハイパー E 表記

3. 3 番目の式は、それ以外の場合です。まず、元の式の最後の数を 1 減らしたものを z とします。次に、元の式から最後の 2 つの数とその間のハイペリオン記号を消して、かわりに (z) を追加します。

具体的に計算すると、次のようになります。

$$\begin{aligned} Ea &= Ea\#1 = 10^a \\ E100 &= E100\#1 = 10^{100} = \text{グーゴル} \\ E100\#2 &= E(E100) = E10^{100} = 10^{10^{100}} = \text{グーゴルプレックス} \\ E100\#3 &= E(E100\#2) = 10^{10^{10^{100}}} \end{aligned}$$

つまり、$Ea\#b$ は 10 の指数を b 個重ねたものに、さらに a を重ねたものになります。

ハイパー E 表記は、さらに E100#100#100#100 のように重ねることで大きくなり、そこから拡張ハイパー E 表記、連鎖 E 表記、拡張連鎖 E 表記といったさらに大きな数を表記するシステムへと発展します。その詳細については本書では踏み込みませんが、サイビアンの巨大数論に関する Web 書籍 「1 から無限大まで – 有限数ガイド」 (One to Infinity: A Guide to the Finite) [38] に、詳しく解説されています。

1.2.5　プロマキシマ

サイビアンの紹介をしたところで、サイビアンがプロマキシマ[39] (promaxima) と名づけた巨大数を紹介します。これはネット上で次のような投稿がされたところから、考えられた数です。

> 宇宙のすべての空間が原子や素粒子のようなもので完全に充填されていたとします。その粒子の数はとても大きいでしょう。さらに、これらの粒子が充填される順列を数え上げるといくつになるでしょうか。N 個の粒子があるとすれば、N の階乗、すなわち $N!$ 通りになるでしょう。これはどのくらい大きな数になるでしょうか。

[38] One to Infinity https://sites.google.com/site/largenumbers/
[39] http://ja.googology.wikia.com/wiki/宇宙論の巨大数

これに対して、サイビアンは観測可能な宇宙が宇宙で最も小さい物質である「宇宙ひも」で満たされたと考えました。宇宙ひもの長さをプランク長すなわち約 10^{-35}m として、観測可能な宇宙の直径を 10^{26}m としました。すなわち、宇宙ひもで観測可能な宇宙を充填すると $(10^{26}/10^{-35})^3 = 10^{183}$ 個の宇宙ひもで充填されることになります。これを N とします。この宇宙ひもの配列は $N! = 10^{183}!$ 通りであり、スターリングの近似を使うと約 $10^{10^{185}} = E185\#2$ 通りとなります。

サイビアンは、この計算にさらに時間の概念を加えました。すなわち、時間の最小単位はプランク時間すなわち約 10^{-43} 秒なので、1 プランク時間における組み合わせが $N!$ 通りであると考えて、t プランク時間における組み合わせを $(N!)^t$ 通りとしました。宇宙が 5×10^{17} 年続くと考えてそれをプランク時間に直して t に入れることで、$(N!)^t = 10^{10^{245}} = E245\#2$ 通りとなります。これをサイビアンはプロマキシマと名づけました。プロマキシマは「確率の最大 (probability maximum)」から名づけたもので、宇宙の歴史として考えられる組み合わせの上限値であり、サイビアンは現実的な数の上限であるとしました。

ここで、サイビアンはプロマキシマは科学的な数であり、$10^{10^{245}}$ はその上限を計算したものであって、物理法則に反する単純化した仮定を使って計算したものであり、実際にはプロマキシマは $10^{10^{245}}$ よりもずっと小さくなるはずであるとしました。たとえば、物質の移動速度は光速を超えないことから、プランク時間で移動できる距離はプランク長を超えません。そのことから、ある時間の次の時間における宇宙ひもの位置は制限されます。また宇宙は宇宙ひもでみたされているわけではありません。しかし、サイビアンはこの数は現実的な数の上限としての意味を持っているとしています。

サイビアンはその後、陽子の半減期が約 10^{35} 年であり、宇宙には約 10^{81} 個の陽子が存在することから、宇宙の寿命は 10^{35} 年はあるとして、さらに宇宙の膨張速度を計算に入れた上で、$10^{6.5446\times10^{343}}$ という数を導き、可能な平行宇宙の数であるとしました。

1.2.6　超過剰数

過剰数[40] (abundant number) は、正の約数の総和が元の数の 2 倍よりも大きい数です。すなわちその数を除く正の約数の総和が元の数よりも大きい数です。たとえば、36 は約数である 1, 2, 3, 4, 6, 9, 12, 18 の和が 55 となり、36 よりも大きいため過剰数です。その逆にその数を除く約数の和が元の数よりも小さい数を不足数と言い、両者が一致する数を完全数と言います。最初の過剰数は 12, 18, 20, 24, 30, 36, 40, 42, 48, ... で、ほとんどの過剰数は偶数となるように見えます。最小の奇数の過剰数は 945 です。2 と 3 で割れない最小の過剰数は

$$5391411025 = 5^2 \times 7 \times 11 \times 13 \times 17 \times 19 \times 23 \times 29$$

で、11 以下の数で割れない最小の過剰数は
$13^2 \times 17^2 \times 19 \times 23 \times 29 \times 31 \times 37 \times 41 \times 43 \times 47 \times 53 \times 59 \times 61 \times 67 \times 71 \times 73 \times 79 \times 83 \times 89 \times 97 \times 101 \times 103 \times 107 \times 109 \times 113 \times 127 \times 131 \times 137 \times 139 \times 149 \times 151 \times 157 \times 163 \times 167 \times 173 \times 179 \times 181 \times 191 \times 193 \times 197 \times 199 \times 211 \times 223 \times 227 \approx 7.970466327 \times 10^{87}$
です。

ここまではクラス 2 の数ですが、ここからクラス 3 の数に入っていきます。そのために「とても過剰な数」について考えてみます。n の正の約数の総和を $\sigma(n)$ とすると、過剰数は $\sigma(n) > 2n$ となる数です。では $\sigma(n) > 1000n$ となる「1000 倍以上過剰な数」は存在するかどうかを考えてみましょう。$N = \mathrm{ceil}(e^{1000})!$ は、そのような数となります。ここで、ceil は天井関数で、$\mathrm{ceil}(x)$ は x 以上の最小の整数です。そして、! は階乗です。N は $\mathrm{ceil}(e^{1000})$ 以下のすべての正の整数で割り切れるため、N は $1 \leq k \leq \mathrm{ceil}(e^{1000})$ のすべての k に対して、$\frac{N}{k}$ で割り切れます。したがって、

$$\sigma(N) > N \sum_{k=1}^{\mathrm{ceil}(e^{1000})} \frac{1}{k} > N \log(\mathrm{ceil}(e^{1000})) > N \log(e^{1000}) = 1000N$$

となり (log は自然対数)、N は 1000 倍以上過剰な数です。$N < 10^{10^{446.94}} = E446.94\#2$ です。

[40] 巨大数研究 Wiki - 過剰数 http://ja.googology.wikia.com/wiki/過剰数

次に、$\sigma(n) > 1000n$ を満たすような最小の数はどの程度の大きさでしょうか。1984 年に、Guy Robin はリーマン予想が正しいとしたときに、$n > 5040$ に対して

$$\sigma(n) < e^{\gamma} n \log\log n$$

が成り立つことを示しました[41] 。ここで、γ はオイラーの定数で、およそ 0.5772156649 です。このことを使うと、$\sigma(n) > 1000n$ であれば

$$\begin{aligned} 1000n &< \sigma(n) < e^{\gamma} n \log\log n \\ \frac{1000}{e^{\gamma}} &< \log\log n \\ n &> e^{e^{1000/e^{\gamma}}} > 10^{10^{243.47}} = E243.47\#2 \end{aligned}$$

となります。これで、$\sigma(n) > 1000n$ を満たす最小の数の上限と下限が

$$E243.47\#2 < n < E446.94\#2$$

と、グーゴルプレックスよりは大きいクラス 3 の数であることが分かりました。

$m < n$ であるすべての自然数 m に対して $\frac{\sigma(m)}{m} < \frac{\sigma(n)}{n}$ を満たす自然数 n を超過剰数 (superabundant number) と言います。$\sigma(n) > 1000n$ を満たす最小の n は超過剰数です。超過剰数は無限にあり、1, 2, 4, 6, 12, 24, 36, 48, 60, 120, 180, 240, 360, 720, 840, 1260, 1680, 2520, 5040, 10080, 15120, 25200, 27720, 55440, 110880, 166320, 277200, 332640, 554400, 665280, 720720, 1441440, 2162160, 3603600, 4324320, 7207200, 8648640, 10810800, 21621600, ... と続きます[42]。

1.3 クラス 4 の巨大数

クラス 4 の数 ($E6\#3$–$E6\#4$): クラスの定義より、クラス 4 の上限は $E6\#4 = 10^{10^{10^{10^6}}}$ です。このクラスになると、大小を比較するには、対数を 2〜3 回取る必要があります。また、前節で「全宇宙の物質全てをインク

[41] Robin, G. (1984) Grandes valeurs de la fonction somme des diviseurs et hypothèse de Riemann. *Journal de Mathématiques Pures et Appliquées* 63, 187–213.

[42] https://oeis.org/A004394

に変えても 10 進数で表記できないほど大きい」というようなたとえはこのクラスの数をたとえるには小さすぎて不適切である、という話を書きました。実際には、一般の人が理解できるたとえとしては、「全宇宙の物質全てをインクに変えても 10 進数で表記できないほど大きい」あたりが限界だろうと思います。したがってここから先はまさに「たとえようもないほど大きな数」を扱うことになります。

1.3.1　グーゴルデュプレックス

10 の 100 乗がグーゴルで、10 のグーゴル乗はグーゴルプレックスです。それでは、10 のグーゴルプレックス乗はなんというのでしょうか。これには、いろいろな名前がついています。最も単純な名前のつけ方は、10 の N 乗が N プレックスであると考えて、**グーゴルプレックスプレックス** (googolplexplex) とする方法です。他にも、同じ数をあらわす言葉としてグーゴルプラスプレックス (googolplusplex)、ガーグーゴルプレックス (gargoogolplex)、グーゴルプレクシアン (googolplexian) 等があります。

サイビアンは、グーゴルプレックスプレックスのかわりに**グーゴルデュプレックス** (googolduplex) と名づけました。つまり、プレックスの前に 2 をあらわす du の接頭辞を加えました。そして、10 のグーゴルデュプレックス乗は 3 をあらわす接頭辞 tri を使って**グーゴルトリプレックス** (googoltriplex) となり、さらに以下のように続きます。

$$
\begin{aligned}
E100 &= 10^{100} = \text{グーゴル (googol)} \\
E100\#2 &= 10^{10^{100}} = \text{グーゴルプレックス (googolplex)} \\
E100\#3 &= 10^{10^{10^{100}}} = \text{グーゴルデュプレックス (googolduplex)} \\
E100\#4 &= \text{グーゴルトリプレックス (googoltriplex)} \\
E100\#5 &= \text{グーゴルクアドリプレックス (googolquadriplex)} \\
E100\#6 &= \text{グーゴルクインプレックス (googolquinplex)} \\
E100\#7 &= \text{グーゴルセクスティプレックス (googolsextiplex)} \\
E100\#8 &= \text{グーゴルセプティプレックス (googolseptiplex)} \\
E100\#9 &= \text{グーゴルオクティプレックス (googoloctiplex)}
\end{aligned}
$$

$E100\#10$ = グーゴルノニプレックス (googolnoniplex)

$E100\#11$ = グーゴルデシプレックス (googoldeciplex)

グーゴルトリプレックス以降はクラス 5 以上の巨大数となりますが、ここでまとめて紹介しておきました。

1.3.2 スキューズ数

それでは次の巨大数の話をするために、少し準備をします。素数計数関数 (prime counting function) $\pi(x)$ は、x 以下の素数の個数を返す関数です。対数積分 $\mathrm{li}(x)$ は、全ての正の実数 $x \neq 1$ において、

$$\mathrm{li}(x) = \int_0^x \frac{dt}{\log(t)}$$

で定義される関数です (log は自然対数)。素数定理によれば $\pi(x)$ は漸近的に $\mathrm{li}(x)$ に等しくなりますが、計算がなされている範囲では、常に $\pi(x) < \mathrm{li}(x)$ となっていて、今でも $\pi(x) > \mathrm{li}(x)$ となるような具体的な x は見つかっていません。ところが、$\pi(x)$ と $\mathrm{li}(x)$ は無限回大小関係が逆転することが証明されているため、いつはじめて $\pi(x) > \mathrm{li}(x)$ となる x が出てくるのかが研究されてきました。

スキューズ数[43] [44] (Skewes number) には 2 つあります。スキューズは、1933 年の論文で、リーマン予想が正しいとしたときに、$Sk_1 = e^{e^{e^{79}}} = E(e)79\#3$ 以下の数に $\pi(x) > \mathrm{li}(x)$ となる数が存在することを証明しました[45]。この Sk_1 が第 **1** スキューズ数と呼ばれます。さらにスキューズは、1995 年に、リーマン予想を仮定することなしに、第 **2** スキューズ数 $Sk_2 = e^{e^{e^{e^{7.705}}}} = E(e)7.705\#4$ 以下の数で $\pi(x) > \mathrm{li}(x)$ となる x が存在することを証明しました[46]。この上限は、Bays and Hudson (2000) [47] によっ

[43] 巨大数研究 Wiki - スキューズ数 http://ja.googology.wikia.com/wiki/スキューズ数

[44] MathWorld - Skewes Number http://mathworld.wolfram.com/SkewesNumber.html

[45] Skewes, S. (1933) On the difference pi(x)-li(x). *Journal of the London Mathemetical Society* s1-8(4): 277–283. doi:10.1112/jlms/s1-8.4.277

[46] Skewes, S. (1955) On the difference pi(x)-li(x). II. *Proceedings of the London Mathematical Society* s3-5 (1): 48–70. doi:10.1112/plms/s3-5.1.48

[47] Bays, C. and Hudson, R. H. (2000), A new bound for the smallest x with $\pi(x) > \mathrm{li}(x)$. *Mathematics of Computation* 69 (231): 1285–1296. doi:10.1090/S0025-5718-99-01104-7

て約 1.3983×10^{316} にまで下げられています。

スキューズ数はいずれもクラス 4 の数です。Sk_1 は $E34\#3 = 10^{10^{10^{34}}}$ 程度です。Sk_2 は $E963\#3 = 10^{10^{10^{963}}}$ 程度ですが、分かりやすく $E3\#4 = 10^{10^{10^{10^3}}}$ とされることもあります。

【定義】第 2 スキューズ数

第 2 スキューズ数 $= e^{e^{e^{e^{7.705}}}} \approx 10^{10^{10^{10^3}}} = E3\#4$

ムナフォは、通常の計算プログラムでは大きすぎて計算できないような巨大数を計算するためのプログラム、ハイパーカルク [48] (Hypercalc) を作成しました。Hypercalc を使うと、第 2 スキューズ数は

入力: `e^e^e^e^7.705`

結果: `10 ^ [ 10 ^ ( 3.299943221955 x 10 ^ 963 ) ]`

と、計算ができます。第 2 スキューズ数をこのように直接計算できるプログラムは、他にはなかなかないだろうと思います。

1.4　クラス 5 以上の巨大数

1.4.1　宇宙論で使われた最大の数

宇宙論で使われた最大の巨大数（であると思われる数）に、クラス 5 の巨大数である

宇宙論で使われた最大の数

`10^10^10^10^10^1.1 = E1.1#5`

という数があります[49] [50]。これは、理論物理学者のドン・ページ (Don Nelson Page) が[51] 「リンデの確率過程的インフレーションという説で生まれるか

[48] Hypercalc http://www.mrob.com/pub/perl/hypercalc.html
[49] http://ja.googology.wikia.com/wiki/宇宙論の巨大数
[50] Longest Possible Time http://www.numberphile.com/videos/longest_time.html
[51] Page, D. N. (1994) Information loss in black holes and/or conscious beings? http://arxiv.org/pdf/hep-th/9411193

もしれないあらゆる多重宇宙の全質量を 1 個のブラックホールに潰して適当な箱の中に入れた」と仮定したときにブラックホールの蒸発後にまたブラックホールができるまで[52] のポアンカレ再帰時間（力学系のある状態を出発点としたときに、初期状態にとても近い状態に戻るまでの時間）を計算したものです。この数は現在の状態の宇宙に対するポアンカレ再帰時間ではないので、「宇宙の全ての物質が元に戻る」という説明は不正確です。意味のある数というよりは、どちらかというと、巨大数を作ることを目的に計算された数、ということのようです。単位はプランク時間 (5.391×10^{-44} 秒) でも 1000 年でもなんでもいい、と Page (1994) に書かれているように、クラス 5 の数になると、クラス 4 までの数をかけても割っても右肩の数（この場合だと 1.1）にほとんど影響を与えなくなってしまうため、時間の単位を気にする必要すらなくなってしまいます。このことは、次章以降で色々と計算をしていきます。

1.4.2 ベントレー数

「現代巨大数論の父」と言われるジョナサン・バウアーズ (Jonathan Bowers) が書いた「永遠の努力」[53] (Forever endeavor) という短編小説は、このレベルの数の大きさの果てしなさについて想像をかきたてます。ここに巨大数研究 Wiki に掲載した概略のストーリーを記します。英語の原文[54] は、もっと詳細に描写されていて面白いです。

> この物語は、10 個のカウンターを作れば 100 万ドルを提供すると宣伝する、奇妙なカードが通りを舞っているのに気が付いた、欲張りなベントレーから始まります。彼はカウンターとは何なのかも知らないままこれを引き受け、1 枚の壁に窓のある窮屈な部屋に通されました。彼が 10 個のカウンターを作るのを手伝う、コンパドリーという名前のアシスタントを除き、彼はその部屋にひとりでいなければなりません。一度カウンターを作り始めてしまうとその部屋から逃げることは許されません。

[52] https://twitter.com/astrophys_tan/status/391928927012134912
[53] 巨大数研究 Wiki - 永遠の努力 http://ja.googology.wikia.com/wiki/永遠の努力
[54] `http://www.polytope.net/hedrondude/foreverendeavor.htm`

部屋には金属のディスクでいっぱいになった大きな段ボール箱が、またその箱の横には 10 本の金属の棒がありました。コンパドリーは棒を取りそれにディスクを取り付けました。そのディスクは 10 の節に分割され、それぞれ 0〜9 の番号が付けられていました。それがカウンターでした。

すでに棒に 1 つのディスクが取り付けられている 1 つ目のカウンターは完成して、コンパドリーがそれを持っていました。2 つ目のカウンター作りから始めることになりました。ベントレーは、棒を手に取ってディスクを取り付け、そしてそれに続いて、コンパドリーは 1 を示すようにディスクを V 字の刻み目 1 つ分回転させます。ベントレーはその作業を繰り返し、コンパドリーは 0 になるまでその数を 1 つずつ増やすのです。ベントレーが 10 枚のディスクを取り付け終わった所で、コンパドリーのカウンターが 0 となって、2 つ目のカウンターが完成しました。

それからコンパドリーは、ベントレーの 2 個目のカウンターを受け取り、それぞれのディスクを分割された桁として使って、カウンターを 0000000000 にセットします。そして、ベントレーは別の棒を取り、3 個目のカウンター作成を開始します。棒にディスクを 1 つ取り付けるごとに、コンパドリーはカウンターを 0000000001, 0000000002, 0000000003, ... と、1 ずつ増やします。箱の中のディスクは、いくら使っても減ることはなく常にいっぱいで、棒は窓の外に向かっていくらでも延びます。

ベントレーとコンパドリーは、睡眠と食事、トイレ以外は休まずに、休日もなく 1 日 18 時間ひたすらこの作業を続けます。食事はいつも、投下口から送られてくるボローニャサンドウィッチと堅く乾燥したパンケーキで、メニューは変わりません。コンパドリーは一言も口を利かず、完全に感情も無く、ベントレーが何を言ってもほとんどなんの関心も示しませんでした。奇妙なことに、ベントレーもコンパドリーも、老いていくようには見えませんでした。しかしながら、ベントレーは 500 年以上もカウンターに取り組んでいるらしいのです。

550 年以上かかって、100 億枚のディスクを取り付け、3 個目のカウンターが完成しました。棒の長さは 10 万キロになります。そしてコンパドリーはその 3 個目のカウンターを受け取り、$\underbrace{000\ldots000}_{100\text{ 億個}}$ にセットしました。ベントレーは別の棒を取って、4 個目のカウンター作りを始め、コンパドリーはその 3 つ目のカウンターを数え始めました。4 個目のカウンターは、ディスクを 10 の 100 億乗個取り付けることで完成します。それは、グーゴルのグーゴル倍の、そのまたグーゴル倍の…と、1 億回繰り返したほどの、気の遠くなるほど大きな数です。とてもその作業が終わる日が来るようには思えません。そして 4 個目のカウンターが完成すれば、今度はそのカウンターを使って、5 個目のカウンター作りを始めることになります。つまり今度は $10^{10^{10000000000}}$ 個のディスクを取り付けることになります。

果たして 10 個目のカウンターが完成する時はやってくるのでしょうか？ とてもやって来るとは思えません。この「永遠の努力」を終わらせる方法は知られていません。

ベントレーが 10 個のカウンターを完成させるまでに、とりつけなければならないディスクの数をベントレー数[55] (Bentley's Number) と言います。

【定義】ベントレー数

$$1+10+\sum_{i=1}^{8} E10\#i = 1+10+10^{10}+10^{10^{10}}+...+10^{10^{10^{10^{10^{10^{10^{10}}}}}}}$$

ベントレー数はクラス 9 の巨大数です。私たちの想像力をもってすれば、これはもうほとんど「無限」と思える数です。

[55]巨大数研究 Wiki - ベントレー数 http://ja.googology.wikia.com/wiki/ベントレー数

1.5　巨大数に関する参考資料

本書では、個別の話題に関する参考資料を脚注に記していますが、本節で巨大数論全般に関する参考資料をいくつか紹介します。

小林銅蟲による『寿司 虚空編』[56] という巨大数漫画は、多くの人が巨大数を知るきっかけとなりました。 小林銅蟲は「巨大数探索スレッド」[57]の議論をまとめた「巨大数研究室」[58] を管理しています。また、巨大数に関する発表をしたときのスライド[59]がいくつか公開されています。『寿司 虚空編』は、当初小学館の「裏サンデー」に掲載されて話題となりましたが、その後 pixiv コミックで無料のウェブコミックとして継続され、2017 年 8 月 10 日に三才ブックスから単行本が発売されました。

数学史家の鈴木真治による『巨大数』[60]という書籍は、20 世紀までの巨大数に関する数学史を一般向けに分かりやすくまとめたものです。

英語のサイトでは、すでに紹介したムナフォとサイビアンのサイトが有名です。 そして、巨大数に関する情報を集めている Googology Wiki [61] というサイトでは、世界の巨大数ファンが日夜巨大数の情報を収集整理し、議論しています。ここは巨大数コミュニティの本拠地と言って良いでしょう。この Googology Wiki の日本語版「巨大数研究 Wiki」[62] では、日本語で巨大数の情報が整理されています。本書では、脚注に巨大数研究 Wiki の該当項目を参照しているところが多くあります。Googology Wiki は、英語と日本語の他にもオランダ語、ドイツ語、フランス語、中国語、ロシア語版、チェコ語版、マレー語版があり、それぞれの言語間で項目ごとにリンクされています。

[56] pixiv コミック『寿司 虚空編』 http://comic.pixiv.net/works/1505
[57] http://ja.googology.wikia.com/wiki/巨大数探索スレッド
[58] 巨大数研究室 http://www.geocities.co.jp/Technopolis/9946/
[59] http://www.slideshare.net/DoomKobayashi/presentations
[60] 鈴木真治 (2016) 『巨大数』岩波書店
[61] Googology Wiki http://googology.wikia.com/
[62] 巨大数研究 Wiki http://ja.googology.wikia.com/

第 2 章
原始再帰関数

前章ではクラスが上がると気の遠くなるような大きさの数となることを見てきましたが、ここからは、クラスという概念を使っても間に合わないほどの大きな数を表記するときに便利な矢印表記について見ていきます。矢印表記で書ける程度の大きさの巨大数は、原始再帰関数で定義されている巨大数だ、ということもできます。つまり本章では原始再帰関数レベルの巨大数を取り扱うことになります。まずは矢印表記についての説明から入って、最後に原始再帰とは何か、という話をします。

2.1 クヌースの矢印表記

2.1.1 矢印表記

アメリカの数学者、計算機科学者で、コンピュータによる組版システム TEX の開発者としても有名なドナルド・クヌース (Donald Knuth) が考案したクヌースの矢印表記[1] [2] [3] (arrow notation) について説明します。ちなみに本書は TEX の上に構築された文書処理システム LATEX で作成しています。

矢印表記は以下で定義されます。

[1]巨大数研究 Wiki - 矢印表記 http://ja.googology.wikia.com/wiki/矢印表記
[2]MathWorld - Arrow Notation http://mathworld.wolfram.com/ArrowNotation.html
[3]Knuth, D. E. (1976) Mathematics and Computer Science: Coping with Finiteness. *Science* 194, 1235–1242. doi:10.1126/science.194.4271.1235

【定義】クヌースの矢印表記

$$
\begin{aligned}
x \uparrow y &= x^y \\
x \uparrow\uparrow 2 &= x \uparrow x, x \uparrow\uparrow y = x \uparrow (x \uparrow\uparrow (y-1)) \\
x \uparrow\uparrow\uparrow 2 &= x \uparrow\uparrow x, x \uparrow\uparrow\uparrow y = x \uparrow\uparrow (x \uparrow\uparrow\uparrow (y-1)) \\
x \uparrow^n y &= x(n \text{ 個の} \uparrow) y = x \uparrow^{n-1} (x \uparrow^n (y-1))
\end{aligned}
$$

指数を表記する a^b という方法は、b を上付き文字で書く必要がありますが、プログラミングをするときやテキストでメールを送信するときなど、上付き文字を使えない環境があります。そういうときに $a \uparrow b$ という書き方がされていました。↑の文字が使えないときには、かわりに ^ の記号が使われていました。a^b という表記は拡張しにくいため、クヌースは $a \uparrow b$ の表記を拡張して矢印表記を考案したものと考えられます。同様に↑の文字が使えないときには、かわりに ^ を使って、$a \uparrow\uparrow b$ を $a \hat{}\, \hat{}\, b$ のように書くことができます。

2.1.2　指数

具体的に計算をしてみましょう。まずは↑が 1 つの場合です。これは指数と同じです。↑を連結させると、指数と同じように右から順番に計算します。Hypercalc を使うと、次のように計算されます。

$$
\begin{aligned}
3 \uparrow 3 &= 3^3 = 27 \\
3 \uparrow 3 \uparrow 3 &= 3^{3^3} = 3^{27} = 7625597484987 \\
3 \uparrow 3 \uparrow 3 \uparrow 3 &= 3^{3^{3^3}} = 1.35 \times 10 \uparrow 3638334640024 \\
3 \uparrow 3 \uparrow 3 \uparrow 3 \uparrow 3 &= 10 \uparrow (6.46 \times 10 \uparrow 3638334640023) \\
3 \uparrow 3 \uparrow 3 \uparrow 3 \uparrow 3 \uparrow 3 &= 10 \uparrow 10 \uparrow (6.46 \times 10 \uparrow 3638334640023)
\end{aligned}
$$

ここで Hypercalc では、$10 \uparrow 10 \uparrow 10 \uparrow 10 \uparrow (6.46 \times 10 \uparrow 3638334640023)$ という計算結果は、「4 PT (6.46 × 10 ^ 3638334640023)」と表示されま

す。最初の 10 ↑を 4 回繰り返す部分を 4 PT と表示することで、オーバーフローしないようにしています。

さてここで右辺を見ると、最後のかっこの中が、途中からずっと同じ数 $6.46 \times 10 \uparrow 3638334640023 = 10 \uparrow 10 \uparrow 12.56$ になっています。つまり 3 ↑を 1 つつけることと、10 ↑をつけることが、途中からまったく変わらなくなってしまっています。本当は変わるのですが、表記されている桁数内では変わらない、ということです。ここで上のピラミッドを見やすく書き直します。

$$
\begin{aligned}
3 \uparrow 3 &= 27 \\
3 \uparrow 3 \uparrow 3 &= 10 \uparrow 12.88 \\
3 \uparrow 3 \uparrow 3 \uparrow 3 &= 10 \uparrow 10 \uparrow 12.56 \\
3 \uparrow 3 \uparrow 3 \uparrow 3 \uparrow 3 &= 10 \uparrow 10 \uparrow 10 \uparrow 12.56 \\
3 \uparrow 3 \uparrow 3 \uparrow 3 \uparrow 3 \uparrow 3 &= 10 \uparrow 10 \uparrow 10 \uparrow 10 \uparrow 12.56 \\
3 \uparrow 3 \uparrow 3 \uparrow 3 \uparrow 3 \uparrow 3 \uparrow 3 &= 10 \uparrow 10 \uparrow 10 \uparrow 10 \uparrow 10 \uparrow 12.56
\end{aligned}
$$

左では 3 ↑を増やして、右では 10 ↑を増やしています。一番右の数は 27,12.88,12.56 と小さくなり、それ以上は小さくなりません。一番右の数を少しだけ増やす効果は、このピラミッドで下に行けばいくほど爆発的に大きくなります。それに対して 3 ↑を 10 ↑に変える効果というのはたいした効果がないので、ほとんど無視できてしまいます。

したがって指数を重ねることによって巨大数を作るためには、使う数は 10 ではなくて 3 で十分だ、ということになります。この段階では 2 でもいいのですが、2 にしてしまうと、この先の計算で $2 \uparrow\uparrow\uparrow\uparrow 2 = 2 \uparrow\uparrow\uparrow 2 = 2 \uparrow\uparrow 2 = 2 \uparrow 2 = 4$ となって面白くないので、3 を使っています。

そのようなわけで $3 \uparrow 3 \uparrow 3$ はクラス 2 の数ですが、$3 \uparrow 3 \uparrow 3 \uparrow 3$ はクラス 3、$3 \uparrow 3 \uparrow 3 \uparrow 3 \uparrow 3$ はクラス 4、と 1 つ↑が増えるごとにクラスが 1 つ上がります。$3 \uparrow 3 \uparrow 3 \uparrow 3 \uparrow 3 \uparrow 3$ はクラス 5 なので、第 2 スキューズ数を超えます。

なぜベースを 3 にしても 10 にしても、増え方が変わらないのでしょうか。ある数 x に対して、3^x と 10^x でどれだけの違いがあるかを考えると、

$$10^x = (3^{\log_3(10)})^x = 3^{x\log_3(10)} \approx 3^{2.1x}$$

なので、x が 2.1 倍程度増える効果になります。次に 10^{10^x} と 3^{3^x} でどの程度の違いになるかというと、

$$\begin{aligned} \log_3(10^{10^x}) &\approx 2.1 \times 10^x \approx 2.1 \times 3^{2.1x} \\ 10^{10^x} &\approx 3^{2.1\times 3^{2.1x}} = 3^{3^{2.1x+\log_3(2.1)}} \approx 3^{3^{2.1x+0.68}} \end{aligned}$$

となり、x を 2.1 倍して 0.68 を足す効果になります。一番下の数を 3 から 10 に増やす効果は、1 段階目の指数箇所を 2.1 倍する程度の効果、2 段階目の指数箇所に 0.68 を足す効果となり、3 段階目の指数箇所に至っては極めて小さな効果になります。たとえば、

$$3 \uparrow 3 \uparrow 3 \uparrow 3 \uparrow 3 \uparrow 3 = 10 \uparrow 10 \uparrow 10 \uparrow 10 \uparrow 12.56$$

という式において、$x = 12.56$ とすれば $2.1x + 0.68 = 27.056$ で、ほぼ $3 \uparrow 3 = 27$ と等しくなっているので、$2.1x + 0.68$ 程度の効果がある、という見積もりが正しいことが分かります。

Hypercalc で、$x = 10 \uparrow 10 \uparrow 10 \uparrow 10$ としたときの 2^x と x^x を計算したところ、同じ $10 \uparrow 10 \uparrow 10 \uparrow 10 \uparrow 10$ になりました。つまり、$2^x \approx x^x$ と考えて良いことになります。ここで $\approx$ の記号は、「その大きさの数の比較ではほぼ等しいとみなせる」という意味での近似の記号です。「ほぼ等しい」の意味は、「値の差が 0 に近い」という意味に限定されません。巨大数は正確な計算ができないので、このような近似の取り扱いが必要になり、今後もこのような意味で近似の記号 $\approx$ を使っていきます。2^x と x^x の比較について考えると、このレベルの巨大数は「対数を 4 回取る」ことで比較ができるため、対数を 4 回取って比較すると、その差は計算できない程に小さくなります。つまり、

$$\log(\log(\log(\log(2^x)))) \approx \log(\log(\log(\log(x^x)))) \approx 10$$

ということを、$2^x \approx x^x$ と書きました。

2.1.3 テトレーション

次は↑が2つ並んだ場合です。これを特別にパワータワー[4] (power tower) とも言います。

$$\begin{aligned} x \uparrow\uparrow y &= x \uparrow (x \uparrow\uparrow (y-1)) \\ &= x \uparrow (x \uparrow (x \uparrow\uparrow (y-2))) \\ &= x \uparrow (x \uparrow (x \uparrow (x \uparrow\uparrow (y-3)))) \\ &= \underbrace{x \uparrow x \uparrow \cdots \uparrow x}_{y \text{ 個の } x} \end{aligned}$$

となり、指数がタワーのように連なるので指数タワーとも言います。矢印表記のことを「タワー表記」と記されることもありますが、指数タワーはテトレーションの意味で使われるため、ペンテーション以上を含む場合には「矢印表記」という用語に統一しておく方が良いと思います。

また、これをテトレーション[5] (tetration) あるいは超冪（ちょうべき）とも言います。テトレーションは、グッドスタイン [6] がギリシャ語の接頭辞で4を意味する tetra-から作った言葉です。テトレーションを使うと、前章最後のベントレー数は

$$\sum_{i=0}^{9} 10 \uparrow\uparrow i$$

と書くことができます。

2.1.4 ハイパー演算子

テトレーションは、繰り返しで得られる4番目の演算子なので、ハイパー4演算子 (hyper-4) とも言います。一般にハイパー n 演算子 $H_n(a,b)$ は、以下で定義されます[7] 。

---

[4] MathWorld - Power Tower http://mathworld.wolfram.com/PowerTower.html

[5] http://ja.googology.wikia.com/wiki/テトレーション

[6] Goodstein, R. L. (1947). Transfinite Ordinals in Recursive Number Theory. *Journal of Symbolic Logic* 12(4): 123–129. doi:10.2307/2266486

[7] http://ja.googology.wikia.com/wiki/ハイパー演算子

【定義】ハイパー n 演算子 $H_n(a,b)$

$$H_n(a,b)=\begin{cases} b+1 & (n=0) \\ a & (n=1,b=0) \\ 0 & (n=2,b=0) \\ 1 & (n\geq 3,b=0) \\ H_{n-1}(a,H_n(a,b-1)) & (\text{それ以外}) \end{cases}$$

$H_0(a,b)=b+1$ は、後者
$H_1(a,b)=a+b$ は、加算（後者の繰り返し）
$H_2(a,b)=a\times b$ は、乗算（加算の繰り返し）
$H_3(a,b)=a^b$ は、冪乗（「べきじょう」と読む。乗算の繰り返し）
$H_4(a,b)=a\uparrow\uparrow b$ は、超冪（冪乗の繰り返し）またはテトレーション
$H_5(a,b)=a\uparrow\uparrow\uparrow b$ は、ペンテーション（テトレーションの繰り返し）
$H_6(a,b)=a\uparrow^4 b$ は、ヘキセーション（ペンテーションの繰り返し）
$H_n(a,b)=a\uparrow^{n-2}b(n\geq 3)$

それではテトレーション↑↑を連ねていきます。

$$\begin{aligned} 3\uparrow\uparrow 3 &= 3\uparrow 3\uparrow 3=7625597484987 \\ 3\uparrow\uparrow 3\uparrow\uparrow 3 &= 3\uparrow\uparrow 7625597484987 \\ &= \underbrace{3\uparrow 3\uparrow\cdots\uparrow 3\uparrow 3}_{7625597484987\text{ 個の }3} \end{aligned}$$

ということで、この時点でクラス 7625597484986 の巨大な数になります。Hypercalc で計算しようにも、「3^」を 7 兆個以上打ち込まないとなりませんので無理です。仮に打ち込んだとしたら、「7625597484983 PT (6.46 × 10 ^ 3638334640023)」という結果が表示されることでしょう。

$$3\uparrow\uparrow 3\uparrow\uparrow 3\uparrow\uparrow 3=\underbrace{3\uparrow 3\uparrow\cdots\uparrow 3\uparrow 3}_{3\uparrow\uparrow 3\uparrow\uparrow 3\text{ 個の }3}$$

となると、クラス 3 ↑↑ 3 ↑↑ 3 程度の巨大数です。Hypercalc では計算できません。まず 3 を 3 ↑↑ 3 ↑↑ 3 回入力することが不可能ですし、仮に

入力できて、計算の時間とメモリの制約を無視しても、Hypercalc の計算できる最大値は $10 \uparrow\uparrow (10^{10})$ であるとされていて、$10 \uparrow\uparrow (3 \uparrow\uparrow 3 \uparrow\uparrow 3)$ 程度のこの数はオーバーフローです。Hypercalc ですらオーバーフローしてしまう巨大数となりました。

そこでクラスの概念を拡張して**ハイパークラス**を定義します。これは本書独自の定義です。

【定義】自然数のハイパークラス

数列 $hc(n)$ を $hc(0) = 6, hc(n+1) = c(hc(n))$ で定義する。ここで $c(n)$ についてはクラスの定義 (p.27 の第 1.2 節) 参照。

ハイパークラス n の自然数とは、
(i) $n = 0$ の時は、$hc(0)$ 以下の自然数である。
(ii) $n > 0$ の時は、$hc(n-1)$ よりも大きく、$hc(n)$ 以下の自然数である。

この定義はクラスの定義とほとんど同じで、変わっているところは、$hc(n+1) = c(hc(n))$ の箇所です。クラスの定義では、$c(n+1) = 10^{c(n)}$ となっていました。クラスを上げるための演算は $10 \uparrow x$ ですが、ハイパークラスを上げるための演算は $c(x)$ を使っています。

ハイパークラス 1 はクラス 6 までの数です。ハイパークラス 2 は、N をハイパークラス 1 の上限であるとしたときに、クラス N までの数です。同様にハイパークラス 3 は、クラス（ハイパークラス 2 の数）で表される自然数です。

このように定義すると、$3 \uparrow\uparrow 3$ はハイパークラス 1、$3 \uparrow\uparrow 3 \uparrow\uparrow 3$ はハイパークラス 2、$3 \uparrow\uparrow 3 \uparrow\uparrow 3 \uparrow\uparrow 3$ はハイパークラス 3、というように、テトレーションを繰り返すごとにハイパークラスが 1 つ増加します。ハイパークラス 3 以上の数は、Hypercalc を使っても計算できません。

矢印が 3 つ並ぶと、テトレーションを繰り返す**ハイパー 5 演算子**となります。これを**ペンテーション**[8] (pentation) と言います。ジョナサン・バウ

[8] http://ja.googology.wikia.com/wiki/ペンテーション

アーズは、3 に 3 をペンテーションした数をトリトリ[9] (tritri) と名付けています。これからも出てくる数で、原始再帰でできる巨大数の例としては面白いので、定義を書いておきます。

【定義】トリトリ

トリトリ $= 3 \uparrow\uparrow\uparrow 3 = 3 \uparrow\uparrow 3 \uparrow\uparrow 3$
トリトリは、クラス 7625597484986、ハイパークラス 2 の巨大数である。

また、$10 \uparrow\uparrow\uparrow 100$ はギャゴルと呼びます[10]。これは、ハイパークラス 100 の巨大数です。

矢印が 4 つ並ぶと、ペンテーションを繰り返すハイパー **6** 演算子で、これをヘキセーション[11] (hexation) と言います。

$$\begin{aligned} 3 \uparrow\uparrow\uparrow\uparrow 3 &= 3 \uparrow\uparrow\uparrow 3 \uparrow\uparrow\uparrow 3 \\ &= 3 \uparrow\uparrow\uparrow \text{トリトリ} \\ &= \underbrace{3 \uparrow\uparrow 3 \uparrow\uparrow \cdots \uparrow\uparrow 3}_{\text{トリトリ個の } 3} \end{aligned}$$

となります。テトレーションの計算をトリトリ回繰り返した数が $3 \uparrow\uparrow\uparrow\uparrow 3$ なので、ハイパークラストリトリほどの大きさの数ということになります。なんだか分からないけどとにかく大きな数、というより他にないような大きな数です。これが、後のグラハム数の定義で出てくる $f(4)$ で、グラハム数はこの数がスタートになっています。

このように、↑という記号を導入したことで、四則演算やべき乗などの記号だけでは、とても書ききれない、Hypercalc をもってしても太刀打ちできないほどの想像を絶する大きな数を表現することができました。スキューズ数やベントレー数などの第 1 章で見たような巨大数は、トリトリから見ると、小さすぎて話にならない数と言うことができます。

[9] http://ja.googology.wikia.com/wiki/トリトリ
[10] http://ja.googology.wikia.com/wiki/ギャゴル
[11] http://ja.googology.wikia.com/wiki/ヘキセーション

さて、このような巨大数の大きさは、どのように評価すれば良いのでしょうか。そもそも正確な計算が現実的な時間ではできない数なので、やみくもに計算を繰り返しても「計算できませんでした」となるだけで何も分かりません。大切なことは、どのような計算をしているか、という構造を見ること、関数の性質をよく知ることです。矢印表記の性質として、

1. 同じ形の表記であれば、左の数を大きくするよりも、右の数を大きくする効果の方が大きい。
2. 数を大きくする効果よりも、繰り返し数を増やす効果の方が大きい。
3. 繰り返し数を増やす効果よりも、↑の数を増やす効果の方が大きい。

といったことがあるため、たとえば、以下のような関係は分かります。

$$\begin{aligned} 100 \uparrow\uparrow\uparrow 100 &< 3 \uparrow\uparrow\uparrow 101 \\ 100000 \uparrow\uparrow\uparrow 100000 \uparrow\uparrow\uparrow 100000 &< 3 \uparrow\uparrow\uparrow 3 \uparrow\uparrow\uparrow 3 \uparrow\uparrow\uparrow 3 \\ 3 \uparrow^{100} 3 \uparrow^{100} 3 \uparrow^{100} 3 \uparrow^{100} 3 &= 3 \uparrow^{101} 5 < 3 \uparrow^{102} 3 \end{aligned}$$

つまり、巨大数を作るという観点からは、使われる数を大きくしても、繰り返しの数を増やしても、↑の数を増やす効果にはかないません。↑の数を増やすことが最も効率的である、ということになります。色々な数と↑が入り乱れている矢印表記の数があったときには、一番↑の数が大きい場所の↑の本数を x として、$3 \uparrow^{x+2} 3$ という数を作れば、その数の方が大きい、ということになります。

さて、ここから先は Hypercalc で計算できないような大きな数の話になりますが、もう少し Hypercalc で遊んでみます。Hypercalc で計算できる演算は、四則演算、冪乗、階乗です。そこで階乗を繰り返したらどうなるかを計算してみましょう。ここで階乗記号を繰り返した!!!という記号は、通常は $10!!! = 10 \times 7 \times 4 \times 1$ のように定義されていますが、Hypercalc では $10!!! = ((10!)!)!$ として計算します。したがって、!を連続して打ち込めば、階乗の繰り返しを簡単に計算できます。ここでは、$3!^n = (\cdots(((3!)!)!)\cdots)$ と、!を n 回繰り返す演算を $!^n$ と表記します。ハイパー E 表記で結果を

書きます。

$$\begin{aligned}
3! &= 6\ (\text{クラス } 0)\\
3!^2 = (3!)! &= 720\ (\text{クラス } 1)\\
3!^3 = ((3!)!)! &= E3.242147497\#2\ (\text{クラス } 2)\\
3!^4 &= E3.242952972\#3\ (\text{クラス } 3)\\
3!^5 &= E3.242952972\#4\ (\text{クラス } 4)\\
3!^6 &= E3.242952972\#5\ (\text{クラス } 5)\\
3!^{100} &= E3.242952972\#99\ (\text{クラス } 99)
\end{aligned}$$

階乗は掛け算の繰り返しなので、冪乗程度の増加率です。それを繰り返すことで、クラスが 1 つずつ上がり、3.242952972 の部分は動かなくなります。クラス n の巨大数に階乗をすると、クラス $n+1$ の巨大数になる、ということになります。実際に、スターリングの近似式 $\log n! \approx n\log n - n$ ($\log$ は自然対数) について、両辺の対数を取って $\log(\log n!) \approx \log(n) + \log(\log n) \approx \log(n)$ とすれば、n が大きいときに階乗は指数関数と同じ程度の増加率になるということが理解できます。

2.1.5　連続関数化

「乗算の繰り返し」で定義される冪乗 a^x は、繰り返し回数である x の定義を自然数から実数へと拡張して連続関数とされているように、「冪乗の繰り返し」で定義されるテトレーションについても、繰り返し回数の実数への拡張がされています。その方法にはいくつかありますが、$a \uparrow\uparrow x$ を、実数 x に対して次のように定義する方法が簡明です。

$$a \uparrow\uparrow x = \begin{cases} \log_a(a \uparrow\uparrow (x+1)) & x \le -1 \\ 1 + x & -1 < x \le 0 \\ a \uparrow (a \uparrow\uparrow (x-1)) & 0 < x \end{cases}$$

すなわち $x > 0$ の時は

$$a \uparrow\uparrow x = \underbrace{a \uparrow a \uparrow \cdots \uparrow a}_{\text{floor}(x)+1 \text{ 個の } a} \uparrow (x - \text{floor}(x))$$

となります。ここで floor は床関数で、$\text{floor}(x)$ は x 以下の最大の整数です。拡張されたテトレーションは、$a = e$ のときに微分可能で導関数が連続な滑らな関数になります。

巨大数の大きさを比較するときには、a を同じ値にそろえればいいのですが、関数の滑らさを重視するのであれば $a = e$ にそろえるのが良く、実用的には $a = 10$ にそろえるのが便利だと思います。たとえば以下のように計算されます。

$$\begin{aligned}
\text{グーゴル} &= 10^{100} = 10 \uparrow 10 \uparrow 2 = 10 \uparrow 10 \uparrow 10 \uparrow \log_{10}(2) \\
&\approx 10 \uparrow 10 \uparrow 10 \uparrow 0.301 = 10 \uparrow\uparrow 2.301 \\
\text{グーゴルプレックス} &= 10^{10^{100}} = 10 \uparrow 10 \uparrow 10 \uparrow 2 \\
&\approx 10 \uparrow 10 \uparrow 10 \uparrow 10 \uparrow 0.301 = 10 \uparrow\uparrow 3.301
\end{aligned}$$

テトレーションをこのように連続関数化すると、テトレーションの逆関数である超対数 (super-logarithm) を次のように定義できます。

$$\text{slog}_a(x) = \begin{cases} \text{slog}_a(a^x) - 1 & x \le 0 \\ -1 + x & 0 < x \le 1 \\ \text{slog}_a(\log_a(x)) + 1 & 1 < x \end{cases}$$

同じ方針でペンテーションを連続関数化すると

$$a \uparrow^3 x = \begin{cases} \text{slog}_a(a \uparrow^3 (x+1)) & x \le -1 \\ 1 + x & -1 < x \le 0 \\ a \uparrow\uparrow (a \uparrow^3 (x-1)) & 0 < x \end{cases}$$

となります。計算の要領を示すために、$3 \uparrow\uparrow\uparrow 2.5$ を $10 \uparrow\uparrow x$ の形に書き直してみます。

$$3 \uparrow\uparrow\uparrow 2.5 \quad = \quad 3 \uparrow\uparrow 3 \uparrow\uparrow 3 \uparrow\uparrow 0.5 = 3 \uparrow\uparrow 3 \uparrow\uparrow 3 \uparrow 0.5$$

$$
\begin{aligned}
&\approx 3 \uparrow\uparrow 3 \uparrow\uparrow 1.73205 = 3 \uparrow\uparrow 3 \uparrow 3 \uparrow 0.73205 \\
&\approx 3 \uparrow\uparrow 3 \uparrow 2.235 \approx 3 \uparrow\uparrow 11.651 \\
&= 3 \uparrow 3 \uparrow 3 \uparrow 3 \uparrow 3 \uparrow 3 \uparrow 3 \uparrow 3 \uparrow 3 \uparrow 3 \uparrow 3 \uparrow 0.651 \\
&\approx 10 \uparrow 10 \uparrow 10 \uparrow 10 \uparrow 10 \uparrow 10 \uparrow 10 \uparrow 10 \uparrow (3.5045 \cdot 10^{15427}) \\
&\approx 10 \uparrow 10 \uparrow 10 \uparrow 10 \uparrow 10 \uparrow 10 \uparrow 10 \uparrow 10 \uparrow 10 \uparrow 15427.5446 \\
&\approx 10 \uparrow 10 \uparrow 10 \uparrow 10 \uparrow 10 \uparrow 10 \uparrow 10 \uparrow 10 \uparrow 10 \uparrow 10 \uparrow 4.1883 \\
&\approx 10 \uparrow 10 \uparrow 10 \uparrow 10 \uparrow 10 \uparrow 10 \uparrow 10 \uparrow 10 \uparrow 10 \uparrow 10 \uparrow 10 \uparrow 0.622 \\
&= 10 \uparrow\uparrow 10.622
\end{aligned}
$$

同様に計算すると、次のようになります[12]。

$$
\begin{aligned}
3 \uparrow\uparrow\uparrow 2.6 &\approx 10 \uparrow\uparrow 20.346 \\
3 \uparrow\uparrow\uparrow 2.7 &\approx 10 \uparrow\uparrow 56.804 \\
3 \uparrow\uparrow\uparrow 2.8 &\approx 10 \uparrow\uparrow 461.864 \\
3 \uparrow\uparrow\uparrow 2.9 &\approx 10 \uparrow\uparrow 88824.2 \\
3 \uparrow\uparrow\uparrow 3 &\approx 10 \uparrow\uparrow 7625597484986.041 \\
&\approx 10 \uparrow\uparrow\uparrow 2.31076
\end{aligned}
$$

ペンテーションの逆関数は超々対数 sslog で、次のようになります。

$$
\mathrm{sslog}_a(x) = \begin{cases} \mathrm{sslog}_a(a \uparrow\uparrow x) - 1 & x \leq 0 \\ -1 + x & 0 < x \leq 1 \\ \mathrm{sslog}_a(\log_a(x)) + 1 & 1 < x \end{cases}
$$

ここまでの考え方を一般化すると、自然数 $n \geq 2$ に対して $a \uparrow^n x$ を

$$
a \uparrow^n x = \begin{cases} \underbrace{\mathrm{ss...s}}_{n-2} \log_a(a \uparrow^n (x+1)) & x \leq -1 \\ 1 + x & -1 < x \leq 0 \\ a \uparrow^{n-1} (a \uparrow^n (x-1)) & 0 < x \end{cases}
$$

[12] http://ja.googology.wikia.com/wiki/ユーザーブログ:Kyodaisuu/テトレーションの連続関数化

のように連続関数化できます。このことから、定義域を $x > 0, n \geq 1, n \in \mathbb{N}$ とすると、

$$a \uparrow^n x = \begin{cases} a^x & 0 < x \leq 1 \text{ or } n = 1 \\ a \uparrow^{n-1} (a \uparrow^n (x-1)) & 1 < x \text{ and } n > 1 \end{cases}$$

とすることができます。この方法で、$e \uparrow^n x$ あるいは $10 \uparrow^n x$ の形で統一して表記すれば、大きさを比較しやすくなります。

2.2 グッドスタイン数列

グッドスタイン数列[13] [14] [15] (Goodstein Sequence) は、面白い性質を持った数列です。まずは n を底とした遺伝的記法 (hereditary base-n notation) を定義します。

自然数の n 進数表記は、

$$\sum_{i=0}^{k} a_i n^i = a_k n^k + a_{k-1} n^{k-1} + \cdots + a_0 n^0$$

と書く事ができます。たとえば 1234 の 10 進数表記は

$$1234 = 1 \times 10^3 + 2 \times 10^2 + 3 \times 10^1 + 4$$

です。ここで $a_i n^i$ は $n^i + n^i + \cdots + n^i$ なので、n^i の和で書くこともできます。たとえば

$$1234 = 10^3 + 10^2 + 10^2 + 10^1 + 10^1 + 10^1 + 10^0 + 10^0 + 10^0 + 10^0$$

です。ここでは $a_k n^k$ の形で話を進めます。自然数を $a_k n^k$ の和で表したときに、指数の部分をさらに n 進数に分解します。そこでまた出て来た指数の部分を、さらに n 進数に分解します。このようにすると、出てくる数が全て n 以下の数となるように、記述できます。これが遺伝的記法です。

[13] http://ja.googology.wikia.com/wiki/グッドスタイン数列
[14] http://mathworld.wolfram.com/GoodsteinSequence.html
[15] 田中一之, 角田法也, 鹿島亮, 菊池誠 (1997) 『数学基礎論講義—不完全性定理とその発展』日本評論社

1234 で 10 進数の場合は、すでに出てくる数がすべて 9 以下となっているので、上記のままで遺伝的記法になります。これを 3 を底にする遺伝的記法に直します。

$$\begin{aligned} 1234 &= 3^6 + 2 \times 3^5 + 2 \times 3^2 + 1 \\ &= 3^{2\times 3} + 2 \times 3^{3+2} + 2 \times 3^2 + 1 \end{aligned}$$

このように出てくる数がすべて 3 以下になります。続いて 2 を底にする遺伝的記法にします。

$$\begin{aligned} 1234 &= 2^{10} + 2^7 + 2^6 + 2^4 + 2 \\ &= 2^{2^3+2} + 2^{2^2+2+1} + 2^{2^2+2} + 2^{2^2} + 2 \\ &= 2^{2^{2+1}+2} + 2^{2^2+2+1} + 2^{2^2+2} + 2^{2^2} + 2 \end{aligned}$$

これが遺伝的記法です。続いてグッドスタイン数列の定義を記します。

【定義】グッドスタイン数列

自然数 n の b を底とする遺伝的記法を書き、b を $b+1$ に書き換えることで得られる数を $B[b](n)$ とする。

このとき $G_0(n) = n$ からはじまるグッドスタイン数列を $G_k(n) = B[k+1](G_{k-1}(n)) - 1$ で定義する。結果が 0 となったときに数列は終了する。

3 から始まるグッドスタイン数列[16] の計算をすると、

$$\begin{aligned} G_0(3) &= 3 = 2 + 1 \\ G_1(3) &= B[2](G_0(3)) - 1 = B[2](3) - 1 = 3 + 1 - 1 = 3 \\ G_2(3) &= B[3](G_1(3)) - 1 = B[3](3) - 1 = 4 - 1 = 3 \\ G_3(3) &= B[4](G_2(3)) - 1 = B[4](3) - 1 = 3 - 1 = 2 \\ G_4(3) &= B[5](G_3(3)) - 1 = B[5](2) - 1 = 2 - 1 = 1 \\ G_5(3) &= B[6](G_4(3)) - 1 = B[6](1) - 1 = 1 - 1 = 0 \end{aligned}$$

[16]The Goodstein sequence G(3) https://oeis.org/A215409

と計算できます。まず、最初に $G_0(3)$ を 2 を底とする遺伝的記法で「2+1」と書きます。次に、$G_1(3)$ の計算では、$G_0(3) = 3$ に対して、$B[2]$ の操作、つまり 2 を底とする遺伝的記法で、2 を 3 に書き換える操作をします。つまり「2+1」の 2 の部分を 3 に書き換えて、「3+1」とします。そこから 1 を引いて「3+1-1」と計算して、結果が 3 となります。これを 3 進数表記すると、そのまま「3」です。

$G_2(3)$ の計算も同様ですが、今度は 3 の底を 4 の底に書き換えるので、「3」の部分がそのまま「4」になって、そこから 1 を引きます。$G_3(3)$ の計算では、4 の底を 5 に書き換えますが、4 という数はないのでそのまま「3」となり、1 を引いて 2 になります。

以下同様に繰り返して、$G_5(3)$ で 0 に達して数列が終了します。

グッドスタイン数列は、$0, 1, 2, 3$ から始まるときにはすぐに終わってしまいますが、4 以上からスタートするとなかなか終わらず、数列の値が急激に上昇します。たとえば 19 から始まるグッドスタイン数列を計算すると、以下のようになります。

$$
\begin{aligned}
G_0(19) &= 19 = 2^{2^2} + 2 + 1 \\
G_1(19) &= 3^{3^3} + 3 + 1 - 1 = 3^{3^3} + 3 = 7625597484990 \\
G_2(19) &= 4^{4^4} + 4 - 1 = 4^{4^4} + 3 \\
G_3(19) &= 5^{5^5} + 2 \\
G_4(19) &= 6^{6^6} + 1 \\
G_5(19) &= 7^{7^7}
\end{aligned}
$$

ここまでは数は増えますが表記は簡潔になります。この次の段階で、表記が爆発します。$G_6(19)$ は、7^{7^7} の 7 を 8 に書き換えて 8^{8^8} にしてから、1 を引きます。$8^{8^8} - 1$ を 8 を底とする遺伝的表記をすると、とても書き切れない数になります。8 を 10 に変えて $10^{10^{10}} - 1$ を 10 進数表記することを考えると、これは 9 が 10^{10} 個続く 10^{10} 桁の数になります。そしてさらに、一番上の桁は指数部分が 9999999999 となって、これもまた 10 桁の数なので、さらに 10 進数で分解する必要があります。$8^{8^8} - 1$ の場合は、8^8 桁の非常に長い数列ができます。

$G_6(19)$ の末尾は、$\cdots 7\times 8^2+7\times 8+7$ となっています。ここから先は、底が 8, 9, 10, … と増えて行き、同時に 1 を引かれることで、一番最後の 7 が 6, 5, 4, … と 1 ずつ引かれていって、$G_{13}(19)$ に達したところで、$\cdots 7\times 15^2+7\times 15$ と定数項が落ちます。次に、16 を底として書き換えて $\cdots 7\times 16^2+7\times 16$ から 1 を引くときに、$7\times 16-1=111$ を 16 進数で書いて $6\times 16+15$ となり、今度は 15 が 0 になるまで続けますが、そうこうしているうちに底がどんどん大きくなり、どんどん急激に値が増加していきます。

さて、このように急速に増加するグッドスタイン数列ですが、必ずいつかは 0 となって終了することが証明されています。そこが面白いところです。したがって、この数列が終了するまでのステップ数を関数と考えることができます。n から始まるグッドスタイン数列がはじめて 0 になるまでのステップ数をグッドスタイン関数 $G(n)$ とします。たとえば $n=3$ のときは $G_0(3)$ からはじまって 6 ステップ目で $G_5(3)=0$ となるので、$G(3)=6$ となります。すなわち $G_k(n)=0$ となる最小の k に対して、$G(n)=k+1$ です。グッドスタイン数列は必ず終了するのでこのような定義で $G(n)$ の値は必ず定まり、$G(n)$ は n がすべての自然数に対して値を持つ全域関数となります。

グッドスタイン関数はとても増加速度が大きい関数になります。たとえば $G(4)=3\times 2^{402653211}-2$ となり、$G(5)$ はトリトリ $(3\uparrow\uparrow\uparrow 3)$ を超えます。トリトリ回以上の繰り返しということになると、まともに計算しても終わるはずがありません。もちろん計算するためのメモリも足りません。囲碁の完全解析を計算するよりもはるかに困難な、終わりの見えない計算になります。$G(6)$ でヘキセーションのレベルに到達し、$G(7)$ では↑を 7 つ重ねる計算になり、$G(8)$ では矢印表記では書けないような大きな数になって、$G(12)$ では次章で紹介するグラハム数を超えて、そこから先はさらに急速に増加していきます。

グッドスタイン関数の面白いところは、数列の定義はおとなしいのに、とてつもなく増加速度が大きい関数が定義されるところです。それでは、どの程度増加速度が大きい関数になるのでしょうか。この関数は原始再帰関

数ではないので、後の章で検証をします。

2.3 原始再帰関数

四則演算しか知らない人が巨大数を作るとしたら、たとえば「99999×99999」とか「$999999 \times 999999 \times 99999$」などと書くでしょう。ところがこのようにしてたくさん 9 を並べて、×をいっぱい使って数を書いたところで、「$10^{10^{1000}}$」にはかないません。「$10^{10^{10^{1000}}}$」ほどの数を作るためには、10^{1000} 桁ほどの数字やかけ算記号を書く必要があるからです。これは四則演算によって作られる関数よりも、べき乗関数の方が増加速度が飛躍的に大きいためです。さらにクヌースの矢印表記という新しい関数を取り入れることで、べき乗などの表現をいくら使っても書き表せないほどの大きな数を作ることができました。

このようにより増加速度が大きい関数を定義することで、より巨大な数を作ることができるため、巨大数を作ることは巨大関数を作る事と同じことになります。今後はいかにして巨大関数を作るか、という議論が中心になります。それはそのまま、巨大数を作るということになります。したがって、本書のタイトルも「巨大関数論」とする方が内容に即していますが、巨大数への想いを込めて、あえて「巨大数論」としました。

さて、巨大関数を作ったとして、その巨大関数がどの程度の大きさなのかを評価する必要があります。そのために再帰理論を導入します。

本章でこれまでに紹介した関数は、グッドスタイン関数を除けば、すべて原始再帰関数 (primitive recursive function)[17]になります。

【定義】原始再帰関数

原始再帰関数は、以下のように定義される。

1. ゼロ関数: $Z() = 0$ （0 変数の関数、つまり定数 0）は原始再帰

[17]田中一之, 角田法也, 鹿島亮, 菊池誠 (1997) 『数学基礎論講義—不完全性定理とその発展』日本評論社

関数である。

2. 後者関数: $S(x) = x + 1$ は原始再帰関数である。

3. 射影関数: $P_i^n(x_1, x_2, \cdots, x_n) = x_i (1 \leq i \leq n)$ は原始再帰関数である。

4. 関数合成: $g_i : \mathbb{N}^n \to \mathbb{N}$, $h : \mathbb{N}^m \to \mathbb{N} (1 \leq i \leq m)$ が原始再帰関数のとき、

$$f(x_1, ..., x_n) = h(g_1(x_1, ..., x_n), ..., g_m(x_1, ..., x_n))$$

で定義される合成関数 $f = h(g_1, ..., g_m) : \mathbb{N}^n \to \mathbb{N}$ は原始再帰関数である。

5. 原始再帰: $g : \mathbb{N}^n \to \mathbb{N}$, $h : \mathbb{N}^{n+2} \to \mathbb{N}$ が原始再帰関数のとき、

$$\begin{aligned} f(x_1, ..., x_n, 0) &= g(x_1, ..., x_n) \\ f(x_1, ..., x_n, y + 1) &= h(x_1, ..., x_n, y, f(x_1, ..., x_n, y)) \end{aligned}$$

で定義される関数 $f : \mathbb{N}^{n+1} \to \mathbb{N}$ は原始再帰関数である。

ここで、関数合成と原始再帰を、それぞれ合成作用素、原始再帰作用素とも呼びます。以下は、原始再帰関数の例です。

1. 定数関数 $f(x) = n$
2. 加算 $x + y$
3. 乗算 xy
4. 冪乗 x^y
5. 階乗 $x!$
6. 大小関係 $x < y$ や同値関係 $x = y$ を判定する関数
7. 素数を判定する関数
8. n 番目の素数を返す関数

9. 以上を繰り返した関数

定数関数は、ゼロ関数に後者関数を繰り返し適用することで生成できます。加算 plus は、後者 S と plus の原始再帰を繰り返すことで、$\mathrm{plus}(x,0)=x, \mathrm{plus}(x,y{+}1)=S(\mathrm{plus}(x,y))$ と計算できます。つまり、後者関数 $(+1)$ を y 回繰り返す、という関係を原始再帰で計算できます。同様に、乗算 times は加算を繰り返すので、加算の原始再帰 $\mathrm{times}(x,y{+}1)=\mathrm{plus}(x,(x,\mathrm{times}(x,y))$ で計算できます。べき乗は乗算の繰り返しなので同様に計算できます。階乗も乗算の繰り返しなので、繰り返し方を工夫すれば原始再帰で計算できます。

さらに原始再帰で大小関係の比較や同値関係の判定もできるため、x は y の約数であるかどうかという判定ロジックも原始再帰となり、約数が存在するかどうかを調べる素数判定も原始再帰になります。したがって素数かどうかを順番に調べることで、n 番目の素数も原始再帰で計算できます。このように、原始再帰の範囲は非常に広く、色々な計算ができます。ふだん扱っている算術的関数のほとんどは原始再帰関数となるようです。

後者の原始再帰で加算を、加算の原始再帰で乗算を、乗算の原始再帰で冪乗を計算することができました。これは、ハイパー演算子で言えば、ハイパー 0 の原始再帰からハイパー 1 を、ハイパー 1 の原始再帰からハイパー 2 を、ハイパー 2 の原始再帰からハイパー 3 を計算したことになります。したがって、ハイパー 3 の原始再帰からハイパー 4 の超冪を計算できます。このように、原始再帰を繰り返すことで、ハイパー演算子が計算できます。つまり、クヌースの矢印表記は原始再帰で計算できます。

なお、原始再帰関数は原始帰納関数と書かれることもあり、本書でも当初はそのように表記していましたが、英語の recursive を再帰と訳し、induction を帰納と訳すことに統一する方が意味が分かりやすくすっきりとするため、本書全体を 2 版からその方針で書き改めることとしました。

【コラム】アンドレ・ジョイスとグーゴロジー

アンドレ・ジョイス (André Joyce) は、巨大数論を意味するグーゴロジー (googology) という言葉を考案した人です。彼は、原始再帰レベルの関数と、その関数によって定義される多くの巨大数の名前を考えました。そして、グーゴロジーだけではなく、巨大数の名前を意味するグーゴロジズム (googologism)、巨大数研究者を意味するグーゴロジスト (googologist) というような言葉も考えました。このジョイスは、どんな人だったのでしょうか。

ジョイスに関する情報は長らく謎につつまれたままで、ネット上の André Joyce Fan Club というファンクラブのサイトに情報があるだけでした。そのサイトでは、ジョイスはフランス系のアメリカ人で作家であるとして紹介されています。

実は、アンドレ・ジョイスはファンクラブのサイトを開いたマイケル・ジョセフ・ハルム (Michael Joseph Halm) という人が考えた実在しない人物でした。André Joyce という名前は聖書によく出てくる and rejoice という言葉をもじった言葉遊びだったようです。つまり、ハルムのペンネームがジョイスだったとする解釈も可能ですが、ファンクラブのサイトはあたかも別人のごとく紹介していたので、ハルムが架空の人物に関する伝記をもっともらしく作成して遊んでいた、と解釈するのが妥当だと思います。

第 3 章

2 重再帰関数

本章では 2 重再帰関数と、2 重再帰関数によって定義される巨大数について解説します。2 重再帰関数は原始再帰関数よりも「強い」関数なので、本章で紹介する巨大数は前章までで紹介した巨大数とは比べものにならないほど大きな巨大数となります。では、関数が「強い」とはどういうことでしょうか。

本書の目的は「巨大数を作る」ことで、そのためには「強い関数を作る」必要があります。そこで、本章で原始再帰と 2 重再帰の比較をすることで、強い関数を作るための手法の 1 つである「対角化」について解説します。対角化をすることでなぜ関数が強くなるのかを理解することが、本書を読み進める上で重要なポイントとなります。

3.1 アッカーマン関数

2 重再帰関数を定義する前に、その具体例としてドイツの数学者ヴィルヘルム・アッカーマン (Wilhelm Friedrich Ackermann, 1869–1962) が考案したアッカーマン関数[1] [2] (Ackermann function) を紹介します。これは値が爆発的に大きくなる関数です。

[1] http://ja.googology.wikia.com/wiki/アッカーマン関数

[2] Ackermann, W. (1928) Zum Hilbertschen Aufbau der reellen Zahlen. *Mathematische Annalen* 99: 118–133. doi:10.1007/BF01459088

【定義】アッカーマン関数

非負整数 x, y に対し、以下のように定義される関数 $A(x, y)$ をアッカーマン関数とする。

$$A(0, y) = y + 1 \tag{3.1}$$
$$A(x + 1, 0) = A(x, 1) \tag{3.2}$$
$$A(x + 1, y + 1) = A(x, A(x + 1, y)) \tag{3.3}$$

$A(x, y)$ を、具体的に計算してみましょう。

$$\begin{aligned}
A(1,1) &= A(0, A(1,0)) = A(0, A(0,1)) = A(0,2) = 3 \\
A(1,2) &= A(0, A(1,1)) = A(0,3) = 4 \\
A(1,3) &= A(0, A(1,2)) = A(0,4) = 5 \\
A(1,y) &= A(0, A(1,y-1)) = A(1,y-1) + 1 = y + 2 \\
A(2,0) &= A(1,1) = 3 \\
A(2,1) &= A(1, A(2,0)) = A(1,3) = 5 \\
A(2,2) &= A(1, A(2,1)) = A(1,5) = 7 \\
A(2,3) &= A(1, A(2,2)) = A(1,7) = 9 \\
A(2,y) &= A(1, A(2,y-1)) = A(2,y-1) + 2 = 2y + 3 \\
A(3,0) &= A(2,1) = 5 \\
A(3,1) &= A(2, A(3,0)) = A(2,5) = 13 = 2^4 - 3 \\
A(3,2) &= A(2, A(3,1)) = A(2,13) = 29 = 2^5 - 3 \\
A(3,3) &= A(2,29) = 61 = 2^6 - 3 \\
A(3,y) &= 2^{y+3} - 3 \text{ (帰納法で証明)} \\
A(4,0) &= A(3,1) = 13 \\
A(4,1) &= A(3, A(3,1)) = A(3,13) = 2^{16} - 3 = 65533 \\
A(4,2) &= A(3,65533) = 2^{65536} - 3 \approx 2 \times 10^{19728} \\
A(4,3) &\approx A(3, 2 \times 10^{19728}) \approx 10^{6 \times 10^{19727}}
\end{aligned}$$

『寿司 虚空編』の第 8 話では、以下のように $x > 2$ の時のアッカーマン関数が矢印表記を使ってあらわされることが解説されています。

$$A(x, y) \quad = \quad 2 \uparrow^{x-2} (y + 3) - 3$$

この式は数学的帰納法で証明できます。

【証明】

- $A(3, 0) = 2 \uparrow 3 - 3 = 5$ は成立
- $A(3, y)$ で成立すると仮定すると、次のように $A(3, y + 1)$ でも成立

$$\begin{aligned} A(3, y+1) &= A(2, A(3, y)) = A(2, 2^{y+3} - 3) \\ &= 2(2^{y+4} - 3) = 2^{y+4} - 3 \end{aligned}$$

  よって、$A(3, y)$ で成立

- ある $x > 2$ に対して $A(x, y)$ が成立すると仮定すると、$A(x+1, y)$ がすべての $y \geq 0$ で成立することを y の帰納法で以下のように証明できる

$$\begin{aligned} A(x+1, 0) &= A(x, 1) = 2 \uparrow^{x-2} 4 - 3 \\ &= 2 \uparrow^{x-2} (2 \uparrow^{x-2} 2) - 3 = 2 \uparrow^{x-1} 3 - 3 \\ A(x+1, y) &= A(x, A(x+1, y-1)) = A(x, 2 \uparrow^{x-1} (y+2) - 3) \\ &= 2 \uparrow^{x-2} (2 \uparrow^{x-1} (y+2)) - 3 \\ &= 2 \uparrow^{x-2} (\underbrace{2 \uparrow^{x-2} 2 \uparrow^{x-2} \ldots \uparrow^{x-2} 2}_{y+2 \text{ 個の } 2}) - 3 \\ &= 2 \uparrow^{x-1} (y+3) - 3 \end{aligned}$$

□

すなわちアッカーマン関数 $A(x, y)$ は、$x = 0$ の時は後者、$x = 1$ の時は加算、$x = 2$ の時は乗算、$x = 3$ の時は冪乗、$x = 4$ の時はテトレーショ

ン、$x = 5$ の時はペンテーション、とハイパー x 演算子に相当する増加速度の関数となっています。

3.2　2 重再帰関数

アッカーマン関数はどんな原始再帰関数よりも増加速度が大きい数です。つまり任意の原始再帰関数 $f(x)$ に対して、

$$f(x) < A(c, x)$$

となる c が存在します。それはアッカーマン関数が 2 重再帰関数であるためです。そのことをこれから示します。なお、ここでは 1 変数関数について議論を進めますが、$f(x)$ が多変数関数であっても同様の議論ができます。

前章の原始再帰の定義式 (p.62) において

$$n = 1, B(y) = f(x_1, y), C(n) = h(x_1, y, n)$$

とすると、

$$\begin{aligned} B(0) &= g(x_1) \\ B(y+1) &= C(B(y)) \end{aligned}$$

となるため、

$$B(y) = C^y(g(x_1))$$

と書けます。つまり $B(y)$ は関数 $g(x_1)$ に対して $C(n)$ の合成を y 回繰り返した関数です。合成の回数を変数とする関数を作ることが、原始再帰の操作となっています。このことを「操作を数え上げる」と表記することとします。すなわち原始再帰の操作は合成操作を数え上げます。したがって、関数 $f(x)$ から、合成操作を有限回 (n 回) 繰り返して得られるいかなる関数 $g(x)$ よりも、初期関数から原始再帰の操作によって得られる関数 $f(x)$ の方が大きくなります。

たとえば $g_0(x)$ に $f(x)$ の合成操作を n 回繰り返して得られる関数 $g_n(x) = f^n(n)$ について、

$$g_1(x) = f(x) \quad : \quad g_1(1) = f(1), g_1(2) = f(2), ..., g_1(n) = f(n)$$

$$\begin{aligned} g_2(x) = f^2(x) \quad &: \quad g_2(1) = f^2(1), g_2(2) = f^2(2), ..., g_2(n) = f^2(n) \\ &\vdots \\ g_n(x) = f^n(x) \quad &: \quad g_n(1) = f^n(1), g_n(2) = f^n(2), ..., g_n(n) = f^n(n) \end{aligned}$$

といった表を作ります。このとき関数 $f(x)$ に合成操作を $x-1$ 回繰り返して得られる関数 $g_x(x) = f^x(x)$ は、

$$g_x(x) = f^x(x) : g_x(1) = f(1), g_x(2) = f^2(2), ..., g_x(n) = f^n(n)$$

と書くことができて、これは上の表において右辺を左上から右下へ対角線上に拾い上げています。

このように操作の回数を数え上げることは対角線上に拾い上げて関数を作成する操作と対応することから、操作の対角化 (diagonize) であるとします。対角線を拾い上げるところは、カントールの対角線論法と似ています。

さて、アッカーマン関数の漸化式 (3.3) (p.66) において、$f(y) = A(x, y), g(y) = A(x+1, y)$ とすると、$g(y+1) = f(g(y))$ となり、したがって

$$g(y) = f^y(g(0)) = f^y(f(1)) = f^{y+1}(1)$$

と書けます。すなわち $g(y)$ は関数 $f(y)$ に対して合成を y 回繰り返して 1 を代入した関数です。これは対角線ではありませんが関数の合成回数を数え上げています。関数 $f^n(y)$ と $f^{y+1}(1)$ を比べると、$y > 10n$ において

$$f^{y+1}(1) = f^n[f^{y-n+1}(1)] > f^n[f^{0.9y+1}(1)] > f^n(y)$$

となるため、合成を有限回繰り返したいかなる関数よりも合成回数を数え上げる関数の方が大きくなります。つまり合成回数を数え上げることで、関数の対角列を取る「対角化」と等しい効果が得られます。

このように $f(y)$ から $g(y)$ を生成する操作は、合成を数え上げる原始再帰操作となるため、$A(x, y)$ に立ち戻って考えると、$A(x+1, y)$ は $A(x, y)$ に対して原始再帰の操作を 1 回操作した関数です。このことから $A(x, y)$ は $A(0, y)$ に対して原始再帰の操作を x 回繰り返した関数であり、原始再帰の

操作を数え上げています。このように合成作用素を数え上げるような原始再帰を数え上げる **2 重再帰** (double recursion) を、以下で定義します。

【定義】2 重再帰

合成作用素を数え上げるような原始再帰操作を数え上げる操作を 2 重再帰作用素（操作）とする。
2 重再帰関数 (double recursive function) とは、定義域と値域が非負整数である非負整数個の引数をとる関数で、引数に対し、ゼロ、後者、射影、合成、原始再帰、2 重再帰の作用素（操作）を有限回適用した関数である。ただしその中で 2 重再帰操作を少なくとも 1 回含むものとする。

射影、合成、原始再帰の操作を n 回繰り返して作られた原始再帰関数 $f(x)$ について、その中で最も関数を増加させる効果が大きい原始再帰の操作を n 回繰り返して作られた関数 $A(n,y)$ は $f(x)$ よりも大きくなります。すなわち $x > n$ において $f(x) < A(x,y)$ ですから、$A(x,y)$ は $f(x)$ と比べて本質的に大きい、ということになります。

関数 f, g に対して、ある n があって、$x > n$ であれば $f(x) > g(x)$ が成り立つ時、f は g を**支配する** (dominate) と言います。アッカーマン関数は、あらゆる原始再帰関数を支配する関数です。また本書では関数 f が関数 g を支配するときに、関数 f は関数 g よりも「強い」という表現も使います。巨大数を作るためにより強い関数を作る、と表現すると感覚的に分かりやすいためです。

ここで、原始再帰関数から 2 重再帰へという流れは、ハンガリーの数学者ローザ・ペータ (Rózsa Péter, 1905–1977) の再帰関数理論が元となっています。ペータが 1932 年にチューリッヒの国際数学者会議で発表した「再帰関数」(Rekursive Funktionen; ドイツ語) と、1936 年にオスロの国際数学者会議で発表した「2 段階目の再帰関数について」 (Über rekursive Funktionen der zweite Stufe) は、現代の再帰関数理論の基礎となっているとされます。そして 1951 年には『再帰関数』(Rekursive Funktionen) という本を発行し、生涯再帰関数に関する論文を発表し続けたことから、ペータは再帰関数理論の創始者と言われます。

Wikipedia の英語版では、double recursion の項にロビンソン[3]がペータの定義を元に定義した次のような 2 重再帰関数の定義が書かれています。関数 $G(n,x)$ が与えられた関数に対して 2 重再帰であるとは、このような再帰的定義がされていることを言います。

1. $G(0,x)$ は与えられている x の関数
2. $G(n+1,0)$ は z の関数である $G(n,z)$ と与えられた関数からの代入によって得られる
3. $G(n+1,x+1)$ は $G(n+1,x)$ と $G(n,z)$ と与えられた関数からの代入によって得られる

この 2 重再帰の定義は、先ほど示した 2 重再帰の定義とおおまかに同じですが厳密には一致しません（たとえば、必ずしも原始再帰を数え上げることを要していない)。しかし具体例としてアッカーマン関数と同じ 2 重再帰関数を示して、それがすべての原始再帰関数を支配することを、ロビンソンは示しています。本書における多重再帰の定義はペータの定義と厳密には一致しないかもしれませんが、再帰の段階を重ねることでより複雑な多重再帰関数を作るという本質は失われていないと思います。

数理論理学や計算機科学においては、「原始再帰関数」の次に多重再帰ではなくて**再帰関数** (recursive function) すなわち**計算可能関数** (computable function) を考えることが多いようです。再帰関数の一つの定義は、原始再帰関数で定義されたゼロ関数、後者関数、射影関数、関数合成、原始再帰に、**最小化** (minimisation) を加えたものです。

すべての引数について定義されている全域的な再帰関数を全域再帰関数 (total recursive function)、全域性の条件を除去したものを部分再帰関数 (partial recursive function) と言います。ここでは、参考までに部分再帰関数における最小化の定義を示します。

$g:\mathbb{N}^{n+1}\to\mathbb{N}$ を部分再帰関数とします。

> $g(x_1,\ldots,x_n,c)=0$ で、かつ各 $z<c$ に対し $g(x_1,\ldots,x_n,z)$ が定義され 0 以外の値を取る

[3]Robinson, R. M. (1948) Recursion and double recursion. *Bulletin of the American Mathematical Society* 54: 987–993. doi:10.1090/S0002-9904-1948-09121-2

ならば $\mu y(g(x_1 \ldots x_n, y) = 0) = c$ とし、そのような c が存在しないとき $\mu y(g(x_1 \ldots x_n, y) = 0)$ は定義されないとします。ここで、μ は μ 作用素 (minimisation operator) です。このとき、

$$f(x_1, \ldots, x_n) \sim \mu y(g(x_1 \ldots x_n, y) = 0)$$

となる関数 $f : \mathbb{N}^n \to \mathbb{N}$ は部分再帰関数になります。ここで、等号 $\sim$ は、f と g が同じ定義域をもち、その上で両関数の値が一致するときに $f \sim g$ とします。

通常、アッカーマン関数は「原始再帰関数ではないが再帰関数である」ような関数の例として登場します。そして再帰関数の範囲はとても広く、この本では第 7 章までは再帰関数によって定義される巨大数について取り扱い、再帰関数の中で関数をどんどん強くしていきます。2 重再帰関数は原始再帰関数よりも強い関数の第 1 歩で、ここからさらに強くしていこうというのが、これからの本書の道筋になります。通常はアッカーマン関数の次に第 8 章の再帰関数を支配するビジービーバー関数あたりまで一気に議論が進んでしまうところを、より丁寧に関数のレベルを高める構造を追いかけるところが、巨大数論ならではということです。

3.3　モーザー数

スタインハウス・モーザー表記[4] [5] (Steinhaus-Moser notation) として知られる表記があります。MathWorld によれば、スタインハウスの多角形表記 (Steinhaus's polygon notation) が定義されたのは 1983 年だとされています。これはスタインハウスが出版した Mathematical Snapshots という本の 1983 年版[6]に書かれているためですが、Mathematical Snapshots の 1950 年版にはすでに多角形表記はあったようなので、初出がいつなのかははっきりしません。そして、モーザーがどこで多角形表記を公表したのか

[4] http://ja.googology.wikia.com/wiki/スタインハウス・モーザー表記

[5] MathWorld - Steinhaus-Moser Notation http://mathworld.wolfram.com/Steinhaus-MoserNotation.html

[6] Steinhaus, H. (1983) Mathematical Snapshots. (Galaxy Books) Oxford University Press.

もはっきりしませんが、MathWorld と Wikipedia には Moser の出典はなしにそのように記録されています。

多角形表記では多角形の中に数を入れる書き方をして、多角形を重ねることと、多角形の辺の数を増やすことで、大きな数を作ります。まず 3 角形の中に x を書くことで、x^x を表記します。ここでは x 角形の中の y を $m(x,y)$ と書きます。

$$\text{(x)} = m(3,x) = x^x = x \uparrow x$$

次に、4 角形の中の x は、x 重の 3 角形の中に x が入っている数です。すなわち

$$\boxed{\text{x}} = m(4,x) = \underbrace{m(3,m(3,\cdots m(3,x)\cdots))}_{x \text{ 個の } m}$$

となります。そして 5 角形の中の x は、x 重の 4 角形の中の x です。

スタインハウスは、5 角形の代わりに円を使ってここで終わりましたが、モーザーはそれを n 角形に拡張しました。これがモーザーの多角形表記です。このとき 5 角形の中の 2 をメガ (mega) として、メガ角形の中の 2 をモーザー数[7] (Moser) としました。

【定義】モーザー数

モーザーの多角形表記で、5 角形の中の 2 をメガとする。メガ角形の中の 2 をモーザー数とする。
モーザー数 $= m(m(5,2),2)$

メガ $= m(5,2)$ の大きさは

$$\begin{aligned} m(3,2) &= 2^2 = 4 \\ m(4,2) &= m(3,m(3,2)) = m(3,4) = 4^4 = 256 \\ m(5,2) &= m(4,m(4,2)) = m(4,256) \end{aligned}$$

[7]巨大数研究 Wiki - モーザー数 http://ja.googology.wikia.com/wiki/モーザー数

となります。ここで $m(4,x)$ の計算は、冪乗の計算を x 回繰り返すので、テトレーションと同じ程度の増加になります。テトレーションの場合には、x^y の y の部分だけが大きくなるところ、$m(4,x)$ では x と y が同時に大きくなるので、厳密には $m(4,x)$ の方が増加は速いのですが、x の部分を大きくすることは、y を増加させることと比べるとたいして増加には寄与しないので、結局は同程度ということになります。Hypercalc で確認してみます。

$$\begin{aligned}
x_1 = m(3,256) &= 256^{256} \approx 3.231700607153 \times 10^{616} \text{ (クラス 2)} \\
x_2 = m(3,x_1) = x_1^{x_1} &\approx 10 \uparrow (1.992373902865 \times 10^{619}) \text{ (クラス 3)} \\
&\approx 10 \uparrow 10 \uparrow 619.29937 \\
x_3 = m(3,x_2) = x_2^{x_2} &\approx 10 \uparrow 10 \uparrow 10 \uparrow 619.29937 \text{ (クラス 4)} \\
x_4 = m(3,x_3) = x_3^{x_3} &\approx 10 \uparrow 10 \uparrow 10 \uparrow 10 \uparrow 619.29937 \text{ (クラス 5)} \\
&= E619.29937\#4 \text{ (ハイパー E 表記)} \\
x_5 &\approx E619.29937\#5 \text{ (ハイパー E 表記)}
\end{aligned}$$

3 ↑の繰り返しも、階乗の繰り返しも、x^x の繰り返しも、すべてある段階以上は 10 ↑の繰り返しと同じで、10 のタワーの右肩の数はびくともしなくなります。この計算から、

$$m(4,256) = x_{256} \approx E619.29937\#256 \text{ (クラス 257)}$$

と、メガの大きさが分かります。

$m(x,y)$ は、$A(x,y)$ と同様に増加する 2 重再帰関数です。$m(4,x)$ は↑↑つまりテトレーションに相当するのと同様に、$m(5,x)$ はペンテーションに相当し、$m(x,y)$ は、ハイパー x 演算子、つまり $\uparrow^{x-2}$ に相当します。ここで、x を 1 つ増やす効果が原始再帰で、その回数を数え上げることが 2 重再帰なので、たとえば $x=5$ と固定したときには原始再帰ですが、x を変数とすると 2 重再帰になります。したがって、メガは原始再帰関数で定義できる数ですが、$m(x,y)$ の x にとても大きな数を代入したモーザー数は、2 重再帰関数によって定義される数です。

3.4 グラハム数

1971 年に、アメリカの数学者ロナルド・グラハム (Ronald Graham) とブルース・ロートシルト (Bruce Lee Rothschild)[8] は「n 次元の超立方体の合計 2^n 個の頂点をすべて結び、それを赤と青の二色のいずれかに塗る。このとき n が充分大きいならば、どのような塗り方をしても、必ず同一平面上にある 4 点でそれらを結ぶ線がすべて同一の色であるものが存在する」という定理（グラハムの定理）を発表しました。この定理の中の「十分大きい n」の 1 つがグラハム数です。 グラハムは論文で、そのような数の例として、以下のような数を示しました。

$$\begin{aligned} F(1,n) &= 2^n, F(m,2) = 4, m \geq 1, n \geq 2, \\ F(m,n) &= F(m—1, F(m,n—1)), m \geq 2, n \geq 3. \\ N &\leq F(F(F(F(F(F(12,3),3),3),3),3),3),3) \end{aligned}$$

クヌースが矢印表記を考案したのは 1976 年であり、この論文が執筆された時点では矢印表記がまだ使えなかったのですが、矢印表記を使うと $F(m,n) = 2 \uparrow^m x$ であり $f(x) = 2 \uparrow^m 3$ としたとき $N \leq f^7(12)$ と書くことができます。これは非常に大きい数ですが、その後、アメリカの数学者マーティン・ガードナー (Martin Gardner, 1914-2010) が 1977 年にサイエンティフィック・アメリカンにさらに大きな数を上限として発表しました[9]。この記事で発表された数がグラハム数[10] [11] (Graham's number) として有名になり、1980 年のギネスブック[12]に

> The highest number ever used in a mathematical proof is a bounding value published in 1977 and known as Graham's number. It concerns bichromatic hypercubes and is inexpress-

[8] Graham, R. L. and Rothschild, B. L. (1971) Ramsey's theorem for n-parameter sets. *Transactions of the American Mathematical Society* 159: 257–292. doi:10.1090/S0002-9947-1971-0284352-8

[9] Gardner, M. (1977) Mathematical games: In which joining sets of points leads into diverse (and diverting) paths. *Scientific American* 237(5), 18–28. doi:10.1038/scientificamerican1177-18

[10] 巨大数研究 Wiki - グラハム数 http://ja.googology.wikia.com/wiki/グラハム数

[11] 『寿司 虚空編』第 1 話 `http://comic.pixiv.net/viewer/stories/6994`

[12] Norris McWhirter et al. (1980) Guiness Book of World Records 1980. p. 193.

ible without the special “arrow” notation, devised by Knuth in 1976, extended to 64 layers.

つまり「数学の証明で使われたことのある最大の数」として記録されました。

【定義】グラハム数

$$f(x) = 3 \uparrow^x 3 = 3 \underbrace{\uparrow \ldots \uparrow}_{x \text{ 個の} \uparrow} 3$$

としたときの $f^{64}(4)$ を、グラハム数とする。すなわち

$$\text{グラハム数} = \left.\begin{array}{c} 3\underbrace{\uparrow \ldots\ldots\ldots\ldots \uparrow}3 \\ 3\underbrace{\uparrow \ldots\ldots\ldots \uparrow}3 \\ \vdots \\ 3\underbrace{\uparrow \ldots\ldots \uparrow}3 \\ 3 \underbrace{\uparrow\uparrow\uparrow\uparrow}3 \end{array}\right\} 64 \text{ 段}$$

これはグラハムが 1971 年に論文で発表した数よりもさらに大きいもので、グラハムが未公表の論文に書いていたことがあるようです。グラハムが論文で発表した数は、このグラハム数と区別して小グラハム数と呼ばれます。

グラハム数の定義において、$f(x)$ は↑の個数を数え上げる 2 重再帰で、$f^{64}(4)$ の計算は関数 $f(x)$ に対する原始再帰です。したがってグラハム数は 2 重再帰に原始再帰を重ねた数で、2 重再帰です。グラハム数の大きさは、次に紹介するコンウェイのチェーン表記を使うと評価できます。

3.5　コンウェイのチェーン表記

イギリスの数学者ジョン・コンウェイ (John Horton Conway) が『数の本』[13] に書いたチェーン表記[14] (chained arrow notation) の定義を、以下に

[13] Conway, J. H. and Guy, R. K. (1996) The book of Numbers. Copernicus. 訳書『数の本』丸善出版.

[14] 巨大数研究 Wiki - チェーン表記 http://ja.googology.wikia.com/wiki/チェーン表記

記します。

【定義】コンウェイのチェーン表記

a, b, c を正の整数、X を 1 つ以上の正の整数のチェーン $a \to b \to ... \to c$ とする。

ルール 1: $a \to b \to c = a \uparrow^c b = a \underbrace{\uparrow \ldots \uparrow}_{c} b$

ルール 2: $X \to 1 = X$

ルール 3: $X \to 1 \to a = X$

ルール 4: $X \to (a+1) \to (b+1) = X \to (X \to a \to (b+1)) \to b$

ここで、ルール 1 は次のルール 1' に変えても同じである。

ルール 1': $a \to b = a^b$

コンウェイの『数の本』では、大きな数についての話の中でグラハム数を紹介した上で、チェーン表記を定義して、チェーン表記で「$3 \to 3 \to 3 \to 3$」はグラハム数よりも大きいと書いています。Aeton の巨大数動画シリーズで[15]、チェーン表記が詳しく解説されています。

クヌースの矢印表記を自然に拡張するとチェーン表記になります。ルール 1 に矢印表記がすでに使われているため、そのことは一見して分かりにくいのですが、実はルール 1 をルール 1' に変えても同じです。そのことは次のように証明できます。

【証明】

ルール 1',2,3,4 が成り立つときに、ルール 1 が成り立つ事を、c に関する帰納法で示す。

(1) $c = 1$ のとき

$$\begin{aligned} a \to b \to 1 &= a \to b \text{ (ルール 2)} \\ &= a^b \text{ (ルール 1')} \\ &= a \uparrow b \uparrow 1 \end{aligned}$$

よって、ルール 1 が成り立つ。

[15] ニコニコ動画 - 巨大数動画シリーズ http://www.nicovideo.jp/mylist/35451262

(2) c においてルール 1 が成り立つと仮定して、$c+1$ でルール 1 が成立することを示す。

$$\begin{aligned} a \to b \to (c+1) &= a \to [a \to (b-1) \to (c+1)] \to c \text{ (ルール 4)} \\ &= a \uparrow^c [a \to (b-1) \to (c+1)] \text{ (ルール 1)} \end{aligned}$$

ここで、今度は $c+1$ を固定して b に関する帰納法が必要となる。今、計算しているのは $a \to b \to (c+1)$ なので、$a \to (b-1) \to (c+1)$ についてはルール 1 が成立しているとする。なお、$b=1$ の時は、ルール 1 の両辺が a となるので、成り立っている。

$$\begin{aligned} a \to b \to (c+1) &= a \uparrow^c [a \uparrow^{c+1} (b-1)] \text{ (上式の続き、ルール 1)} \\ &= a \uparrow^c [\underbrace{a \uparrow^c a \uparrow^c \ldots \uparrow^c a}_{b-1}] \text{ (矢印表記の定義)} \\ &= \underbrace{a \uparrow^c a \uparrow^c \ldots \uparrow^c a}_{b} \\ &= a \uparrow^{c+1} b \text{ (矢印表記の定義)} \end{aligned}$$

以上で、ルール 1 が成り立つことが示された。 □

このチェーンは、とても増加率の高い関数です。まず 3 つの変数の場合は、

$$\begin{aligned} 10 \to 3 \to 2 &= 10 \uparrow\uparrow 3 = 10 \uparrow 10 \uparrow 10 > \text{グーゴル} \\ 10 \to 5 \to 2 &= 10 \uparrow\uparrow 5 = 10 \uparrow 10 \uparrow 10 \uparrow 10 \uparrow 10 \\ 3 \to 3 \to 3 &= 3 \uparrow\uparrow\uparrow 3 = \text{トリトリ} \end{aligned}$$

となります。クヌースの矢印表記で書くことのできる原始再帰程度の数は、3 変数チェーンで表記できる程度の大きさとなります。

モーザー数をチェーン表記で近似します。メガ $m(x,2)$ は↑が $x-2$ 個連なる効果と同じなので、$3 \to 3 \to x-2$ 程度の大きさになります。この見積もりだと、メガは $3 \to 3 \to 3$、つまりトリトリ程度となって、若干大き過ぎる気もしますが、その程度のおおざっぱな見積もりです。$f(x) = 3 \to 3 \to x$ とすると $f(3) = 3 \to 3 \to 3 =$ トリトリなので、メガよりも $f(3)$ の方が大き

い数です。モーザー数の大きさを評価すると、以下のように近似できます。

$$\begin{aligned} 3\to 3\to 2\to 2 &= 3\to 3\to 27 \\ &< 3\to 3\to m(5,2)\approx \text{モーザー数} \\ &< 3\to 3\to f(3) \\ &< 3\to 3\to f(27) \\ &= 3\to 3\to 3\to 2 \text{ (以下の計算を参照)} \end{aligned}$$

グラハム数は、$f(x)=3\underbrace{\uparrow\ldots\uparrow}_{x\text{ 個の}\uparrow}3=3\to 3\to x$ としたときの $f^{64}(4)$ です。

$$\begin{aligned} 3\to 3\to 1\to 2 &= 3\to 3\to 1=f(1)=27 \\ 3\to 3\to 2\to 2 &= 3\to 3\to (3\to 3\to 1)=f^2(1)=f(27) \\ 3\to 3\to 3\to 2 &= 3\to 3\to (3\to 3\to 2\to 2)=f^3(1)=f^2(27) \\ 3\to 3\to n\to 2 &= f^n(1)=f^{n-1}(27) \\ 3\to 3\to 64\to 2 &= f^{64}(1)<f^{64}(4)=\text{グラハム数} \\ 3\to 3\to 65\to 2 &= f^{64}(27)>f^{64}(4)=\text{グラハム数} \end{aligned}$$

したがって

$$3\to 3\to 64\to 2<\text{グラハム数}<3\to 3\to 65\to 2$$

となります。また

$$3\to 3\to 3\to 3=3\to 3\to (3\to 3\to 2\to 3)\to 2>3\to 3\to 65\to 2$$

となります。コンウェイは『数の本』でチェーン表記の定義をしてから、$3\to 3\to 3\to 3$ がグラハム数よりも大きいと詳細な計算手順を示さずに書いています。したがって、チェーン表記は「グラハム数を超えるために作られた表記」とも言えるでしょう。

チェーン表記は 2 重再帰関数です。アッカーマン関数の式

$$A(x+1,y+1)=A(x,A(x+1,y))$$

について、$x = b, y = a$ として変数の順番を交換し、

$$A(a+1, b+1) = A(A(a, b+1), b)$$

さらに定数項 X を加えると

$$A(X, a+1, b+1) = A(X, A(a, b+1), b)$$

となってチェーンのルール 4

$$X \to (a+1) \to (b+1) = X \to (X \to a \to (b+1)) \to b$$

と一致することから、チェーンの漸化式はアッカーマンと同じ 2 重再帰の式です。すなわち $X \to a \to b$ の右端を 1 加える $X \to a \to b+1$ が原始再帰で、それを数え上げることが 2 重再帰となっています。したがってチェーン表記は 2 重再帰をたくさん繰り返す 2 重再帰関数である、ということになります。チェーンでは、矢印を 1 つ伸ばすこと、すなわち変数を 1 つ増やすことで、基本 2 重再帰操作を 1 回することになります。変数が n 個のチェーンでは、基本 2 重再帰操作が $n-2$ 回行われます。

第 4 章 多重再帰関数

原始再帰関数よりも強い 2 重再帰関数を定義できたように、2 重再帰関数よりも強い 3 重再帰関数、さらに強い 4 重再帰関数を作れば、もっと強い関数を定義することができるでしょう。本章では、多重再帰関数によって、関数をどんどん強めていきます。

まずは、原始再帰、2 重再帰の定義を拡張して、n 重の多重再帰関数を次のように定義します。

【定義】多重再帰

$n-1$ 重再帰操作を数え上げる操作を n 重再帰作用素（操作）とする。n 重の多重再帰関数とは、定義域と値域が非負整数である非負整数個の引数をとる関数で、引数に対し、ゼロ、後者、射影、合成、原始再帰、2 重再帰、3 重再帰、...、n 重再帰の作用素（操作）を有限回適用した関数である。ただし、その中で n 重再帰操作を少なくとも 1 回含むものとする。

4.1 多変数アッカーマン関数

巨大数探索スレッド[1] その 7 で、たろうによって、アッカーマン関数の拡張である多変数アッカーマン関数 [2] が以下のように定義されました。

[1] http://ja.googology.wikia.com/wiki/巨大数探索スレッド
[2] http://ja.googology.wikia.com/wiki/多変数アッカーマン関数

【定義】多変数アッカーマン関数

n 個の非負整数を引数として以下のように定義される関数 A を n 変数アッカーマン関数とする。

$$
\begin{aligned}
A(\square, a) &= a+1 \\
A(\mathrm{X}, b+1, 0, \square, a) &= A(\mathrm{X}, b, a, \square, a) \\
A(\mathrm{X}, b+1, 0) &= A(\mathrm{X}, b, 1) \\
A(\mathrm{X}, b+1, a+1) &= A(\mathrm{X}, b, A(\mathrm{X}, b+1, a))
\end{aligned}
$$

ただし、a, b : 0 以上の整数, $\square$: 0 個以上の 0, X : 0 個以上の 0 以上の整数

最後の 2 式により、右から 2 個目の変数は原始再帰、その数え上げが 2 重再帰となります。2 式目が多重再帰を定義する肝となります。$\square$に含まれる 0 の数を n とすると、b の項は右から $n+3$ 番目となります。$n=0$ の時は、 次に示す 3 変数アッカーマンの式と一致します。つまり右から 3 変数目の b を 1 つ増やす操作が、右から 2 変数目に変数 a を用いた 2 重再帰の操作となり、右から 3 変数目が 2 重再帰を数え上げる 3 重再帰となります。このようにして、右から $n+2$ 番目の項が $n+1$ 重再帰であり、右から $n+3$ 番目の b の項は、右から $n+2$ 番目の $n+1$ 重再帰を数え上げる (変数 a 回の操作をする) ことで $n+2$ 重再帰の操作となります。したがって n 変数アッカーマン関数の 1 番左の項は、$n-1$ 重再帰となり、それを変数に持つ n 変数アッカーマン関数は n 重再帰関数となります。

多変数アッカーマン関数の例として、3 重再帰の 3 変数アッカーマン関数を以下のように書くことができます。

【定義】3 変数アッカーマン関数

非負整数 x, y, z に対し、以下のように定義される関数 $A(x, y, z)$ を 3 変数アッカーマン関数とする。

$$\begin{aligned} A(0,0,z) &= z+1 \\ A(x+1,0,z) &= A(x,z,z) \\ A(x,y+1,0) &= A(x,y,1) \\ A(x,y+1,z+1) &= A(x,y,A(x,y+1,z)) \end{aligned}$$

ただし後述するように、これはアッカーマンオリジナルの 3 変数アッカーマン関数とは異なる。

この 3 変数アッカーマン関数が、2 重再帰を数え上げる 3 重再帰操作を含み、したがってどんな 2 重再帰関数よりも大きい 3 重再帰関数であることを示します。

2 変数アッカーマンの定義と見比べることにより、

$$A(0,y,z) = A(y,z)$$

が成り立ちます。また、$A_x(y,z) = A(x,y,z)$ とすると、

$$\begin{aligned} A_x(0,z) &= A_{x-1}(z,z) \\ A_x(y+1,0) &= A_x(y,1) \\ A_x(y+1,z+1) &= A_x(y,A_x(y+1,z)) \end{aligned}$$

となり、$A_{x-1}(z,z)$ から $A_x(z,z)$ を生成する操作は、2 重再帰操作となります。

したがって、$A(x,0,z) = A_x(0,z) = A_{x-1}(z,z)$ は、関数 $A(0,0,z) = z+1$ に対して 2 重再帰操作を x 回繰り返した関数であり、これは 2 重再帰操作を数え上げる 3 重再帰関数です。

$A(x,y,z)$ において、x を 1 つ増やすことは、2 重再帰操作をすることなので、チェーン表記においてチェーンを 1 つ伸ばすことに相当します。そのことを確かめるために、$(a+1) \to \cdots (a+1) \to (z+1) \to (y+1)$ と $a+1$ が x 個続くチェーン $f_a(x,y,z)$ について考えます。

チェーンの定義から

$$\begin{aligned} f_a(0,y,z) &= (z+1)^{y+1} \\ f_a(x+1,0,z) &= f_a(x,z,a) \\ f_a(x+1,y,0) &= f_a(x,0,a) \\ f_a(x,y+1,z+1) &= f_a(x,y,f_a(x,y+1,z)) \end{aligned}$$

となります。$f_a(x,y,z)$ と $A(x,y,z)$ を比べると、若干異なるものの、同程度の増加度の関数となることが分かります。したがってチェーンを 1 個伸ばす (変数の数を増やす) ことは、$A(x,y,z)$ において x を 1 つ増やすことに相当します。そしてチェーン長が変数化された $f_a(x,y,z)$ は $A(x,y,z)$ と同様に 3 重再帰です。

より正確には、$x=1, y>1$ または $x>1, y+z>0$ のとき、

$$A(x,y,z) < (x+1 \text{ 個の } 3 \text{ のチェーン}) \to (z+2) \to (y+1) < A(x,y,z+1)$$

が成立することが帰納法で証明されています[3]。たとえば $x=3, y=2, z=1$ とすると $3 \to 3 \to 3 \to 3 \to 3 \to 3$ は $A(3,2,1)$ よりも大きく $A(3,2,2)$ よりも小さいということになります。

4.2　多重再帰に見えてそうでない関数

2 重再帰も多重再帰の一種ですが、この節に限って 3 重再帰以上を多重再帰とします。チェーンの定義は一見すると多重再帰に見えますが、2 重再帰の章で示したように、2 重再帰操作を繰り返した 2 重再帰関数です。一方、前節で示したように、適切に n 変数アッカーマン関数を定義することで、n 重再帰関数を作ることができます。ここで n 重再帰関数はいかなる $n-1$ 重再帰関数よりも大きい、という明確な強弱関係を持ちます。

アッカーマン関数のアッカーマンによるオリジナルの定義は

$$\begin{aligned} A(x,y,0) &= x+y \\ A(x,0,1) &= 0 \end{aligned}$$

[3] アッカーマンチェーン定理 http://gyafun.jp/ln/ackchain.html

$$\begin{aligned} A(x,0,2) &= 1 \\ A(x,0,z) &= x \\ A(x,y,z) &= A(x,A(x,y-1,z),z-1) \end{aligned}$$

というもので、3 変数でした。これに対して、様々な 2 変数バージョンが考えられて、その中で Peter and Robinson が前章で紹介した形の 2 変数アッカーマン関数を作成しました。アッカーマンが考えたオリジナルの 3 変数アッカーマン関数は、前章の 2 変数アッカーマン関数と同じ程度の増加率を持つ 2 重再帰関数です。通常、3 変数のアッカーマン関数と言えばこのアッカーマンのオリジナルの定義を意味しますが、本書ではたろうが定義した多変数アッカーマン関数の定義にしたがった、3 重再帰のアッカーマン関数を 3 変数アッカーマン関数とします。

チェーン表記のように、一見、多重再帰のように見えてそうでない関数には、他にはたとえばこのような関数があります。

$$\begin{aligned} A(0,0,z) &= z+1 \\ A(0,y+1,0) &= A(0,y,1) \\ A(0,y+1,z+1) &= A(0,y,A(x,y+1,z)) \\ A(x+1,y+1,z+1) &= A(x,A(x+1,y,z+1),A(x+1,y,z+1)) \end{aligned}$$

この式は、次のように計算されます。

$$\begin{aligned} & A(x+1,y+1,z+1) \\ = \; & A(x,A(x+1,y,z+1),A(x+1,y,z+1)) \\ = \; & A(x-1,A(x,A(x+1,y,z+1)-1,A(x+1,y,z+1)), \\ & A(x,A(x+1,y,z+1)-1,A(x+1,y,z+1))) \\ = \; & ... \\ = \; & A(0,A2,A2) \ (A2 \text{ は } x,y,z \text{ の 2 重再帰関数}) \\ = \; & A(A2,A2) \end{aligned}$$

このように、$A(x,y,z)$ の計算は 2 重再帰です。3 項漸化式において、このように最初に x を減らして、次に y を減らすという計算方法では、2 重再

帰どまりです。一方、3 変数アッカーマン関数では、最初に y を減らして、y が 0 に到達してはじめて x を減らすことができます。そのために、x を 1 減らす計算が y を減らす計算を数え上げることができます。この計算順序の違いが、2 重再帰か 3 重再帰かの差を生んでいます。このことが多重再帰関数の強さを理解する上では重要です。

4.3　拡張チェーン表記

アメリカの IT 技術者ピーター・ハーフォード[4] (Peter Hurford) は、チェーン表記の矢印に下付き文字をつけて

$$a \to_c b = \underbrace{a \to_{c-1} a \to_{c-1} \cdots \to_{c-1} a \to_{c-1} a}_{b \text{ 個の} \to_{c-1}}$$

というルールを加えた拡張チェーン表記[5] (Peter Hurford's extensions of chained arrow notation) を考案して、2011 年にブログで公開しました。そのブログ記事は消えていますが、次のような定義でした。

【定義】拡張チェーン表記

a, b, c を正の整数、X を 1 つ以上の正の整数の $\to_c$ チェーン $a_1 \to_c a_2 \to_c ... \to_c a_n$ とする。拡張チェーン表記は、以下の 5 つの規則によって計算する。ここで→の下につく数はチェーンの種類をあらわし、1 通りでなければならない。

$$\begin{aligned} a \to_1 b &= a^b \\ a \to_c b &= \underbrace{a \to_{c-1} a \to_{c-1} \cdots \to_{c-1} a}_{b \text{ 個の} \to_{c-1}} \quad (c > 1) \\ X \to_c 1 &= X \end{aligned}$$

[4] http://peterhurford.com/
[5] 巨大数研究 Wiki - チェーン表記 http://ja.googology.wikia.com/wiki/チェーン表記

$$
\begin{aligned}
X \to_c 1 \to_c a &= X \\
X \to_c (a+1) \to_c (b+1) &= X \to_c (X \to_c a \to_c (b+1)) \to_c b
\end{aligned}
$$

すなわち 3 変数以上のときには、必ず同じ種類の→なのでそのままチェーン表記の規則で計算をして、2 変数になったときに、2 番目の規則によって→の下の数が 1 つ減ります。この拡張チェーン表記を使うと、

$$
\begin{aligned}
3 \to_2 3 &= 3 \to 3 \to 3 \to 3 \\
3 \to_2 4 &= 3 \to 3 \to 3 \to 3 \to 3 \\
3 \to_2 5 &= 3 \to 3 \to 3 \to 3 \to 3 \to 3
\end{aligned}
$$

のように、$3 \to_2 x$ が $x+1$ 変数のチェーン表記に相当する関数となります。したがって $3 \to_2 x$ は 3 重再帰です。

$\to_2$ のチェーンを伸ばすことは、チェーンを伸ばすという 2 重再帰操作に相当するため、$\to_2$ の多変数チェーンは 3 重再帰 1 回+2 重再帰たくさん、という関数になります。$3 \to_3 x$ は、$x+1$ 変数の $\to_2$ チェーンなので、3 重再帰 2 回分です。$3 \to_n x$ は、3 重再帰を $n-1$ 回繰り返した 3 重再帰関数で、$3 \to_x x$ は 4 重再帰関数となります。

4.4 ふぃっしゅ数

ここから著者が考案したふぃっしゅ数[6] (Fish number) の説明に入ります。はじめにことわっておきますが、ふぃっしゅ数の定義はかなりややこしくて、多くの人が「定義が理解できない」と悩んでいます。著者のオリジナリティはここにあるので本書では詳しく解説しますが、ふぃっしゅ数以外にも巨大数論の面白さはいろいろとありますので、ふぃっしゅ数の話を飛ばして次の章に進んでいただいてもいっこうに構いません。

[6] http://ja.googology.wikia.com/wiki/ふぃっしゅ数

4.4.1　ふぃっしゅ数バージョン 1

まずはふぃっしゅ数を考案した経緯から説明します。そもそものきっかけは、2ch 掲示板の巨大数探索スレッド[7] というスレッドでの、どれだけ大きな数を考えることができるか、というお遊びでした。グラハム数がとにかく巨大な数であるということから、グラハム数を使って以下の様な数が定義されました。

161 名前：１３２人目の素数さん ：02/06/20 22:25

>> 156　じゃあグラハム数でいってみよう

グラハム数の定義はご存知だと思うが 3 ↑↑↑↑ 3（これがどれだけ超巨大かはグラハム数スレ参照）の数だけ 3 と 3 の間に↑が挟まった数を 1 段階として、2 段階は 1 段階の数だけ 3 と 3 の間に↑がある数と繰り返した 63 段階目の数がグラハム数と定義されてる。

この前段階の数だけ↑が挟まる数が次の段階という 63 回の変換の 1 回をＧ変換と名付ける。

ＮＯ１　グラハム数回だけＧ変換した数回変換した数回変換した～この繰り返しを

ＮＯ２　グラハム数回だけＧ変換した数回変換した数回変換した～この繰り返しを

ＮＯ３　グラハム数回だけＧ変換した数回変換した数回変換した～この繰り返しを

ＮＯ４　グラハム数回だけＧ変換した数回変換した数回変換した～この繰り返しを

とやっていって、Ｎｏがグラハム数回まで到達したら終り

‥‥だけだと面白くないので

（中略）　　以上のように延々繰り返してＮＯの種類がグラハム数種類に到達した時の数

[7] http://ja.googology.wikia.com/wiki/巨大数探索スレッド

これに対して、このように大きな数を定義するプロセスを追うことで、グラハム数を定義の中で用いずに、さらにグラハム数よりも大きな数、上記の数よりも大きな数を定義しよう、という目的で、以下のような考察をしました (2ch 過去ログ[8] の 317-319 の発言)。

> これまでの書き込みで「いかにして大きな数を作るか」というプロセスを一般化すると、大きな数と増加の程度が大きい関数を生み出していくプロセスだと表現できる。
>
> たとえば、「m という数に $f(x)$ という変換を n 回繰り返す」という表現をするときに、$m, f(x), n$ に使えるのは今までに定義された数と関数のみ。そこで、数と関数を双方ともに帰納的に定義していくプロセスを追っていくことにする。

そこで、自然数 m と関数 $f(x)$ のペアから、自然数 n と関数 $g(x)$ のペアを生み出す変換（写像）を

$$S : [m, f(x)] \to [n, g(x)]$$

と表記することにすると、たとえば $3 \uparrow\uparrow\uparrow 3$ は、自然数 4 と、$f(x) = 3$（$\uparrow$ が x 個）3 から $f(4)$ とあらわされる。$3 \uparrow\uparrow\uparrow\uparrow 3$ 個だけ $\uparrow$ がはさまった数、は $f(f(4))$ である。したがって、これを 64 回繰り返した数は $f^{64}(4)$ となり、この操作は $g(x) = f^{64}(x)$ という関数を自然数 64 と関数 $f(x)$ から生み出す操作にほかならないため、

$$S : [m, f(x)] \to [f^m(m), f^m(x)]$$

と書くと、$m = 64, f(x)$ からグラハム数よりも大きい $f^{64}(64)$ という数が生み出される。

これまでのスレッドにかかれた数は、いろいろなタイプの S 変換を数回、$>> 161$ でもせいぜい 10 回程度行っているにすぎない。S 変換については、上記の S 変換よりも、Ackermann タイプの S 変換の方がより関数を増加させる。

[8] 2ch 過去ログ http://www.geocities.co.jp/Technopolis/9946/log/ln023.html

そこで、これから先は「いかにしてより大きな数、関数を生み出す S 変換を作り出すか（これを「より大きい S 変換」と呼ぶ」といった考察をする。

その第 1 段階として、Ackermann 関数にならい、

$$\begin{aligned} B(0,n) &= f(n) \\ B(m+1,0) &= B(m,1) \\ B(m+1,n+1) &= B(m,B(m+1,n)) \\ g(x) &= B(x,x) \end{aligned}$$

としたときに、

$$S : [m, f(x)] \to [g(m), g(x)]$$

とする。少なくとも、これ以上に大きい S 変換はこれまでにあらわれていない。したがって、たとえば $[3, f(x) = x + 1]$ にこの S 変換を 10 回ほど繰り返せば、ゆうに $>> 161$ を越える。

では、この S 変換をさらに大きくするにはどうすれば良いか。

それには、S 変換を $f(m)$ 回繰り返した変換を S2 変換とすれば良い。すなわち、$m, f(x), S$ からさらに大きな S2 変換を生み出すことができる。このプロセスを、

$$SS : [m, f(x), S] \to [n, g(x), S2]$$
$$\text{ただし } g(x) = S2[m, f(x)], n = g(m)$$

という SS 変換で記述することにする。

$[3, x + 1, S]$ に SS 変換を 1 回かけると、S 変換を 4 回くりかえす変換が得られ、さらにもう 1 回かけると、S 変換を大変な数繰り返した変換が得られるため、2 回くりかえしただけで、すでにこのスレッドに登場したいかなる有限の数よりも大きな数が得られる。

この発言の直後に、ふぃっしゅ数が定義されています。このように、数と関数から数と関数を生成する「S 変換」という概念を導入する事で、巨大数を生み出そうとしたのがふぃっしゅ数です。これがふぃっしゅ数の定義です[9] 。

【定義】ふぃっしゅ数バージョン 1

1. 自然数と関数のペアから、自然数と関数のペアへの写像 S（S 変換）を以下で定義する。

$$S(m, f(x)) = (g(m), g(x))$$

ただし $g(x)$ は以下で与えられる。

$$\begin{aligned} B(0, n) &= f(n) \\ B(m+1, 0) &= B(m, 1) \\ B(m+1, n+1) &= B(m, B(m+1, n)) \\ g(x) &= B(x, x) \end{aligned}$$

2. 自然数、関数、変換の組から同様の組を生み出す写像 SS（SS 変換）を以下で定義する。

$$SS(m, f, S) = (S^{f(m)}(m, f),\ S^{f(m)})$$

ここで右辺は ((自然数, 関数), 変換) の形をしているが、これを (自然数, 関数, 変換) の 3 つ組と同一視する。

3. 3 つ組 (m_0, f_0, S_0) を $m_0 = 3$, $f_0(x) = x + 1$, S_0 は S 変換とするとき、

$$SS^{63}(m_0, f_0, S_0)$$

の第 1 成分をふぃっしゅ数バージョン 1、第 2 成分をふぃっしゅ関数バージョン 1 と定義する。

[9] http://ja.googology.wikia.com/wiki/ふぃっしゅ数バージョン 1

ここから先、色々なバージョンのふぃっしゅ数が登場することになるため、このふぃっしゅ数を「ふぃっしゅ数バージョン 1」と名付けました。またふぃっしゅ数バージョン 1 を F_1、ふぃっしゅ関数バージョン 1 を $F_1(x)$ とします。一般にふぃっしゅ数バージョン N を F_N、ふぃっしゅ関数バージョン N を $F_N(x)$ と表記します。ここで、なぜ 63 という数が使われているかとよく聞かれますが、グラハム数は $3 \uparrow\uparrow\uparrow\uparrow 3$ を 0 段階目として「63 段階目」という説明がされていたことから、グラハム数をまねて 63 という数を使いました。

また当初は

$$\begin{aligned} S : [m, f(x)] &\rightarrow [g(m), g(x)] \\ SS : [m, f(x), S] &\rightarrow [n, g(x), S2] \\ S2 &= S^{f(m)} \\ S2 : [m, f(x)] &\rightarrow [n, g(x)] \end{aligned}$$

のように表記していましたが、ミカヅキモが表記を分かりやすく整理しました[10]。『寿司 虚空編』では、こちらの表記が使われています。どちらの表記でも同じ定義です。

その後ふぃっしゅ数の大きさが計算されました。同スレッドの 329,331,377-379 の計算内容を転記します。

$[3, f(x) = x + 1]$ に S 変換を 1 回すると、

$$\begin{aligned} B(0, n) &= n + 1 \\ B(m + 1, 0) &= B(m, 1) \\ B(m + 1, n + 1) &= B(m, B(m + 1, n)) \\ g(x) &= B(x, x) \end{aligned}$$

となるので、$B(m, n)$ はアッカーマン関数と一致し、$g(x) = A(x, x)$ となるため、

$$S : [3, x + 1] \rightarrow [A(3, 3), A(x, x)]$$

[10] http://ja.googology.wikia.com/wiki/ユーザーブログ:Mikadukim/SS 変換の明示的な表記について

となる。

$A(3,3) = 61$ なので[11]、S 変換 1 回ではまだたいした大きさにはならない。

S 変換の 2 回目。今度は、

$$\begin{aligned} B(0,n) &= A(n,n) \\ B(m+1,0) &= B(m,1) \\ B(m+1,n+1) &= B(m,B(m+1,n)) \\ g(x) &= B(x,x) \end{aligned}$$

となるが、この $g(x)$ 関数はとてつもない関数になる。

$$\begin{aligned} g(1) &= B(1,1) = B(0,B(1,0)) \\ &= B(0,B(0,1)) = B(0,A(1,1)) \\ &= B(0,3) = A(3,3) = 61 \\ g(2) &= B(2,2) = B(1,B(2,1)) \\ &= B(1,B(1,B(2,0))) = B(1,B(1,B(1,1))) \\ &= B(1,B(1,61)) \\ &= B(1,B(0,B(1,60))) \end{aligned}$$

このあたりで、すでに書き下すことが困難になってくる。

$$\begin{aligned} B(1,1) &= 61 \\ B(1,2) &= A(61,61) \\ B(1,3) &= A(A(61,61),A(61,61)) \end{aligned}$$

という調子で関数が増えていくので、$B(1,61)$ はとんでもない数。$g(2) = B(1,B(1,61))$ なので、$g(2)$ ですでにグラハム数を超えているように思う。

[11]『寿司　虚空編』第 2 話 http://comic.pixiv.net/viewer/stories/6995

$g(2)$ ですでにグラハム数を超えてしまい、さらに $g(x)$ は x が増えるにつれてものすごい勢いで増えるので、$g(61)$ の大きさは想像を絶する。S 変換 2 回目にして、$g(61)$ というとんでもない数が得られることになる。

S 変換の 3 回目は

$$\begin{aligned} B(0,n) &= g(n) \\ B(m+1,0) &= B(m,1) \\ B(m+1,n+1) &= B(m,B(m+1,n)) \\ gg(x) &= B(x,x) \end{aligned}$$

としたときの $gg(x)$ となるので、

$$\begin{aligned} gg(1) &= B(1,1) = B(0,B(1,0)) \\ &= B(0,B(0,1)) = B(0,g(1)) \\ &= B(0,61) = g(61) \\ gg(2) &= B(2,2) = B(1,B(2,1)) \\ &= B(1,B(1,B(2,0))) = B(1,B(1,B(1,1))) \\ &= B(1,g(61)) \\ B(1,1) &= g(61) \\ B(1,2) &= g(g(61)) \\ B(1,3) &= g(g(g(61))) \end{aligned}$$

つまり、$gg(2)$ は 61 を $g(x)$ に代入して…と $g(61)$ 回繰り返した数。この調子で $gg(3), gg(4)$... と増えていき、$gg(g(61))$ が S 変換を 3 回繰り返した数。

SS 変換 2 回目は、SS 変換 1 回によって得られた $m, f(x), S$ に対して $S^{f(m)}$ とする、すなわち S 変換（つまり最初の S 変換を 4 回繰り返す変換）を $f(m)$ 回繰り返す。ここで、$f(m)$ 回と

はSS変換1回、つまりS変換4回によって得られる大きな数mを、これまたS変換4回によって得られる増加率の大きな関数$f(x)$に代入した数なので、とてつもなく大きな数。その数だけ、新S変換を繰り返す、ということ。

つまり、大きな数と関数から大きな数と関数を生み出し、その生み出された大きな数と関数から大きなS変換を生み出し、大きなS変換がさらにとてつもなく大きな数と関数を生み出す、とお互いがお互いを増幅させていく。

この計算に対して、巨大数探索スレッドでは以下の様な反応となりました。

380 名前：１３２人目の素数さん ：02/07/02 19:47

うぎゃーーーーーーーーーーーーーっ！！！
でっ、でけえ！！！
フィッシュ数は文句なし世界一、宇宙一の数だ！！！！！！
グラハム数とフィッシュ数を比べると、グラハム数は
限りなく0に近い
お疲れ様でした、

386 名前：１３２人目の素数さん ：02/07/02 20:25

ていうか
現実にあるものを文字にするだけの数字とかより
フィッシュ数のこと考えろ！
もうすごいから。すごいしヤバイから。

388 名前：１３２人目の素数さん ：02/07/02 20:32

小・中学生の頃、講談社のブルーバックスの宇宙や物理の本で
全宇宙のニュートリノの数とか光子の数が10＾88とかいう数字を見て
ぶったまげてたのが、なんかカワイク思えるよ

単に「とても大きな数を定義した」という以上の何者でもないのですが、これがこのように感銘を呼ぶところが巨大数の不思議な魅力です。

巨大数探索スレッドでは、このふぃっしゅ数を具体的に計算しようとして、それがいかに巨大な数になるか、といったことで盛り上がりを見せました[12] 。それが巨大数探求の道へのスタートでした。

Code Ass (@aycabta) [13] が、ふぃっしゅ数バージョン 1 の Ruby による計算プログラム[14] を作成しました。ふぃっしゅ数は大きすぎるので、メモリや時間の制限で実際には計算できませんが、プログラムを読むことで計算の手順を確認することができます。Code Ass は、他にもアッカーマン関数[15] と矢印表記[16] とチェーン表記[17] の計算プログラムを作成しました。

ふぃっしゅ数 F_1 の S 変換は、関数 $f(x)$ に 2 重再帰の操作をして関数 $g(x)$ を生成する操作です。F_1 は関数に 2 重再帰の操作を数多く繰り返すことによって、より大きな関数を生成するしくみである、と表現できます。したがって 2 重再帰を 2 回施した時点で、2 重再帰を 1 回に原始再帰を繰り返したグラハム数と比べて比較にならないほど大きな関数が生成されることとなります。ただしこの議論は厳密ではありません。なぜならば 2 重再帰の操作は何度繰り返しても 2 重再帰のままなので、2 重再帰の回数では生成される関数の大小を比較できないからです。ここではグラハム数生成に使われている操作も F_1 の S 変換も同等のアッカーマン的な 2 重再帰操作であるため、このような比較が可能となっています。

F_1 の定義において、$f(x)$ から $g(x)$ を生成する S 変換の漸化式は、$f(z) = A_x(z,z)$、$g(z) = A_x(z,z)$ とすると、3 変数アッカーマンの式と一致します。このことから、3 変数アッカーマンにおける 2 重再帰操作と F_1 の定義における 2 重再帰操作は、同じであることが分かります。F_1 の定義において、$[3, x+1]$ に S 変換を i 回繰り返して得られた関数を $S_i(x)$ とすると、S 変

[12] 2ch 過去ログ http://www.geocities.co.jp/Technopolis/9946/log/ln024.html
[13] Twitter @aycabta https://twitter.com/aycabta
[14] https://github.com/aycabta/fish-number
[15] https://github.com/aycabta/ackermann
[16] https://github.com/aycabta/arrow-notation
[17] https://github.com/aycabta/chained-arrow-notation

換の式は

$$\begin{aligned} S_{i+1}(0,n) &= S_i(n,n) \\ S_i(m+1,0) &= S_i(m,1) \\ S_i(m+1,n+1) &= S_i(m,S_i(m+1,n)) \end{aligned}$$

と書くことができ、3 変数アッカーマンの式と完全に一致します。したがって、F_1 の S 変換 i 回後の計算において、$B(m,n) = A(i,m.n)$ となります。S 変換を 1 回繰り返した後の $g(2)$ がグラハム数を超えるという計算は、$A(1,2,2)$ がグラハム数よりも大きいことを計算したことになります。実際に

$$\begin{aligned} A(1,1,0) &= A(1,0,1) = A(1,1) = 3 \\ A(1,1,1) &= A(1,0,3) = A(3,3) = 61 \\ A(1,1,2) &= A(1,0,61) = A(61,61) > 3 \to 3 \to 2 \to 2+2 \\ A(1,1,3) &> A(1,0,f^2(1)+2) > 3 \to 3 \to 3 \to 2+2 \\ A(1,1,x) &> 3 \to 3 \to x \to 2 \\ A(1,1,65) &> 3 \to 3 \to 65 \to 2 > \text{グラハム数} \end{aligned}$$

と、$A(1,1,65)$ でグラハム数を超えます。

$$\begin{aligned} A(1,2,0) &= A(1,1,1) = 61 \\ A(1,2,1) &= A(1,1,61) > 3 \to 3 \to 61 \to 2 \\ A(1,2,2) &> A(1,1,3 \to 3 \to 61 \to 2) \\ &> 3 \to 3 \to (3 \to 3 \to 61 \to 2) \to 2 \end{aligned}$$

となるため、$A(1,2,2)$ はグラハム数よりもはるかに大きな数です。

次に F_1 を 4 変数アッカーマン関数と比較します。まずは 4 変数アッカーマンを小さい方から順番に計算してみます。

$$\begin{aligned} A(1,0,1,0) &= A(1,0,0,1) = A(1,0,1) = A(1,1) = 3 \\ A(1,0,1,1) &= A(1,0,0,(A(1,0,1,0)) \end{aligned}$$

$$= \quad A(1,0,0,3) = A(3,0,3) = A(2,3,3)$$

よって、$A(1,0,1,1)$ は 2 重再帰です。

$$\begin{aligned} A(1,0,1,2) &= A(1,0,0,A(1,0,1,1)) \\ &= A(1,0,0,A(2,3,3)) \\ &= (A(2,3,3),0,(2,3,3)) \\ A(1,0,1,3) &= A(1,0,0,A(1,0,1,2)) \\ &= A(A(1,0,1,2),0,A(1,0,1,2)) \\ A(1,0,1,n+1) &= A(1,0,0,A(1,0,1,n)) \\ &= A(A(1,0,1,n),0,A(1,0,1,n)) \end{aligned}$$

$A(1,0,1,2)$ は、2 重再帰を $A(2,3,3)$ 回繰り返した数です。F_1 の定義では、SS 変換が S 変換で生じた 2 重再帰の数の回数だけ 2 重再帰を繰り返しているので、これに相当します。モーザー数では、原始再帰を原始再帰の回数 (メガ回) 繰り返すことで 2 重再帰の数ができました。それと同じことで、2 重再帰を 2 重再帰の回数繰り返しているので、ここから 3 重再帰に入ると考えます。

SS 変換の回数を重ねるごとに、その回数だけ S 変換を繰り返すので、SS 変換は $A(1,0,1,n)$ の n を 1 増やすことに相当します。実際には、SS 変換の指数部が m ではなくて $f(m)$ になっているので、もっと複雑に見えますが、m から $f(m)$ へと増やすことは原始再帰なので、2 重再帰から見ると無視できてしまいます。

SS 変換の 1 回目は S 変換 4 回なので、2 重再帰を 2 回繰り返している $A(1,0,1,1)$ よりは大きく、その回数以上 S 変換を繰り返した SS 変換の 2 回目は、$A(1,0,1,2)$ よりは大きく、…といったことで、SS 変換を 63 回繰り返した F_1 は、$A(1,0,1,63)$ よりは大きいと分かります。

さらに計算を続けます。

$$\begin{aligned} A(1,0,2,0) &= A(1,0,1,1) = A(2,3,3) \\ A(1,0,2,1) &= A(1,0,1,(A(1,0,2,0))) \end{aligned}$$

$$= \quad A(1,0,1,(A(2,3,3)))$$

$A(1,0,2,1)$ は、S 変換を繰り返す SS 変換を $A(2,3,3)$ 繰り返した数です。これは、F_1 を超えています。SSS 変換を定義すれば $A(1,0,2,1)$ になり、SSSS 変換は $A(1,0,3,1)$ になり…といった感じになることでしょう。

さらに大きい $A(1,1,1,1)$ を計算してみます。

$$\begin{aligned} A(1,1,1,1) &= A(1,1,0,A(1,1,1,0)) \\ &= A(1,1,0,A(1,1,0,1)) \\ &= A(1,1,0,A(1,0,1,1)) \\ &= A(1,1,0,A(2,3,3)) \\ &= A(1,0,A(2,3,3),A(2,3,3)) \end{aligned}$$

これは、SSSS…と、S の文字数を $A(2,3,3)$ 回繰り返した変換によって作られる数なので、F_1 の定義をいくら拡張しても追いつかない数です。

以上の計算から、

$$A(1,0,1,63) < F_1 < A(1,0,2,1) < A(1,1,1,1)$$

が F_1 の大きさになります。

巨大数探索スレッドでは、S 変換がチェーンを 1 つ延長する効果を持つことが、比較的早い段階で検証されていました[18] 。

87 名前：名無しのような物体 ◆ W7plq.175s ：02/10/05 15:41

> さて、前スレにも書きましたが、$3 \to a-2 \to b <$
> $2 \to a \to b < 3 \to a-1 \to b$ であることが確認できました。一応帰納法で証明しましたが読みたいですか？
> さらに、これを用いて
>
> $$\begin{aligned} B(x,y) &= (2 \to (y+3) \to (x-2)) - 3 \\ &\approx 3 \to (y+1 \sim 2) \to (x-2) \end{aligned}$$

[18] 巨大数探索スレッド 3 の過去ログ http://www.geocities.co.jp/Technopolis/9946/log/ln032.html

$$
\begin{aligned}
C(x,y) &\approx 3 \to 3 \to (y+1\text{〜}2) \to (x+1) \\
D(x,y) &\approx 3 \to 3 \to 3 \to (y+1\text{〜}2) \to (x+1)
\end{aligned}
$$

まで計算できました。どうやら S 変換 1 回で $3 \to$ がひとつ延長されるようです。これでいよいよふぃっしゅ数が (近似的に) 求められるのか？

Aeton は、ふぃっしゅ数バージョン 1 を拡張チェーン表記でこのように近似しました[19] [20]。

$$5(\to 2)64(\to 2)2 < F_1 < 6(\to 2)64(\to 2)2$$

4.4.2　ふぃっしゅ数バージョン 2

ふぃっしゅ数バージョン **2** [21] は以下のように定義されます[22]。

【定義】ふぃっしゅ数バージョン **2**

1. 自然数と関数のペアから、自然数と関数のペアへの写像 S（S 変換）を以下で定義する。

$$S(m, f(x)) = (g(m), g(x))$$

ただし $g(x)$ は以下で与えられる。

$$
\begin{aligned}
B(0,n) &= f(n) \\
B(m+1,0) &= B(m,1) \\
B(m+1,n+1) &= B(m,B(m+1,n)) \\
g(x) &= B(x,x)
\end{aligned}
$$

関数のみに着目して $Sf(x) = g(x)$ と書くこともできる。

[19] Togetter: ふぃっしゅ数の計算 http://togetter.com/li/568230
[20] Twitter @aetonal (2015 年 9 月 14 日) https://twitter.com/aetonal/status/643407430471282688
[21] http://ja.googology.wikia.com/wiki/ふぃっしゅ数バージョン 2
[22] 当初の定義と微妙に異なる　https://twitter.com/mikadukim_math/status/887305561212993537

2. 変換 S に対して、新たな変換 S^* を次で定義する。

$$(S^*f)(x) = (S^x f)(x)$$

自然数、関数、変換の組から同様の組を生み出す写像 SS（SS 変換）を以下で定義する。

$$SS(m, f, S) = ((S^{f(m)}f)(m),\ (S^{f(m)})^* f,\ S^{f(m)})$$

3. 3 つ組 (m_0, f_0, S_0) を $m_0 = 3$, $f_0(x) = x + 1$, S_0 は S 変換とするとき、

$$SS^{63}(m_0, f_0, S_0)$$

の第 1 成分をふぃっしゅ数バージョン 2 (F_2)、第 2 成分をふぃっしゅ関数バージョン 2 ($F_2(x)$) と定義する。

F_2 の定義はややこしく、F_3 の定義の方がすっきりしているので、F_3 で理解する方が良いと思います。F_2 は、F_3 が生み出される前のあまり洗練されていない定義です。

F_2 の定義は F_1 と似ていますが、SS 変換の定義で変換 S^* を用いて S 変換を数え上げています。すなわち、

$$(S^*f)(x) = (S^x f)(x)$$

によって生成される関数 $(S^*f)(x)$ は、
$(S^*f)(1)$ は $f(x)$ に S 変換を 1 回の $Sf(x)$ に 1 を代入した $Sf(1)$
$(S^*f)(2)$ は $f(x)$ に S 変換を 2 回の $S^2f(x)$ に 2 を代入した $S^2f(2)$
$(S^*f)(3)$ は $f(x)$ に S 変換を 3 回の $S^3f(x)$ に 3 を代入した $S^3f(3)$
$(S^*f)(n)$ は $f(x)$ に S 変換を n 回の $S^nf(x)$ に n を代入した $S^nf(n)$
というような関数です。この関数 $(S^*f)(x)$ は、関数 $f(x)$ に S 変換を有限回 (N 回) 施したいかなる関数よりも、$x > N$ において大きくなります。これが、つまり 2 重再帰である S 変換の操作を数え上げることによって、いかなる 2 重再帰よりも強い 3 重再帰の操作とした SS 変換を定義した、ということです。$F_2(x)$ は、3 重再帰操作を 63 回施した 3 重再帰関数です。

SS 変換を a 回、S 変換を b 回したときにできる関数 $g_{a,b}(n) = B_{a,b}(0,n)$ は、$A(a,b,0,n)$ のオーダーになることを、帰納的に示します。まず $a=0, b=0$ のときは $B_{0,0}(0,n) = n+1 = A(0,0,0,n)$ が成り立っています。そして $B_{a,b}(m,n) = A(a,b,0,n)$ が成り立っているとして、そこに S 変換を 1 回することで、関数 $A(a,b+1,0,n)$ に変換されることを示します。このときの S 変換の式は、

$$\begin{aligned} B_{a,b}(0,n) &= A(a,b,0,n) \\ B_{a,b}(m+1,0) &= B_{a,b}(m,1) \\ B_{a,b}(m+1,n+1) &= B_{a,b}(m,B(m+1,n)) \\ B_{a,b+1}(0,n) &= B_{a,b}(n,n) \end{aligned}$$

となり、$B_{a,b}(m,n) = A(a,b,m,n)$ と書き換えると、最初の式は

$$A(a,b,0,n) = A(a,b,0,n)$$

の恒等式となり、残りの式は

$$\begin{aligned} A(a,b,m+1,0) &= A(a,b,m,1) \\ A(a,b,m+1,n+1) &= A(a,b,m,A(a,b,m+1,n)) \\ A(a,b+1,0,n) &= A(a,b,n,n) \end{aligned}$$

となって、この漸化式は多変数アッカーマン関数の定義と一致します。よって、S 変換によって $A(a,b,0,n)$ が $A(a,b+1,0,n)$ に変換されることが示されました。

次に SS 変換については、SS 変換を a 回した $B_{a,0}(0,n)$ という関数に対して、SS 変換をもう 1 回する $B_{a+1,0}(0,n)$ は、S 変換を n 回する $B_{a,n}(0,n)$ と等しいため（実際には $f(m)$ 回繰り返す操作が入っているためもう少し大きいのですが、ここでは大体の大きさ（オーダー）を計算するため、n 回とします）、

$$B_{a+1,0}(0,n) = B_{a,n}(0,n)$$

となり、これはアッカーマン関数の式

$$A(a+1,0,0,n) = A(a,n,0,n)$$

と等しいため、帰納的に $g_{a,b}(n) = B_{a,b}(0,n) \approx A(a,b,0,n)$ が示されました。

SS 変換を 63 回繰り返すことで、$A(63,0,0,n)$ が得られます。よって、得られる数は $A(1,0,0,0,63) = A(63,0,0,63)$ のオーダーとなります。

4.4.3 ふぃっしゅ数バージョン 3

ふぃっしゅ数バージョン **3**(F_3) [23] [24] は、以下のように定義されます。

【定義】ふぃっしゅ数バージョン **3**

1. 関数 $f(x)$ から $g(x)$ への写像 $s(n)$ $(n > 0)$ を以下のように定める。

$$\begin{aligned} s(1)f &:= g; g(x) = f^x(x)[\text{原始再帰}] \\ s(n)f &:= g; g(x) = [s(n-1)^x]f(x)(n > 1)[n \text{ 重再帰}] \end{aligned}$$

2. 関数 $f(x)$ から $g(x)$ への写像 $ss(n)$ $(n > 0)$ を以下のように定める。

$$\begin{aligned} ss(1)f &:= g; g(x) = s(x)f(x)[\text{再帰回数の数え上げ}] \\ ss(n)f &:= g; g(x) = [ss(n-1)^x]f(x)(n > 1)[\text{さらに数え上げ}] \end{aligned}$$

3. ふぃっしゅ関数 $F_3(x)$ を以下のように定める。

$$F_3(x) := ss(2)^{63}f; f(x) = x + 1$$

4. ふぃっしゅ数 $F_3 := F_3^{63}(3)$ とする。

ミカヅキモは、この F_3 の定義を次のように書き換えました[25]。定義は変わらないので、理解しやすい方で理解するのが良いと思います。

[23] http://ja.googology.wikia.com/wiki/ふぃっしゅ数バージョン 3

[24] 巨大数研究 Wiki - ユーザー:Kyodaisuu/F3 の展開 http://ja.googology.wikia.com/wiki/ユーザー:Kyodaisuu/F3 の展開

[25] http://ja.googology.wikia.com/wiki/ユーザーブログ:Mikadukim/ふぃっしゅ数バージョン 3 の書き換え

自然数全体から自然数全体への写像を関数と呼ぶ。関数全体から関数全体への写像を変換と呼ぶ。関数 f に対して、関数 f^* を

$$f^*(x) = f^x(x)$$

で定義し、変換 T に対して、変換 T^* を

$$(T^*f)(x) = (T^xf)(x)$$

で定義する。$s(n), ss(n), F_3$ を、次のように定義する。

$$\begin{aligned} s(1)f &= f^* \\ s(n) &= s(n-1)^* \quad (n \geq 2) \\ (ss(1)f)(x) &= (s(x)f)(x) \\ ss(n) &= ss(n-1)^* \quad (n \geq 2) \\ F_3 &= (ss(2)^{63}f_0)^{63}(3), \quad f_0(x) = x+1 \end{aligned}$$

この定義は、F_1 や F_2 と比べると異なった表現が用いられていますが、内容は F_2 の拡張です。また、「巨大数探索スレッド」で定義された F_3(旧 F_3)の定義とも、異なっています。旧 F_3 の定義の中で、最も本質的な部分を記述したものが新 F_3 です。旧 F_3 の定義については、煩雑になるためここには記しません。

そこで、F_1,F_2, 旧 F_3, 新 F_3 の定義の変遷について表 4.1 にまとめます。

つまり、F_1 から旧 F_3 まで、定義を拡張することでより高い再帰程度を持つ関数、そして巨大数を作ってきましたが、その結果、旧 F_3 の定義は非常に複雑なものになりました。新 F_3 では、旧 F_3 の定義の中から、高い再帰程度を持つ関数を作るために必要な定義のみを残して簡略化しました。旧 F_3 と新 F_3 の大きさは厳密には異なりますが、関数の再帰程度が同程度になります。

ここで、新 F_3 の定義の 2 つの簡略化について説明します。まずは、1 つ目の簡略化についてです。SS 変換は「数と関数と S 変換」から「数と関数と S 変換」への写像であり、S 変換の回数を数え上げることで、大きな関数を作っています。S 変換そのものは、S 変換の回数を増やすことで大きくし

表 4.1: F_1 から F_3 までの定義の変遷

バージョン	定義
F_1	2 重再帰操作である S 変換 S 変換を多数回繰り返す操作 (SS 変換)
F_2	S 変換 (2 重再帰) の回数を数え上げる SS 変換 (3 重再帰) (SS 変換は数、関数、変換から数、関数、変換への写像)
旧 F_3	F_2 の S 変換,SS 変換,... に相当する $s(n)$ 変換 (多重再帰) 再帰回数を数え上げる $ss(1)$ 変換
新 F_3	F_3 の $s(n)$ 変換の定義を関数から関数への写像へと簡略化 $s(1)$ 変換の定義を 2 重再帰から原始再帰へと簡略化

ていますが、そのこと自体は関数を強くすることに本質的に役立っていません。したがって SS 変換の定義の中で、S 変換を大きくするしくみは「無駄なしくみ」ということになります。また、数を関数に代入する定義についても、関数さえ定義されれば最終的にできた関数に数を代入すればいいのですから、「無駄な定義」ということになります。このように、SS 変換の定義は「関数から関数への写像」の定義の部分が本質的ということになります。そこで、$s(n)$ 変換は「関数から関数への写像」を定義としました。

次に、新 F_3 の 2 つ目の簡略化について説明します。これまで、S 変換はアッカーマン関数を元にした 2 重再帰操作で定義されていました。しかし、2 重再帰は原始再帰操作の数え上げで定義できますから、原始再帰からスタートすれば十分ということになります。そうすれば、原始再帰の数え上げから 2 重再帰、その数え上げから 3 重再帰と、多重再帰を定義できます。そこで新 F_3 の $s(1)$ は原始再帰を定義としました。新 F_3 の $s(2)$、すなわち

$$\begin{aligned} s(1)f &:= g; g(x) = f^x(x) \\ s(2)f &:= g; g(x) = [s(1)^x]f(x) \end{aligned}$$

から計算される $s(2)$ は、F_1 や F_2 の S 変換とは異なりますが、同じ程度の大きさの 2 重再帰操作となります。$s(1)f := g;\ g(x) = f^{x+1}(1)$ とすれば、

$s(2)$ は F_1 や F_2 の S 変換と一致します。

F_3 の $s(n)$ と多変数アッカーマン関数の大きさは、以下のように比較できます。

$f(x) = x + 1$ とすると

$$[s(1)^{a_1}][s(2)^{a_2}]...[s(n)^{a_n}]f(x) \approx A(a_n, ..., a_2, a_1, x)$$

ここで $s(1)$ の定義として $s(1)f := g; g(x) = f^{x+1}(1)$ を採用すると両辺がイコールになります。このことは F_3 の $s(n)$ と多変数アッカーマン関数の漸化式を比較すると両者が一致することで示すことができます。

つまり、多重再帰の計算手順を定義したものが多変数アッカーマン関数、多重再帰における「操作の数え上げ」を「関数から関数への写像」で表現したものが F_3 の $s(n)$ であり、両者は等価である、ということになります。

$ss(1)$ は、$g(x) = s(x)f(x)$ によって関数 $f(x)$ に対する x 重再帰を定義しています。ここから先の大きさの評価は、多変数アッカーマン関数や配列表記では間に合わないので、次の章で「急増加関数」を使って評価します。

4.5　バード数から配列表記へ

数学の修士号を持っているイギリス人のクリス・バード (Chris Bird) [26] がホームページ uglypc.ggh.org.uk 上で公開していたバードの矢印回転表記とバード数について、巨大数探索スレッドでは 2002 年から色々と議論がされてきました。

まず矢印回転表記については、拡張チェーン表記のように矢印に添え字をつけることを、矢印を回転することで表記し、矢印を n 回転する $\uparrow n$ という記号を導入しています。したがって拡張チェーン表記と同様に、矢印を伸ばす操作は 2 重再帰操作で、矢印を回転させることが 2 重再帰操作を数え上げる 3 重再帰操作となります。そして拡張チェーン表記で $x \rightarrow_x x$ が 4 重再帰関数となることと同様に、矢印を回転させる回数を数える矢印

[26] 巨大数研究 Wiki - Chris Bird http://ja.googology.wikia.com/wiki/Chris_Bird

回転関数が 4 重再帰関数となります。拡張チェーン表記とバードの矢印回転表記は、細かい定義の違いはあってもほぼ同じ関数であり同等の増加速度です。

バード数は、この 4 重再帰の矢印回転関数に 2 重再帰、原始再帰、合成を組み合わせた 4 重再帰関数で、多変数アッカーマン関数で $A(1,0,1,2,2)$ よりは小さい数です。

巨大数探索スレッドでは、ふぃっしゅ数に対してバード数が見つけられて、魚と鳥の対決だと盛り上がっていましたが、バードは日本の掲示板でそんなことで盛り上がっているとはまったく知らなかったことでしょう。そのバードのバード数のページはなくなり、巨大数の探求は終わっていたかに見えました。ところがバードは、海外のグーゴロジスト達といっしょにさらに巨大数の探求を進めていました。

バードはロバート・ムナフォのサーバーの上に、巨大数のページ[27] を開設し、古いバード数よりも大きな巨大数を作っていました。バードのホームページには、いくつかの段階を持って巨大数が定義されています。まず最初に、バードの線形表記 (Bird's linear notation) を考えています。この発想は、次に説明する BEAF にもつながります。

数学修士号を持つアメリカのアマチュア数学者、ジョナサン・バウアーズ (Jonathan Bowers) [28] [29] は、スビス・サイビアンに「現代巨大数論の父」と評価されています。バウアーズは、巨大数以外ではポリトープ、すなわち高次元に拡張された多面体に関する研究でも知られています。バウアーズは 2002 年に最初のバージョンの配列表記をホームページに公開しました。それがきっかけとなって、少しずつアマチュアの数学愛好家の中に巨大数に対する関心が強くなり、やがてバウアーズはバードやジョン・スペンサー (John Spencer) の助けを借りて、巨大数を表記する **BEAF** [30] (Bowers Exploding Array Function) という手法を考えて、2007 年に公開しました。これは非常に大きな巨大数を生み出すシステムで、高次元の多

[27] Chris Bird's Super Huge Numbers http://mrob.com/users/chrisb/
[28] http://ja.googology.wikia.com/wiki/Jonathan_Bowers
[29] Hedrondude's Home Page http://www.polytope.net/hedrondude/home.htm
[30] 巨大数研究 Wiki - BEAF http://ja.googology.wikia.com/wiki/BEAF

面体を研究しているバウアーズならではの突飛な発想があり、想像力を刺激させられるものです。このように、インターネットでアマチュアの数学家たちが巨大数の表記法やその強さを評価をするための理論について議論をするきっかけとなったのがバウアーズであるとして、サイビアンはバウアーズを「現代巨大数論の父」と評価しています。英語圏ではバウアーズの配列表記がきっかけとなり、日本語圏では巨大数探索スレッドがきっかけとなり、いずれも 2002 年頃からインターネットでアマチュア数学愛好家たちの巨大数探求が始まったことになります。いずれもグラハム数とチェーン表記を越えようとするところがとっかかりとなっているあたりも共通しています。

バウアーズはまた、非常に多くの面白い巨大数の名前をつけました。その中の 1 つが、本書で紹介したトリトリです。バウアーズが巨大数に様々な面白い名前をつけたことも、多くの人が巨大数に興味を持つきっかけとなりました。2016 年 2 月現在で、バウアーズが考えた巨大数の名前は 352 個あるとされています。そして巨大数を生み出す表記法をたくさん考えると同時に、多くの巨大数の名前をつけるという流儀は、他の人たちも真似をしました。たとえば、スビス・サイビアンは 2008 年末に公開が開始された Web 書籍でハイパー E 表記を出発とする拡張 E 表記システムを開発し、2015 年 3 月時点で 15610 個の巨大数の名前をつけたとされています。超階乗表記を考案したローレンス・ホロム (Lawrence Hollom) も、同様にその表記に対応した巨大数の命名規則を考案しています。アレックス・チャオクヒァオ (Aarex Tiaokhiao) は、いくつかの既存の巨大数表記法の簡単な拡張と、それに対する多くの巨大数の命名をしています。強配列表記を考案した Hyp cos も、同様に強配列表記に対応して多くの巨大数の名前をつけています。このように巨大数表記の体系を作って、同時に多くの巨大数の名前をつけるという人たちが多く出てきたことは、バウアーズの影響力の大きさを示しています。

BEAF ではまず**配列表記**[31] (array notation) を考えます。BEAF の構築にはバードも関わっていたこともあり、この BEAF の配列表記とバードの線形表記はまったく同じです。以下がその定義です。

[31]巨大数研究 Wiki - 配列表記 http://ja.googology.wikia.com/wiki/配列表記

【定義】BEAF の配列表記

配列は正の整数の有限の数列 $A = (a_1, a_2, \ldots, a_n)$ により定義される。配列表記はこの数列から 1 つの正の整数への写像、すなわち関数 $v(A) = \{a_1, a_2, \ldots, a_n\}$ であり、次のような規則によって計算される。

1. $\{a\} = a$, $\{a, b\} = a^b$
2. $\{a, b, c, \ldots, n, 1\} = \{a, b, c, \ldots, n\}$
3. $\{a, 1, b, c, \ldots, n\} = a$
4. 3 番目の要素が 1 の時

$$\begin{aligned} & \{a, b, \underbrace{1, \ldots, 1}_{x \text{ 個の } 1}, c, d, \ldots, n\} \\ = \ & \{\underbrace{a, a, \ldots, a}_{x+1 \text{ 個の } a}, \{a, b-1, \underbrace{1, \ldots, 1}_{x \text{ 個の } 1}, c, d, \ldots, n\}, c-1, d, \ldots, n\} \end{aligned}$$

   - その次の 1 でない要素の前にある全ての要素は最初の要素に置き換わり、
   - 上記のうち最後のものは、元々の配列の 2 番目の要素が 1 差し引かれたものに置き換わり、
   - 先述の 1 でない要素は 1 差し引かれる。

5. 規則 1〜4 のいずれの場合にも当てはまらない場合

$$\{a, b, c, d, \ldots, n\} = \{a, \{a, b-1, c, d, \ldots, n\}, c-1, d, \ldots, n\}$$

ためしに計算をしてみます。まずは 3 変数の場合です。

$$\begin{aligned} \{3,3,3\} &= \{3, \{3,2,3\}, 2\} \\ &= \{3, \{3, \{3,1,3\}, 2\}, 2\} \\ &= \{3, \{3,3,2\}, 2\} = \{3, \{3, \{3,2,2\}, 1\}, 2\} \\ &= \{3, \{3, \{3,2,2\}\}, 2\} \end{aligned}$$

$$
\begin{aligned}
&= \{3, 3^{\{3,2,2\}}, 2\} \\
&= \{3, 3^{\{3,\{3,1,2\},1\}}, 2\} \\
&= \{3, 3^{\{3,3\}}, 2\} = \{3, 3^{3^3}, 2\} \\
&= \{3, 3 \uparrow\uparrow 3, 2\} = \{3, \{3, 3 \uparrow\uparrow 3-1, 2\}\} \\
&= 3 \uparrow \{3, 3 \uparrow\uparrow 3-1, 2\} \\
&= 3 \uparrow 3 \uparrow \{3, 3 \uparrow\uparrow 3-2, 2\} \\
&= 3 \uparrow 3 \uparrow 3 \uparrow \{3, 3 \uparrow\uparrow 3-3, 2\} \\
&= ... = 3 \uparrow\uparrow 3 \uparrow\uparrow 3 \\
&= 3 \uparrow\uparrow\uparrow 3
\end{aligned}
$$

つまり $\{3,3,3\}$ はトリトリになります。というよりも、トリトリの定義が $\{3,3,3\}$ で、それは $3 \uparrow\uparrow\uparrow 3$ と等しいので、$3 \uparrow\uparrow\uparrow 3$ がトリトリであるという説明をしていました。一般に、3 変数の BEAF に対しては、

$$\{a,b,c\} = a \to b \to c = a \uparrow^c b$$

が成立します。

【証明】

c に関する帰納法で証明する。$c=1$ の時は $\{a,b\} = a \uparrow b$ となるため成立する。$c-1$ で成立するときに、c では

$$
\begin{aligned}
\{a,b,c\} &= \{a, \{a, b-1, c\}, c-1\} \\
&= a \uparrow^{c-1} \{a, b-1, c\} = a \uparrow^{c-1} \{a, \{a, b-2, c\}, c-1\} \\
&= a \uparrow^{c-1} a \uparrow^{c-1} \{a, b-2, c\} \\
&= ... = \underbrace{a \uparrow^{c-1} a \uparrow^{c-1} ... \uparrow^{c-1} a}_{b \text{ 個の } a} = a \uparrow^c b
\end{aligned}
$$

となり、成立する。 □

次に 4 変数の場合です。

$$\{3,65,1,2\} = \{3,3,\{3,64,1,2\},1\} = \{3,3,\{3,64,1,2\}\}$$

$$\begin{aligned} &= \{3,3,\{3,3,\{3,63,1,2\}\}\} \\ &= \{3,3,\{3,3,\{3,3,\{3,62,1,2\}\}\}\} \end{aligned}$$

これはおよそグラハム数になります

$$\{65,2,2,2\} = \{65,\{65,1,2,2\},1,2\} = \{65,65,1,2\}$$

も、だいたいグラハム数です。

配列表記と多変数アッカーマン関数を比較すると、両者には

1. 配列表記では 1 が最小の数だが、多変数アッカーマンでは 0 が最小の数となっている。
2. 数を並べる順番が左右逆になっている。配列表記では右の数の方が数を大きくする効果が大きく、多変数アッカーマンではその逆である。
3. 配列表記は $\{a,b\} = a^b$ の 2 変数関数が基本となり、多変数アッカーマンは $A(a) = a+1$ の 1 変数関数（後者関数）が基本となっている。
4. 配列表記では $\{a,b,1,\ldots,1,c,d,\ldots,n\} =$
   $\{a,a,a,\ldots,\{a,b-1,1,\ldots,1,c,d,\ldots,n\},c-1,d,\ldots,n\}$
   と、前の数が全部 a に変わる。多変数アッカーマンでは $A(X,b+1,0,\square,a) = A(X,b,a,\square,a)$ と、1 つ右の数だけ変わる。

といったような違いがありますが、再帰を重ねる構造は似ています。4 番目の点がだいぶ違うように見えますが、多変数アッカーマン関数でも

$$\begin{aligned} A(2,0,0,0,0,5) &= A(1,5,0,0,0,5) \\ &= A(1,4,5,0,0,5) \\ &= A(1,4,4,5,0,5) \\ &= A(1,4,4,4,5,5) \end{aligned}$$

のように、結局は 1 つずつ数が変わっていくので、本質的にはそれほど変わりません。

ここで、配列表記の要素がそれぞれどのような役割を持っているかを説明します。$f(m) = \{n+1, m, a_0, X\}(X$ は 0 個以上の要素) とすると

$$\begin{aligned}
\{n+1, 2, a_0+1, X\} &= \{n+1, \{n+1, 1, a_0+1, X\}, a_0, X\} \\
&= \{n+1, n+1, a_0, X\} = f(n+1) \\
\{n+1, 3, a_0+1, X\} &= \{n+1, \{n+1, 2, a_0+1, X\}, a_0, X\} \\
&= \{n+1, f(n+1), a_0, X\} \\
&= f^2(n+1) \\
\{n+1, 4, a_0+1, X\} &= \{n+1, \{n+1, 3, a_0+1, X\}, a_0, X\} \\
&= f^3(n+1) \\
\{n+1, m+1, a_0+1, X\} &= f^m(n+1)
\end{aligned}$$

となって、2 番目の要素 $m+1$ が関数 f の合成を繰り返す回数 m をあらわしています。そして

$$\begin{aligned}
\{n+1, 2, a_0+2, X\} &= \{n+1, n+1, a_0+1, X\} \\
&= f^n(n+1)
\end{aligned}$$

となり、$\{n+1, 2, a_0+1, X\} = f(n+1)$ から $\{n+1, 2, a_0+2, X\} = f^n(n+1)$ へと、3 番目の要素を 1 増やすことで、合成を数え上げる原始再帰 $f^n(n)$ の効果があります。したがって、3 番目の要素は原始再帰の回数となります。さらに

$$\begin{aligned}
& \{n+1, m+1, 1, a_1+1, X\} \\
= \ & \{n+1, n+1, \{n+1, m, 1, a_1+1, X\}, a_1, X\}
\end{aligned}$$

という式では、4 番目の要素を 1 減らすために、原始再帰を $\{n+1, m, 1, a_1+1, X\}$ 回の数え上げをしています。これが原始再帰を数え上げる 2 重再帰です。つまり 4 番目の要素は 2 重再帰の回数です。3 番目と 4 番目の要素が 1 になると、5 番目の要素が 1 減ることで 2 重再帰を数え上げる 3 重再帰となります。同様に考えていくと、$n+2$ 番目の要素が n 重再帰の回数をあらわします。このように多変数アッカーマン関数と左右の方向が逆の関係になります。

以上の関係をふまえて、配列表記と多変数アッカーマン関数の大きさを比較すると、$n>1$ に対して次のようになります。

$$\begin{aligned}\{n+1,n+1,n+1,n+1\}=\{n+1,2,1,1,2\} &\approx A(1,0,0,n)\\ \{\underbrace{n+1,...,n+1}_{5\text{ 個}}\}=\{n+1,2,1,1,1,2\} &\approx A(1,0,0,0,n)\\ \{n+1,2,a_0+1,a_1+1,...,a_k+1\} &\approx A(a_k,...,a_1,a_0,n)\end{aligned}$$

$f(n)=A(a_k,...,a_2,a_1,a_0,n)$ として

$$\{n+1,m+1,a_0+1,a_1+1,a_2+1,...,a_k+1\}\approx f^m(n)$$

ここで、配列表記では

$$\begin{aligned}\{1,a_0,a_1,...,a_k\} &= 1\\ \{2,a_0,a_1,...,a_k\} &= 4\ (k>1,a_0>1,a_k>1\text{ のとき})\end{aligned}$$

となります。つまり、4 変数以上の配列表記で一番左の要素が 2 のときは 4 になってしまいます。以下のように計算されます。

$$\begin{aligned}\{2,a_0,a_1,...,a_k\} &= \{2,\{2,a_0-1,a_1,...,a_k\},a_1-1,...,a_k\}\\ &= ...=\{2,X,1,...,a_k\}(\text{ここで }X\text{ はある数})\\ &= \{2,2,Y,...,a_k\}(\text{ここで }Y=\{2,X-1,1,...,a_k\})\\ &= \{2,\{2,1,Y,...,a_k\},Y-1,...,a_k\}\\ &= \{2,2,Y-1,a_2,...,a_k\}\\ &= \{2,2,Y-2,a_2,...,a_k\}\\ &= \{2,2,1,a_2,...,a_k\}\\ &= \{2,2,1,1,a_3,...,a_k\}\\ &= \{2,2,1,1,...,1\}\\ &= \{2,2\}=4\end{aligned}$$

つまり、3 番目の要素が 1 まで落ちた直後に配列表記の 4 番目のルールによって 2 番目の要素が 2 になってしまい、そうすると 3 番目の要素が減っ

ても値が変わらないため 3 番目の要素が 1 にまで落ちます。さらに 4 番目、5 番目の要素と 1 まで落ちていき、最終的には 4 となります。

したがって配列表記で巨大数を表記するときには、一番左の要素は 3 以上とします。

ちなみに Googology Wiki で開催された第 1 回国際巨大数オリンピック[32] [33] では

> チェーン表記と配列表記に関する次の不等式を満たすような 4 つの整数 (a, b, c, d) の組をすべて求めなさい。
>
> $$a \to b \to c \to d > \{a, b, c, d\}$$

という問題が出題されました。詳しく検討すると $a = 2, b > 2, c > 0, d > 1$ がその答えとなります。

このように、n 変数の配列表記は $n-1$ 変数アッカーマン関数と増加速度が対応し、$n-2$ 重再帰関数となります。4 変数の配列表記は 2 重再帰関数でチェーン表記と対応し、5 変数の配列表記は 3 重再帰で拡張チェーン表記に対応します。つまり配列表記（バードの線形表記と同じ）では 5 変数以上になるとチェーン表記を超えます。このことをバードが証明したものがバードの証明[34] [35] (Bird's proof) です。日本の巨大数論では、チェーン表記が 2 重再帰（3 変数アッカーマン）のレベルであることが分かったことで巨大数の理解が大きく進みましたが、海外の巨大数論では、このバードの証明によってチェーン表記が 4 変数配列表記（=4 変数のバードの線形表記）のレベルであることが分かったことが重要で、バードはチェーンの拡張である矢印回転表記を捨てて、バードの線形表記を基本として、さらに発展をさせて、バードの配列表記、そして新しいバード数 (p. 219) を定義しました。

---

[32] 第 1 回国際巨大数オリンピック http://ja.googology.wikia.com/wiki/スレッド:1793

[33] http://googology.wikia.com/wiki/User_blog:Wythagoras/Results_of_the_First_International_Googological_Olympiad

[34] 巨大数研究 Wiki - バードの証明 http://ja.googology.wikia.com/wiki/バードの証明

[35] 5 変数以上のバードの線形表記がチェーン表記よりも強い証明 http://www.mrob.com/users/chrisb/Proof.pdf

第 5 章 順序数

ここから先は、前章で紹介したような多重再帰関数よりも大きな関数、そして巨大数を探求する事になります。それでは、そのように大きな関数を、どうやって比較するのでしょうか。

たとえば原始再帰関数であれば矢印表記、2 重再帰関数であればチェーン表記、多重再帰関数であれば多変数アッカーマン関数によって比較、近似ができます。ここから先、多変数アッカーマン関数よりも強い関数を評価するためには、さらに強い関数を物差しに使えば良いのですが、そこで本章で導入する順序数がとても良い物差しになります。

巨大数論において、順序数を物差しとして関数の強さを評価する方法は、主に 2 つあります。

- 順序数階層を使う方法では、順序数から関数への写像を使って、関数を近似します。関数を強めていくときの階層的な構造が、順序数階層で表現されます。
- 順序数解析を使う方法では、関数の強さをその関数を定義する理論体系の強さと関連付けて評価します。理論体系の強さを順序数で表すために順序数解析を使います。

そして、順序数階層と順序数解析はお互いに関連もしています。

そこで、本章ではまず順序数、順序数階層、順序数解析について順次解説し、これまでに出てきた巨大数と関数の大きさを、順序数階層によって評価します。本章で順序数によって関数の強さを評価するための準備を整

えてから、次の章へと進みます。本書に書いた順序数に関する説明を、よりかみくだいて説明した「順序数講座」[1] も、ご参照ください。

5.1 順序数

集合論の創始者であるドイツの数学者、ゲオルク・カントール (Georg Cantor, 1845–1918) は、自然数を拡張した**順序数** (ordinal number) という概念を考えました。順番に並べられたものに対して、順序数で「番号」を振ることができます。ここで、その「順番に並べられたもの」が有限であれば「番号」は自然数で足りますが、無限にあるときにも番号を振ることができるようにしたものが順序数です。

カントールは、自然数の集合と実数の集合では、お互いの要素の間に一対一の対応づけをすることができない、つまり実数の集合は自然数の集合よりも本質的に「大きい」集合であることを示したことで有名です。そのことから、無限集合にも色々な「大きさ」があるということになり、無限集合の大きさをあらわす数をカントールは**超限数** (transfinite number) と名付けました。これは現代数学では基数に相当します。基数と順序数は違います。感覚的な表現では、基数が集合の「大きさ」を示すもので、順序数は「順序」を数えるものです。自然数では両者が一致しますが、無限集合では一致しないために明確に区別する必要があります。順序数と基数の違いについては、順序数を定義した後に改めて説明します。

5.1.1 整列集合と順序型

順序数は、次のように定義されます[2] [3]。

【定義】順序数

順序数とは、整列集合の順序型である。

[1] http://ja.googology.wikia.com/wiki/ユーザーブログ:Kyodaisuu/順序数講座

[2] MathWorld - Ordinal Number http://mathworld.wolfram.com/OrdinalNumber.html

[3] 他の定義方法もあるが、本書では MathWorld で採用されていて、簡明なこの定義を使って説明する。ZF 公理系では順序同型の集合全体を集合として取り扱うことができないという問題はあるものの、型理論や Quine の新基礎集合論ではこの定義も有効であると英語版 Wikipedia には記述されている。

そこで、まずは整列集合と順序型について定義します。

まずは**順序集合**[4] (ordered set) についてです。順序集合とは「順序」の概念が定義された集合です。順序が定義されると、集合の 2 つの要素を比較して、どっちが先でどっちが後かという順序を考えることができます。ここで、順序が決まるかもしれないし、決まらないかもしれません。そして必ず順序が決まる集合は**全順序集合** (totally ordered set) と呼ばれます。全順序集合に対して、順序が決まらないかもしれない一般の順序集合は**半順序集合** (partially ordered set) とも呼ばれます。全順序集合は半順序集合の一種です。

たとえば、自然数全体の集合、有理数全体の集合、実数全体の集合は、通常の大小関係によって全順序集合となります。実数全体の集合から 2 つの要素である実数 a, b を取り出せば、その大小関係（順序）が必ず決まるからです。

一方、集合 X の部分集合全体の集合は、包含関係 $\subset$ によって半順序集合になります。X の 2 つの部分集合は、片方がもう片方を包含することもあれば、お互いにお互いを包含していないこともあるからです。

このことを、もっときっちりと書きます。集合 X に対して、次の条件をみたす二項関係 $\leq$ が定められるとき、二項関係 $\leq$ は集合 X 上の全順序であると言います。そして、このような集合と全順序の組 $(X, \leq)$ を全順序集合と言います。また、二項関係 $\leq$ が全順序律以外の 3 つの条件をみたすとき、$\leq$ を半順序、組 $(X, \leq)$ を半順序集合（または順序集合）と言います。

1. X の任意の元 a に対して、$a \leq a$（反射律）
2. $x \leq y$ かつ $y \leq z$ ならば $x \leq z$（推移律）
3. $x \leq y$ かつ $y \leq x$ ならば $x = y$（反対称律）
4. X の任意の元 x, y に対して、$x \leq y$ または $y \leq x$ のいずれか（両方でもよい）が必ず成り立つ。（全順序律）

[4] Wikipedia - 順序集合 http://ja.wikipedia.org/wiki/順序集合

整列順序 (well-order) を備えている集合を整列集合[5] [6] (well-ordered set) と言います。ここで、集合 X 上の整列順序関係とは、整礎な全順序関係のことであり、半順序 $\leq$ が整礎 (well-founded) であるとは、以下のように定義されます。ここで $a < b$ とは $a \leq b$ かつ $a \neq b$ のことです。

【定義】整礎

集合 X 上の半順序 $\leq$（あるいは狭義半順序 $<$）が整礎であるとは、X の要素による無限降下列が存在しない（つまり、X の要素による列 $a_0 > a_1 > a_2 > \ldots$ の長さは必ず有限になる）ということである。

また、集合 X 上の整列順序関係を、全順序関係でありかつ「X の任意の空でない部分集合 A に対し、A の最小元 a_0 が存在する（つまり、任意の A の元 a に対して、$a_0 \leq a$ が成り立つ）」と定義することもできます。後述の選択公理（あるいはそれよりも弱い従属選択公理）を仮定すると、これは無限降下列が存在しないことと同値になるためです。

整数全体の集合に通常の大小関係を入れると、最小元がないので整列集合ではありませんが、0 以上の整数全体の集合に通常の大小関係を入れると整列集合になります。

0 以上の有理数全体の集合に通常の大小関係を入れると、整列集合にはなりません。部分集合として「正の有理数」を取ったときに、最小元が存在しない（どんなに小さい数 a を取っても、さらに小さい数 $a/2$ が集合の要素となる）ためです。また、数列 $a_n = 1/n$ は無限降下列になります。

ここで選択公理 (axiom of choice)[7] [8] とは、公理的集合論 (axiomatic set theory) における公理のひとつで、どれも空でないような集合を元とする集合（すなわち、集合の集合）があったときに、それぞれの集合から一つずつ元を選び出して新しい集合を作ることができるというものです。カントールは、選択公理を自明なものとみなしていました。本書でも選択公理を仮定します。

選択公理について、公理的集合論における位置付けを少し説明します。カ

[5] 数学 Wiki - 整列集合 http://ja.math.wikia.com/wiki/整列集合
[6] Wikipedia - 整列集合 http://ja.wikipedia.org/wiki/整列集合
[7] Wikipedia - 選択公理 http://ja.wikipedia.org/wiki/選択公理
[8] 選択公理 http://alg-d.com/math/ac/

ントールが提唱した初期の集合論は、集合は「ものの集まり」であると素朴に定義する**素朴集合論** (naive set theory) と呼ばれます。ところが、素朴集合論では「自分自身を要素として含まない集合全体の集合」を考えるとラッセルのパラドックスという矛盾が生じ、他にも「どんな集合でも自由に定義できる」ことによるパラドックスがいろいろと生じます。そこで、集合とはどういうものであるかを公理によって定めることで、「おかしな集合」が定義されることによる矛盾が生じないように理論体系を組み立てようとする公理的集合論が発展しました。

その中で現在一般的に使われている集合の公理系は、ドイツの論理学者・数学者であるエルンスト・ツェルメロ (Ernst Friedrich Ferdinand Zermelo, 1871–1953) と、ドイツ出身でイスラエルに移住した数学者アドルフ・フレンケル (Abraham Halevi Adolf Fraenkel, 1891–1965) らが構築した**ツェルメロ・フレンケルの集合論** (Zermelo-Fraenkel set theory) における **ZFC** 公理系です。ZFC 公理系から選択公理を除いた公理系を **ZF** 公理系と言います。つまり、ZF は Zermelo と Fraenkel の頭文字で、それに選択公理の選択 (choice) の C を加えたものが ZFC です。

次に、**順序型**[9] [10] [11] (order type) の定義をします。順序型とは全順序集合を分類する方法の一つで、すべての全順序集合 $(A, \leq)$ は、順序型に関係付けられます。

2 つの集合 A と B が同じ順序型であるとは、**順序同型** (order isomorphic) であることです。

【定義】順序同型

全順序集合 $(A, \leq)$ と $(B, \leq)$ が順序同型であるとは、すべての A の要素 a_1, a_2 に対して、

$$a_1 \leq a_2 \iff f(a_1) \leq f(a_2)$$

となるような A から B への全単射 f が存在することである。

[9] 数学 Wiki - 順序型 http://ja.math.wikia.com/wiki/順序型
[10] Wikipedia - 順序型 http://ja.wikipedia.org/wiki/順序型
[11] MathWorld - Order Type http://mathworld.wolfram.com/OrderType.html

ここで**全単射** (bijective) [12] とは、写像 $f : A \to B$ に対し、2 つの条件

1. 全射性: $f(A) = B$
2. 単射性: 任意の A の元 a_1, a_2 について、$f(a_1) = f(a_2)$ ならば $a_1 = a_2$

がともに成り立つことを言います。「一対一対応」と言うこともあります。

以上で整列集合と順序型の定義ができましたので、順序数の定義ができたことになります。

5.1.2　超限順序数とフォン・ノイマンの順序数表記

順序数の定義ができたので、まずは有限な全順序集合の順序数、すなわち**有限順序数** (finite ordinal) を考えます。有限な全順序集合は、要素の無限降下列を作ることができないので整礎です。全順序集合 A と B がそれぞれ k 個の要素を持つとき（k は非負整数)、順序同型なので同じ順序型を持ちます。その順序型が順序数で、$0, 1, 2, 3, ...$ のように要素の個数 k と等しい自然数（非負整数）で表記します。このように有限順序数が自然数で表記されるため、順序数は自然数を拡張した概念であるとされます。

有限でない順序数を**無限順序数 (超限順序数)** (inifinite ordinal) と言います。超限順序数は、通常ギリシャ文字の小文字で表記されます。最小の超限順序数は ω と表記され、自然数全体の集合 $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$ の順序型です。これはカントールが定義した超限数 (transfinite number) の中で最小です。

数列 $0, 1, 2, 3, \cdots$ は ω に収束します。このとき $0, 1, 2, 3, \cdots$ を、ω に収束する**基本列** (fundamental sequence) あるいは**収束列**と呼びます。ω に対する基本列は 1 種類とは限りません。たとえば、数列 $0, 2, 4, 6, ...$ も ω への基本列となります。

最小の無限順序数 ω に対して、$\omega + 1, \omega + 2, ...$ といった順序数を定義することができます。なお、順序数では交換法則が成り立ちません。$1 + \omega$ は

[12] Wikipedia - 全単射 http://ja.wikipedia.org/wiki/全単射

$1+1, 1+2, 1+3, ...$ と計算したときの極限であると考えて、$1+\omega=\omega$ となります。

全順序集合 $(A, \leq)$ の順序数が α であれば、α よりも小さい順序数全体の集合と順序同型になります。そのことから、順序数を自分自身よりも小さいすべての順序数の集合であると定義することができます。ハンガリー出身のアメリカの数学者で、多くの学問分野で重要な貢献をしたジョン・フォン・ノイマン (John von Neumann, 1903 - 1957) は、順序数をそのように定義しました。これは標準的な順序数の表記法で、その表記法によれば以下のようになります。

$$
\begin{aligned}
0 &= \{\} \text{ (空集合 = 0 個の要素を持つ集合)} \\
1 &= \{0\} = \{\{\}\} \text{ (1 個の要素を持つ集合)} \\
2 &= \{0,1\} = \{\{\}, \{\{\}\}\} \text{ (2 個の要素を持つ集合)} \\
3 &= \{0,1,2\} \\
4 &= \{0,1,2,3\} \\
\omega &= \{0,1,2,...\} \text{ (全ての自然数の集合)} \\
\omega+1 &= \{0,1,2,...,\omega\} \\
\omega+2 &= \{0,1,2,...,\omega,\omega+1\} \\
\omega+3 &= \{0,1,2,...,\omega,\omega+1,\omega+2\} \\
\omega+\omega &= \{0,1,2,...,\omega,\omega+1,\omega+2,...\}
\end{aligned}
$$

ここで、順序型は集合の濃度 (cardinality) とは異なるので注意が必要です。集合 A から B への全単射が存在するとき、A と B は濃度が等しいと言いますが、この全単射は順序関係を保つ必要がありませんので、順序同型の定義とは異なります。

たとえば、順序数 ω の集合 $A=\{0,1,2,...\}$ と順序数 $\omega+1$ の集合 $B=\{0,1,2,...,\omega\}$ について、A から B への全単射 f が存在するかどうかを考えます。$f(0)=\omega$, $f(k)=k-1 (k>0)$ とすれば、全単射ができるので A と B は濃度が等しくなります。ところがこの全単射は

$$a_1 \leq a_2 \iff f(a_1) \leq f(a_2)$$

を満たしていません（$a_1 = 0, a_2 = 1$ のときに不成立)。この式を満たすような全単射 f は存在しないので（もし存在すれば、$f(k) = \omega$ となる自然数 k が存在するが、順序同型の条件によって $f(k+1)$ は ω よりも大きくなければならず、そのような B の要素は存在しないため)、ω と $\omega + 1$ は濃度は等しいのですが順序同型ではないということが分かります。

集合 A の濃度をあらわす基数 (cardinal number) は、A との間に全単射が存在する順序数の中で最小のものであると定義されます。ω は可算濃度の順序数の中で最小なので、可算無限集合の基数は ω となり、これを ω_0 またはアレフ 0（$\aleph_0$) とも書きます。$\omega + 1$ の基数は ω です。

5.1.3　極限順序数の基本列

α が順序数のとき、ある順序数 β が存在して $\alpha = \beta + 1$ となるならば、α は後続順序数 (successor ordinal) であると言います。0 でも後続順序数でない順序数は極限順序数 (limit ordinal) と呼びます。ω は極限順序数で、$\omega + a(a = 1, 2, 3, ...)$ は後続順序数です。すべての順序数は、0 または後続順序数または極限順序数です。極限順序数は、基本列によって定義できるものと、基本列が存在しないものがあります。

極限順序数は、次のように定義することもできます。

> α よりも小さいすべての順序数 ζ に対して、$\zeta < \xi < \alpha$ となる順序数 ξ が存在するとき、α は極限順序数である。

たとえば $0, 1, 2, \cdots, \omega, \omega + 1$ という数列を考えると、$\alpha = \omega + 1$ に対しては $\zeta = \omega$ とすると $\omega < \xi < \omega + 1$ となる ξ が存在しないので、$\omega + 1$ は極限順序数の定義には合致しませんが、$\alpha = \omega$ に対しては、α よりも小さい順序数 ζ（つまり自然数）を取ると、$\xi = \zeta + 1$ が必ず存在するため、極限順序数の定義に合致します。

$\omega + 1, \omega + 2, \cdots$ という基本列を考えると、この数列は $\omega + \omega$ に収束します。したがって $\omega + \omega$ は極限順序数です。このときに $\omega + \omega$ のことを $\omega \times 2$ と書きます。ここでも順序数に関しては通常の交換法則が成り立たず、$2 \times \omega = \omega$ となりますので（$2 + 2 + \cdots$ と計算していったときの極限

は、$1+1+\cdots$ と計算することの極限と同じ)、2ω ではなく $\omega \times 2$ と表記する必要があります。

極限順序数 $\omega \times 2$ に対して、後続順序数 $\omega \times 2 + a(a = 1, 2, 3, ...)$ を定義できます。そして $\omega \times 3$ は $\omega \times 2 + 1, \omega \times 2 + 2, ...$ を基本列とする極限順序数です。このように後続順序数の定義 $(+1)$ と極限順序数の定義を繰り返す事で、順序数が定義されます。基本列 $a_1, a_2, a_3, \cdots$ の極限順序数 α を $\alpha = \lim(a_1, a_2, a_3, ...)$ と表記することにすると、以下のように順序数が定義されます。

$$\begin{aligned}
\omega \times 3 &= \lim(\omega \times 2, \omega \times 2 + 1, \omega \times 2 + 2, ...) \\
\omega \times 4 &= \lim(\omega \times 3, \omega \times 3 + 1, \omega \times 3 + 2, ...) \\
\omega \times a &= \lim(\omega \times (a-1), \omega \times (a-1) + 1, \omega \times (a-1) + 2, ...)
\end{aligned}$$

このように、極限順序数 $\omega \times a$ は、$\omega \times (a-1)$ から後続順序数を順番に計算する基本列を作れば定義できます。続いて、このように定義された $\omega \times a$ を使って、

$$\omega^2 := \omega \times \omega = \lim(0, \omega, \omega \times 2, \omega \times 3, ...)$$

といった基本列を作ることにより、ω^2 という順序数を定義できます。これよりも大きな順序数については、これまでと同じような基本列の作り方で、以下のように極限順序数の定義を続けることができます。

$$\begin{aligned}
\omega^2 + \omega &= \lim(\omega^2, \omega^2 + 1, \omega^2 + 2, ...) \\
\omega^2 + \omega \times a &= \lim(\omega^2 + \omega \times (a-1), \omega^2 + \omega \times (a-1) + 1, ...) \\
\omega^2 \times 2 &= \omega^2 + \omega^2 = \lim(\omega^2, \omega^2 + \omega, \omega^2 + \omega \times 2, ...) \\
\omega^3 &= \omega^2 \times \omega = \lim(0, \omega^2, \omega^2 \times 2, \omega^2 \times 3, ...) \\
\omega^\omega &= \lim(\omega^0, \omega^1, \omega^2, \omega^3, \omega^4, ...) \\
\omega^\omega \times 2 &= \omega^\omega + \omega^\omega = \lim(\omega^\omega + \omega^0, \omega^\omega + \omega^1, \omega^\omega + \omega^2, ...) \\
\omega^{\omega+1} &= \omega^\omega \times \omega = \lim(0, \omega^\omega, \omega^\omega \times 2, \omega^\omega \times 3, ...) \\
\omega^{\omega+2} &= \omega^{\omega+1} \times \omega = \lim(0, \omega^{\omega+1}, \omega^{\omega+1} \times 2, \omega^{\omega+1} \times 3, ...) \\
\omega^{\omega \times 2} &= \omega^{\omega+\omega} = \lim(\omega^\omega, \omega^{\omega+1}, \omega^{\omega+2}, ...)
\end{aligned}$$

$$
\begin{aligned}
\omega^{\omega^2} &= \omega^{\omega\times\omega} = \lim(\omega^0, \omega^\omega, \omega^{\omega\times 2}, \omega^{\omega\times 3}, ...) \\
\omega^{\omega^\omega} &= \lim(\omega^{\omega^0}, \omega^{\omega^1}, \omega^{\omega^2}, \omega^{\omega^3}, ...) \\
\omega^{\omega^\omega} + \omega &= \lim(\omega^{\omega^\omega}, \omega^{\omega^\omega} + 1, \omega^{\omega^\omega} + 2, ...) \\
\omega^{\omega^\omega} + \omega \times 2 &= \lim(\omega^{\omega^\omega} + \omega, \omega^{\omega^\omega} + \omega + 1, \omega^{\omega^\omega} + \omega + 2, ...)
\end{aligned}
$$

また、それぞれの極限順序数に自然数を加えた順序数、たとえば $\omega^{\omega^\omega} + 3$ は、後続順序数になります。

ここまでで、ω、ω^ω、ω^{ω^ω} を定義することができました。ここで ω^^a := ω^ω^ω^...ω (a 個の ω) と表記することにすると、同様の定義を繰り返せば、

$$\omega, \omega^\omega, \omega\text{^^}3, \omega\text{^^}4, \omega\text{^^}5, ...$$

を、順次定義できます。そこでこれを基本列として、極限順序数 ϵ_0（エプシロン・ノート）を

【定義】エプシロン・ノート

$$\epsilon_0 = \lim(1, \omega, \omega^\omega, \omega\text{^^}3, \omega\text{^^}4, \omega\text{^^}5, ...)$$

と定義します。ϵ_0 は、ω から有限回の加算・乗算・冪乗では到達できない最小の順序数です。

5.1.4　カントールの標準形

全ての順序数は、カントールの標準形[13] (Cantor normal form) (しばしば CNF と略される) で一意に書くことができます。

[13] http://www.proofwiki.org/wiki/Definition:Cantor_Normal_Form

【定義】カントールの標準形

順序数 α を、正の整数 $c_1, c_2, c_3, \cdots, c_k$ と順序数 $\beta_1 > \beta_2 > \cdots > \beta_k \geq 0$ によって、

$$\begin{aligned} \alpha &= \sum_{i=1}^{k} \omega^{\beta_i} c_i \\ &= \omega^{\beta_1} c_1 + \omega^{\beta_2} c_2 + \cdots + \omega^{\beta_k} c_k \end{aligned}$$

の形で書くとき、これをカントールの標準形と言う。

ϵ_0 をカントール標準形で書くと $\epsilon_0 = \omega^{\epsilon_0}$ となります。これは自分自身を参照しているので、定義式として使うにはふさわしくありません。このように、カントール標準形は順序数を表記するための万能の手法ではありませんが、ϵ_0 よりも小さい順序数を表記するときには便利です。

ϵ_0 は可算濃度の順序数なので、基数は ω です。つまり ϵ_0 よりも小さい順序数と自然数の間に全単射が存在します。たとえば、ϵ_0 よりも小さい順序数 α をカントール標準形で書いた時の文字数を $f(\alpha)$ として、$f(\alpha)$ が小さい順序数から、それぞれ小さい順に並べていけば良いのです。

- $f(\alpha) = 1 : 0, 1, 2, ..., 9, \omega$
- $f(\alpha) = 2 : 10, 11, 12, ..., 99, \omega 2, \omega 3, ..., \omega 9, \omega^2, \omega^3, ..., \omega^9, \omega^\omega$

のように並べることができます。

5.2 順序数階層

本書では、順序数 α から、自然数から自然数への関数 $\mathbb{N} \to \mathbb{N}$ への写像によって、関数の階層を定める手法を順序数階層 (ordinal hierarchy) と呼ぶこととします。具体的にはハーディ階層、急増加関数等があり、これから説明します。

5.2.1 ハーディ階層

1904 年にイギリスの数学者ゴッドフレイ・ハーディ (Godfrey Harold Hardy, 1877–1947) が定義したハーディ階層[14] [15] (Hardy function) は、順序数 α に対して自然数から自然数への関数 $H[\alpha] : \mathbb{N} \to \mathbb{N}$ を定める順序数による関数の階層で、以下のように定義されます。

【定義】ハーディ階層

$$
\begin{aligned}
H[0](n) &= n \\
H[\alpha+1](n) &= H[\alpha](n+1) \\
H[\alpha](n) &= H[\alpha[n]](n)(\text{極限順序数}\alpha\text{の基本列が}\alpha[n]\text{ である時})
\end{aligned}
$$

順序数の定義の中で、後続順序数の定義 (順序数に+1 する) と、基本列の極限による極限順序数の定義を、それぞれ関数に $x+1$ を代入する操作と、対角化の操作に対応させることにより、関数の階層性を順序数の階層性と対応づけています。

ここで定義の中に基本列が用いられていますが、基本列の取り方は一様ではないため、α の定義にあらわれるすべての基本列の取り方を厳密に定めて、はじめて $H[\alpha]$ が定義されたことになります。ただし、基本列のとり方による増大度の違いよりは、順序数の大きさによる増大度の違いの方がはるかに大きいため、基本列の取り方いかんに関わらず、$\alpha > \beta$ であれば $H[\alpha] > H[\beta]$ がおおよそ成り立つと考えて、H[順序数] と記述して大体の大きさを表します。

また、基本列の中でなんらかの方法で 1 つの正規な基本列 (canonical fundamental sequence) $\alpha[0], \alpha[1], \alpha[2], ...$ を与えることで、$H[\alpha]$ の定義は一意に定まります。

そのような方法の 1 つとして、巨大数論では $\alpha \leq \epsilon_0$ に対して、次のようなワイナー階層 (Wainer hierarchy) がよく使われます。

[14] http://ja.googology.wikia.com/wiki/ハーディ階層

[15] Hardy, G.H. (1904) A theorem concerning the infinite cardinal numbers. *Quarterly Journal of Mathematics* 35: 87–94.

【定義】ワイナー階層

$\alpha \leq \epsilon_0$ に対して、α の基本列 $\alpha[0], \alpha[1], \alpha[2], ...$ を次のように定める。

$$
\begin{aligned}
\omega[n] &= n \\
\omega^{\alpha+1}[n] &= \omega^{\alpha} n \\
\omega^{\alpha}[n] &= \omega^{\alpha[n]} (\alpha\text{は極限順序数}) \\
(\omega^{\alpha_1} + \omega^{\alpha_2} + \cdots + \omega^{\alpha_k})[n] &= \omega^{\alpha_1} + \omega^{\alpha_2} + \cdots + \omega^{\alpha_k}[n] \\
& \quad (\alpha_1 \geq \alpha_2 \geq \cdots \geq \alpha_k \text{の場合}) \\
\epsilon_0[0] &= 1, \epsilon_0[n+1] = \omega^{\epsilon_0[n]}
\end{aligned}
$$

ここで、$\epsilon_0[0] = 0$ とされることもあります。本書でも、ϵ_0 以下の順序数に対してはワイナー階層を用います。たとえば $\omega[n] = n$ は、ω の基本列が $0, 1, 2, 3, ...$ となることを意味します。ここまでの順序数の基本列は、すべてこのワイナー階層を用いて書きました。

それでは、ワイナー階層を使ってハーディ階層を計算してみます。

$$
\begin{aligned}
H[3](n) &= H[2](n+1) = H[1](n+2) = H[0](n+3) = n+3 \\
H[m](n) &= n+m \\
H[\omega](n) &= H[n](n) = n+n = 2n \\
H[\omega+2](n) &= H[\omega+1](n+1) = H[\omega](n+2) = 2(n+2) \\
H[\omega+k](n) &= 2(n+k) \\
H[\omega \times 2](n) &= H[\omega+n](n) = 2(n+n) = 4n \\
H[\omega \times 2 + 2](n) &= H[\omega \times 2 + 1](n+1) = H[\omega \times 2](n+2) = 4(n+2) \\
H[\omega \times 3](n) &= H[\omega \times 2 + \omega](n) = 4(n+n) = 8n = 2^3 n \\
H[\omega \times k](n) &= 2^k n \\
H[\omega^2](n) &= H[\omega \times n](n) = 2^n n \\
H[\omega^2+3](n) &= H[\omega^2](n+3) = 2^{n+3}(n+3) \\
H[\omega^2+\omega](n) &= H[\omega^2+n](n) = 2^{n+n}(n+n) = 2^{2n}(2n) \\
H[\omega^2+\omega+k](n) &= H[\omega^2+\omega](n+k) = 2^{2(n+k)}2(n+k)
\end{aligned}
$$

$$
\begin{aligned}
H[\omega^2+\omega\times 2](n) &= H[\omega^2+\omega+n](n) = 2^{2(n+n)}2(n+n) = 2^{4n}(4n)\\
H[\omega^2+\omega\times 3](n) &= H[\omega^2+\omega\times 2+n](n) = 2^{4(n+n)}4(n+n)\\
&= 2^{8n}(8n) = 2^{2^3 n}(2^3 n)\\
H[\omega^2+\omega\times k](n) &= 2^{2^k n}(2^k n)\\
H[\omega^2\times 2](n) &= H[\omega^2+\omega\times n](n) = 2^{2^n n}(2^n n)
\end{aligned}
$$

ここで、ハーディ階層には次のような性質があります。

ハーディ階層による関数の合成

$$H[\alpha+\beta](n) = H[\alpha](H[\beta](n))$$

ただし、$\alpha+\beta=\gamma+\beta, \alpha>\gamma$ を満たす γ が存在する場合を除く。

つまり、関数 $H[\alpha+\beta](n)$ は関数 $H[\alpha](n)$ と $H[\beta](n)$ の合成関数です。ただし書きは、$\alpha=\omega^2+2, \beta=\omega$ のときには

$$\alpha+\beta=\omega^2+2+\omega=\omega^2+\omega$$

のように計算されて $+2$ が消えてしまうので、そのような場合を除くということです。

$\alpha=\omega^2, \beta=\omega+3$ のときに $H[\alpha+\beta](n)=H[\alpha](H[\beta](n))$ が成立していることを確認します。

$$H[\alpha+\beta](n) = H[\omega^2+\omega+3](n) = 2^{2(n+3)}2(n+3)$$

一方、

$$
\begin{aligned}
H[\alpha](n) &= H[\omega^2](n) = 2^n n\\
H[\beta](n) &= H[\omega+3](n) = 2(n+3)\\
H[\alpha](H[\beta](n)) &= 2^{2(n+3)}2(n+3)
\end{aligned}
$$

となるため、たしかに成立しています。

関数の合成を繰り返すと、

$$H[\alpha\times 2](n) = H[\alpha+\alpha](n) = H[\alpha](H[\alpha](n)) = H[\alpha]^2(n)$$

$$\begin{aligned}
H[\alpha\times 3](n) &= H[\alpha\times 2+\alpha](n) = H[\alpha\times 2](H[\alpha](n)) \\
&= H[\alpha]^2(H[\alpha](n)) = H[\alpha]^3(n) \\
H[\alpha\times 4](n) &= H[\alpha]^4(n) \\
H[\alpha\times k](n) &= H[\alpha]^k(n) \\
H[\alpha\times \omega](n) &= H[\alpha]^n(n)
\end{aligned}$$

となります。関数 $H[\alpha\times\omega](n)$ は関数 $H[\alpha](n)$ の合成を n 回繰り返した関数、つまり合成を数え上げる原始再帰の操作をした関数です。このことを使うと、

$$\begin{aligned}
H[\omega](n) &= 2n \\
H[\omega^2](n) &= H[\omega\times\omega](n) = H[\omega]^n(n) = 2^n n
\end{aligned}$$

となり、

$$H[\omega^3](n) = H[\omega^2\times\omega](n) = H[\omega^2]^n(n)$$

を計算するために、$f(n) = 2^n n$ として

$$\begin{aligned}
H[\omega^2](n) &= f(n) = 2^n n \\
H[\omega^2\times 2](n) &= f^2(n) = 2^{2^n n}(2^n n) \\
H[\omega^2\times 3](n) &= f^3(n) = 2^{2^{2^n n}(2^n n)}2^{2^n n}(2^n n)
\end{aligned}$$

のように計算されていきます。ここでざっくりと $f(n) = 2^n$ とすれば

$$\begin{aligned}
H[\omega^2](n) &> f(n) = 2^n \\
H[\omega^2\times 2](n) &> f^2(n) = 2\uparrow 2\uparrow n > 2\uparrow\uparrow 2 \\
H[\omega^2\times 3](n) &> f^3(n) = 2\uparrow 2\uparrow 2\uparrow n > 2\uparrow\uparrow 3 \\
H[\omega^2\times 4](n) &> f^4(n) = 2\uparrow 2\uparrow 2\uparrow 2\uparrow n > 2\uparrow\uparrow 4
\end{aligned}$$

のようにテトレーションで近似できて、

$$H[\omega^3](n) > 2\uparrow\uparrow n$$

となります。$H[\omega^2](n) > 2^n$（べき乗）の原始再帰から $H[\omega^3](n) > 2\uparrow\uparrow n$（テトレーション）ができたように、テトレーションの原始再帰から

$H[\omega^4](n) > 2 \uparrow\uparrow\uparrow n$（ペンテーション）ができて、一般に $m > 1$ のときに

$$H[\omega^m](n) \quad > \quad 2 \uparrow^{m-1} n$$

となります。

【証明】

帰納法で証明する。$m = 2$ では成立する。$m = k$ で成立するとすれば、$m = k + 1$ の時

$$\begin{aligned} H[\omega^k](n) &> 2 \uparrow^{k-1} n \\ H[\omega^k \times 2](n) &= H[\omega^k]^2(n) > 2 \uparrow^{k-1} 2 \uparrow^{k-1} n > 2 \uparrow^k 2 \\ H[\omega^k \times 3](n) &= H[\omega^k]^3(n) > 2 \uparrow^{k-1} 2 \uparrow^{k-1} 2 \uparrow^{k-1} n > 2 \uparrow^k 3 \\ H[\omega^{k+1}](n) &= H[\omega^k \times \omega](n) > 2 \uparrow^k n \end{aligned}$$

となり、成立している。 □

5.2.2　急増加関数

巨大数論では、ハーディ階層と似ている急増加関数[16] (FGH; Fast-growing hierarchy) [17] $F[\alpha](n)$ がよく使われます。

【定義】急増加関数 (FGH)

$$\begin{aligned} F[0](n) &= n + 1 \\ F[\alpha + 1](n) &= F[\alpha]^n(n) \\ F[\alpha](n) &= F[\alpha_n](n) (\alpha\text{が極限順序数の時}) \end{aligned}$$

巨大数研究 Wiki では $f_\alpha(n)$ という表記が使われていますが、重要な順序数を下付き文字にしてしまうと読みにくくなるので本書では $F[\alpha](n)$ の

[16] 巨大数研究 Wiki - 急増加関数 http://ja.googology.wikia.com/wiki/急増加関数

[17] 田中一之, 角田法也, 鹿島亮, 菊池誠 (1997) 『数学基礎論講義—不完全性定理とその発展』日本評論社

表記を使います。すなわち、

$$F[\alpha](n) = f_\alpha(n)$$

です。$\alpha < \epsilon_0$ の時、ハーディ階層と急増加関数の間には

$$F[\alpha](n) = H[\omega^\alpha](n)$$

という関係があります。急増加関数の 1 番目の定義式については

$$F[0](n) = n + 1 = H[1](n) = H[\omega^0](n)$$

と一致し、2 番目の定義式については $F[\alpha](n) = H[\omega^\alpha](n)$ が成立すれば

$$\begin{aligned} F[\alpha+1](n) &= F[\alpha]^n(n) = H[\omega^\alpha]^n(n) \\ &= H[\omega^\alpha \times \omega](n) \\ &= H[\omega^{\alpha+1}](n) \end{aligned}$$

となって、$F[\alpha+1](n)$ についても成立するためです。$\alpha = \epsilon_0$ のときには、$F[\alpha](n) = H[\omega^\alpha](n)$ は厳密には成立しませんが

$$\begin{aligned} F[\epsilon_0](3) &= F[\omega^{\omega^\omega}](3) \\ &= H[\omega^{\omega^{\omega^\omega}}](3) \\ &\approx H[\epsilon_0](4) \end{aligned}$$

となり、$F[\epsilon_0](n) \approx H[\epsilon_0](n+1) = H[\epsilon_0+1](n)$ の近似が成り立ちます。もっと大雑把に $F[\epsilon_0](n) \approx H[\epsilon_0](n)$ と近似しても良いでしょう。

ハーディ階層と急増加関数を比較すると、急増加関数は順序数が+1 されるだけで $f^n(n)$ へととても関数が大きく変わりますので、非常に大まかな近似をするときに適していますが、ハーディ階層は、より細やかな近似をするときに適しています。

1953 年にポーランドの数学者アンジェイ・グジェゴルチク (Andrzej Grzegorczyk, 1922–2014) が発表したグジェゴルチク階層 (Grzegorczyk hierarchy) [18]は、急増加関数の元になったとされています。厳密な定義は省略し

[18]Grzegorczyk, A. (1953) Some classes of recursive functions, *Rozprawy Matematyczne* 4: 1–45.

ますが、グジェゴルチク階層の n 番目の階層 $\mathcal{E}^n$ に属する関数は、次のような関数になります。

- $\mathcal{E}^0$ は $x+1, x+2, ...$ を含む
- $\mathcal{E}^1$ は加法関数 $x+y, 3x, ...$ を含む
- $\mathcal{E}^2$ は乗法関数 $xy, x^3, ...$ を含む
- $\mathcal{E}^3$ はべき乗関数 $x^y, 2^{2^{2^x}}$ を含む (初等再帰関数, elementary recursive function)
- $\mathcal{E}^4$ はテトレーション関数を含む

ハイパー n 演算子は $\mathcal{E}^n$ に含まれますが、$\mathcal{E}^{n-1}$ に含まれません。グジェゴルチク階層に含まれる関数はすべて原始再帰関数で、原始再帰関数はグジェゴルチク階層のどこかに含まれます。グジェゴルチク階層のレベルを超限順序数に拡張することで急増加関数とすることができることから、急増加関数を拡張グジェゴルチク階層 (extended Grzegorczyk hierarchy) と呼ぶこともあります。

数学史家の鈴木真治によると[19]、ハーディ階層が定義された 1904 年の論文は、カントールの定理を精密化しようとしたものであり、証明論へ応用されることは想像もしていなかっただろうとのことです。もちろん急激に増加する関数を作ろうという意図はなかったことでしょう。そして、急増加関数の歴史的初出はグジェゴルチク階層がよく挙げられるものの、そのオリジナルの定義は現行の急増加関数とはかなり違うので、これを初出とするのは疑問だとしています。1953 年に出版されたイギリスの数学者ジョン・エデンサー・リトルウッド (John Edensor Littlewood, 1885–1977) の著作を集めた『数学雑談』[20] に収録されている「大きな数」というエッセイに書かれている関数の定義は、現行の急増加関数の定義に近く、急激に大きくなる関数を見つけようとする意図を持って定義されたものであるため、これが急増加関数の歴史的初出であろうとしています。鈴木真治の本では、

[19] 鈴木真治 (2016)『巨大数』岩波書店
[20] Littlewood, J. E. (1953) A mathematician's miscellany. Methuen

リトルウッドによる急増加関数の定義を『数学雑談』の和訳本[21]から引用しています。

さて、ハーディ階層の計算で $H[\omega^m](n) > 2 \uparrow^{m-1} n$ を示しましたので、

$$F[m](n) = H[\omega^m](n) > 2 \uparrow^{m-1} n$$

となります。この近似計算については、『寿司 虚空編』第 6 話で $m = 3$ について計算がされています。関数 $F[\alpha](n)$ を合成する操作を数え上げる原始再帰関数が $F[\alpha + 1](n)$ となることから、原始再帰操作を m 回使ってできる関数は、$F[m](n)$ 程度の大きさになります。変数 n の回数だけ原始再帰操作を数え上げる関数 $F[n](n)$ は、どのような原始再帰関数よりも大きい 2 重再帰関数です。

$$F[\omega](n) = F[n](n)$$

となることから、$F[\omega](n)$ は 2 重再帰関数となります。そして、矢印表記やアッカーマン関数を使って

$$F[\omega](n) > 2 \uparrow^{n-1} n = A(n+1, n-3) + 3 > A(n, n)$$

のように近似できます。

グラハム数は $f(n) = 3 \uparrow^n 3$ としたときの $f^{64}(4)$ であり、$f(n) \approx F[\omega](n)$ なので、

$$\begin{aligned} F[\omega + 1](n) &= F[\omega]^n(n) \approx f^n(n) \\ F[\omega + 1](64) &= f^{64}(64) \approx \text{グラハム数} \end{aligned}$$

となります。巨大数論の近似に慣れると、$f(n) = 3 \uparrow^n 3$ は ω で $f^n(n)$ は+1 だから、$\omega + 1$ のレベルの関数になると分かるようになります。

チェーン表記は、チェーンの右端の数を 1 つ増やすことが原始再帰なので順序数に 1 を足すことに相当し、チェーンを 1 つ伸ばすことは順序数に ω を足す 2 重再帰に相当します。したがって、

$$3 \to 3 \to n \approx F[\omega](n)$$

[21] ボロバシュ (編) 金光 滋 (訳) (1990) 『リトルウッドの数学スクランブル』近代科学社

$$
\begin{aligned}
3 \to 3 \to 3 \to n &\approx F[\omega+\omega](n) = F[\omega \times 2](n) \\
3 \to 3 \to 3 \to 3 \to n &\approx F[\omega \times 2 + \omega](n) = F[\omega \times 3](n)
\end{aligned}
$$

となり、拡張チェーン表記では次のようになります。

拡張チェーン表記の急増加関数による近似

$$
\begin{aligned}
3 \to_2 n &\approx F[\omega \times \omega](n) = F[\omega^2](n) \\
3 \to_2 3 \to_2 n &\approx F[\omega^2 + \omega](n) \\
3 \to_2 3 \to_2 3 \to_2 n &\approx F[\omega^2 + \omega \times 2](n) \\
3 \to_2 3 \to_2 3 \to_2 3 \to_2 n &\approx F[\omega^2 + \omega \times 3](n) \\
3 \to_3 n &\approx F[\omega^2 + \omega \times \omega](n) = F[\omega^2 \times 2](n) \\
3 \to_4 n &\approx F[\omega^2 \times 3](n) \\
3 \to_n n &\approx F[\omega^2 \times \omega](n) = F[\omega^3](n)
\end{aligned}
$$

このように、急増加関数の順序数に 1 を足すことは原始再帰で、ω を足すことは 2 重再帰、ω^2 を足すことは 3 重再帰となり、一般に n 重再帰について

- n 重再帰作用素は急増加関数の順序数に ω^{n-1} を足す効果を持つ
- n 重再帰関数は $F[\omega^{n-1}a_{n-1} + \omega^{n-2}a_{n-2} + ... + a_0](n)$ と近似できるような再帰の構造を持った関数

となります。このことが理解できれば、n 重再帰という考え方よりも急増加関数で再帰の構造を考える方が分かりやすいと思います。

次に、多変数アッカーマン関数とふぃっしゅ数バージョン 3、すなわち F_3 の $s(n)$ 変換について、急増加関数を使って近似します。

$f_b(n) = A(X, b, n)$ (X は 0 個以上の 0 以上の整数) とすると、

$$
\begin{aligned}
f_{b+1}(n) &= A(X, b+1, n) \\
&= A(X, b, A(X, b+1, n-1)) \\
&= f_b(A(X, b+1, n-1)) \\
&= f_b^2(A(X, b+1, n-2))
\end{aligned}
$$

$$\begin{aligned} &= \quad \cdots = f_b^n(A(X, b+1, 0)) \\ &\approx \quad f_b^n(n) \end{aligned}$$

となります。このことと $s(1)$ 変換の定義式 $s(1)f(x) = f^x(x)$ から、以下の 3 つが同じ計算をしていることが分かります。

1. 多変数アッカーマンで $f_b(n) = A(X, b, n)$ として $f_{b+1}(n) \approx f_b^n(n)$
2. $s(1)$ 変換の $s(1)f(n) = f^n(n)$
3. 急増加関数の $F[\alpha+1](n) = F[\alpha]^n(n)$

つまり、多変数アッカーマン関数で右から 2 番目の項に 1 を加えることと、$s(1)$ 変換をすることは、いずれも急増加関数で順序数に 1 を足すことと同じです。

$f(n) = n + 1$ とすると、

$$\begin{aligned} A(2, n) &\approx s(1)f(n) = f^n(n) = F[1](n) = 2n \\ A(3, n) &\approx s(1)^2 f(n) = F[2](n) = 2^n n \\ A(4, n) &\approx s(1)^3 f(n) = F[3](n) \approx 2 \uparrow^2 n \\ A(m+1, n) &\approx s(1)^m f(n) = F[m](n) \approx 2 \uparrow^{m-1} n \\ A(n, n) &\approx s(1)^n f(n) = F[n](n) \end{aligned}$$

のように、$s(1)$ 変換と急増加関数は完全に一致し、アッカーマン関数は両者に近似されます。そして $A(1, 0, n) = A(n, n)$, $s(2)f(n) = s(1)^n f(n)$, $F[\omega](n) = F[n](n)$ より、上式の最後は

$$A(1, 0, n) \approx s(2)f(n) = F[\omega](n)$$

と書き換えることができます。3 変数アッカーマン関数、$s(2)$ 変換、ω がこのように対応します。ここに $f_{b+1}(n) \approx f_b^n(n)$, $s(1)$ 変換, $F[\alpha+1](n) = F[\alpha]^n(n)$ を重ねることで

$$\begin{aligned} A(1, 1, n) &\approx s(1)s(2)f(n) = F[\omega+1](n) \\ A(1, 2, n) &\approx s(1)^2 s(2)f(n) = F[\omega+2](n) \end{aligned}$$

$$\begin{aligned} A(1,3,n) &\approx s(1)^3 s(2) f(n) = F[\omega+3](n) \\ A(1,n,n) &\approx s(1)^n s(2) f(n) = F[\omega+\omega](n) \end{aligned}$$

ここで、最後の式は次の式と同じになります。

$$A(2,0,n) \approx s(2)^2 f(n) = F[\omega \times 2](n)$$

このように、

$$\begin{aligned} A(1,0,n) &\approx s(2) f(n) = F[\omega](n) \\ A(2,0,n) &\approx s(2)^2 f(n) = F[\omega \times 2](n) \end{aligned}$$

となることから、以下同様に計算を続けると

$$\begin{aligned} A(3,0,n) &\approx s(2)^3 f(n) = F[\omega \times 3](n) \\ A(4,0,n) &\approx s(2)^4 f(n) = F[\omega \times 4](n) \\ A(n,0,n) &\approx s(2)^n f(n) = F[\omega \times n](n) = F[\omega^2](n) \\ A(1,0,0,n) &\approx s(3) f(n) = F[\omega^2](n) \\ A(2,0,0,n) &\approx s(3)^2 f(n) = F[\omega^2 \times 2](n) \\ A(n,0,0,n) &\approx s(3)^n f(n) = F[\omega^2 \times n](n) = F[\omega^3](n) \\ A(1,0,0,0,n) &\approx s(4) f(n) = F[\omega^3](n) \end{aligned}$$

のように計算されます。同様に計算を続けると、たとえば

$$A(3,2,1,0,n) \approx s(2)s(3)^2 s(4)^3 f(n) = F[\omega^3 \times 3 + \omega^2 \times 2 + \omega](n)$$

のようになり、一般に次のようになります。

多変数アッカーマン関数と $s(n)$ 変換の急増加関数による近似

$$\begin{aligned} A(a_k,...,a_2,a_1,a_0,n) &\approx s(1)^{a_0} s(2)^{a_1} ... s(k+1)^{a_k} f(n) \\ &= F[\omega^k \times a_k + ... + \omega^2 \times a_2 + \omega \times a_1 + a_0](n) \\ A(\underbrace{1,1,...,1}_{n}) &\approx s(n) f(n) \approx F[\omega^\omega](n) \end{aligned}$$

すなわち、多変数アッカーマン関数と $s(n)$ 変換の上限は、$F[\omega^\omega](n)$ です。

さらに、F_3 の $ss(n)$ 変換については、

$$\begin{aligned}
ss(1)f(n) &= s(n)f(n) \approx F[\omega^{\omega}](n) \\
ss(1)^2 f(x) &\approx F[\omega^{\omega}+\omega^{\omega}](n) = F[\omega^{\omega}\times 2](n) \\
ss(2)f(n) &= ss(1)^n f(n) \approx F[\omega^{\omega}\times\omega](x) = F[\omega^{\omega+1}](x) \\
ss(3)f(n) &= ss(2)^n f(n) \approx F[\omega^{\omega+2}](n) \\
ss(4)f(n) &= ss(3)^n f(n) \approx F[\omega^{\omega+3}](n) \\
ss(n)f(n) &\approx F[\omega^{\omega\times 2}](n)
\end{aligned}$$

となり、ふぃっしゅ関数とふぃっしゅ数は次のように近似されます。

$$\begin{aligned}
F_3(n) &= ss(2)^{63}f(n) \approx F[\omega^{\omega+1}\times 63](n) \\
F_3 &= F_3^{63}(3) \approx F[\omega^{\omega+1}\times 63+1](63)
\end{aligned}$$

BEAF の配列表記については、多変数アッカーマン関数との間に

$$\begin{aligned}
\{n+1, m+1, a_0+1, a_1+1, a_2+1, ..., a_k+1\} &\approx f^m(n), \\
f(n) &= A(a_k, ..., a_2, a_1, a_0, n)
\end{aligned}$$

という関係があることを示しました ($n>1$)。したがって急増加関数によって次のように近似できます。

配列表記の急増加関数による近似

$n>1$ として

$$\begin{aligned}
& \{n+1, m+1, a_0+1, a_1+1, a_2+1, ..., a_k+1\} \\
\approx\ & F[\omega^k\times a_k + ... + \omega^2\times a_2 + \omega\times a_1 + a_0]^m(n) \\
=\ & H[\omega^{\omega^k\times a_k+...+\omega^2\times a_2+\omega\times a_1+a_0}\times m](n) \\
\{\underbrace{3, 3, ..., 3}_{n}\} \approx\ & F[\omega^{\omega}](n)
\end{aligned}$$

ふぃっしゅ数バージョン 1 と 2 はすでに多変数アッカーマン関数で近似されているので、以上の関係から急増加関数や配列表記でも近似できます。F_3 までの近似をまとめます。

ふぃっしゅ数バージョン 3 までの近似

$$
\begin{aligned}
F_1 &\approx A(1,0,1,63) \approx \{4,64,1,1,2\} \approx F[\omega^2+1](63) \\
F_2 &\approx A(63,0,0,63) \approx \{4,2,1,1,64\} \approx F[\omega^3](63) \\
F_3 &\approx F[\omega^{\omega+1} \times 63 + 1](63)
\end{aligned}
$$

5.2.3 緩増加関数

順序数階層として、もう 1 つ**緩増加関数**[22] (slow-growing hierarchy) を紹介しておきます。

【定義】緩増加関数 (SGH)

$$
\begin{aligned}
G[0](n) &= 0 \\
G[\alpha+1](n) &= G[\alpha](n)+1 \\
G[\alpha](n) &= G[\alpha_n](n) (\alpha\text{が極限順序数の時})
\end{aligned}
$$

これはハーディ階層よりも遅く成長します。ワイナー階層を使うと

$$
\begin{aligned}
G[0](n) &= 0 \\
G[1](n) &= 1 \\
G[2](n) &= 2 \\
G[m](n) &= m \\
G[\omega](n) &= n \\
G[\omega 2](n) &= 2n \\
G[\omega^2](n) &= n^2 \\
G[\omega^\omega](n) &= n^n \\
G[\omega^\omega+\omega](n) &= n^n+n
\end{aligned}
$$

[22] 巨大数研究 Wiki - 緩増加関数 http://ja.googology.wikia.com/wiki/緩増加関数

$$G[\omega^{\omega^{\omega}}](n) \quad = \quad n^{n^n}$$

のように、順序数の ω を変数 n に置換した関数になるのが面白いところです。したがって、

$$G[\epsilon_0](n) = n \uparrow\uparrow n$$

となります。順序数階層の中ではとても遅く成長するため、巨大数論であまり使われることはありませんが、テトレーションレベルの関数を順序数で階層化するのには適しています。

ここまで紹介した 3 つの順序数階層の成長速度は、緩増加関数 $<$ ハーディ階層 $<$ 急増加関数となることが分かりました。そしてハーディ階層は順序数 ϵ_0 で急増加関数に追いつきます。それでは、緩増加関数は急増加関数にどこかで追いつくのでしょうか？ 実は、「自然な」基本列を使うと $\psi_0(\Omega_\omega)$ という順序数で追いつきます。このことはいずれまた出てきます。緩増加関数がどの順序数で急増加関数に追いつくかは基本列の選び方によります。基本列の選び方によっては、ϵ_0 で追いつくこともあります。

5.3 順序数解析

順序数解析 (ordinal analysis) では、数学の理論体系の証明力を順序数によって評価します。理論体系の証明力を順序数で評価するとはどういう意味か、という話をするためには、まずは理論体系を形式化するための形式体系とは何か、ということから話を始める必要があります。

順序数による関数の評価をするにあたり、当面は順序数階層だけを理解していれば大丈夫（なので、とりあえず順序数解析の話は飛ばして次の章に進んでも良い）ですが、やがて順序数解析を使う関数や巨大数が出てきますので、ここで説明をしておきます。

形式体系についてしっかりと理解するためには、数理論理学 (mathematical logic) や数学基礎論 (foundations of mathematics) の分野の本を読み込む必要がありますが、ここでは簡潔に記してそのとっかかりとします。

5.3.1 形式体系

形式体系 (formal system) とは、広い意味では数学のモデルにおける明確に定義された (well-defined) 抽象的思考体系で、以下の要素から構築されます。

1. 式を構築するために使う有限個の記号 (symbol)。
2. 記号から論理式を構築するための文法 (grammer)。通常は、ある式が正しい式であるかどうかを判定するための手続きが必要とされる。
3. 論理式によって記述される公理 (axiom) あるいは型変数が入る公理型 (axiom schema)。型変数に代入可能な論理式や項の個数が可算無限であれば、公理型は可算無限の公理を表すことになる。
4. 論理式から他の論理式を導く推論規則 (rule of inference)。

ここで、形式体系では集合や自然数などといった、議論する対象である議論領域 (domain of discourse) が想定されていますが、形式体系は「記号、文法、公理、推論規則」のみの体系であって、議論領域は形式体系とは独立していると考えるようです。形式体系で使われている記号が「本当は何を意味するか」は形式体系は関知しないということです。そのようにすることで、1 つの形式体系が異なる議論領域を兼ねることができて便利な場合もあるためです。厳密な議論をするのでなければ、議論領域を形式体系に含めて、まとめて 1 つの形式体系と考えても支障はないでしょうから、本書ではそのあたりは適宜緩やかに考えます。

公理とすでに証明された定理から、推論規則によって導かれた論理式は定理となります。ある論理式を証明することは、その論理式が定理であることを導くこととも言えます。

5.3.2 一階述語論理

現代の数学で標準的な形式体系は述語論理 (predicate logic) です。述語論理の特徴は、論理式に含まれる変数を量化 (quantification) できるところにあります。量化とは、「全ての」とか「いくつかの」といったような量に

関する表現です。たとえば、「2 は自然数である」「3 は自然数である」という文章を主語と述語に分解すると、それぞれ「自然数である」が述語になります。このような述語を持っている文は「n は自然数である」と主語を変数 (variable) で書き換えることができて、それは n に関する条件文（真偽を判定できる文）になります。すると、その条件文が真となるような n の量について言及する量化した表現が生まれます。たとえば「全ての自然数 n に対して」とか「ある自然数 n に対して」といった文です。それが述語論理の特徴ということです。

量化は量化子 (quantifier) という記号を使って表現されます。量化子には「全ての〜に対して (for all)」を意味する全称記号 (universal quantifier) $\forall$ と「〜が存在する (exists)」を意味する存在記号 (existential quantifier)$\exists$ があります。

数学基礎論では、主に一階述語論理 (first-order predicate logic) をベースとして数学の理論体系を形式化します。一階述語論理は、変数の量化のみを許す述語論理で、変数の量化に加えて述語や関数の量化を許す述語論理は二階述語論理 (second-order predicate logic) です。それでは一階述語論理について説明します。

1. 記号は、論理記号と非論理記号に分かれます。論理記号 (logical symbol) は、

   - 量化子 $\forall$ と $\exists$
   - 論理演算 $\wedge$（論理積)、$\vee$（論理和)、$\neg$（否定)、$\rightarrow$（論理包含)、$\leftrightarrow$（同値）
   - 括弧 ( )
   - 無限個の変数 $x, y, z, ...$
   - 等号 $=$（含まない場合もある）

   です。ここで、これらすべての記号が必要ではなく、たとえば $\neg, \rightarrow, \forall$ だけを定義すれば他の量化子と論理演算はすべてその記号のみで書き換え可能なため、必要な記号だけで代表させる場合もあります。等号の取り扱いにはいくつかの流儀があり、論理記号に等号を含む場合は

「等号付き一階述語論理」です。等号を含まない場合は「等号無し一階述語論理」で、その場合は非論理記号として等号を導入します。

非論理記号 (non-logical symbol) は、解釈によって意味が変わる述語記号と関数記号です。定数記号は 0 変数関数なので関数記号に含まれます。そして、非論理記号に何を使うかで一階述語論理の言語が決まります。たとえば集合を議論領域とするときには、非論理記号として述語記号 $\in$ を使います。非論理記号をシグネチャ (signature) としてまとめて表記することがあります。

2. 記号から論理式を構成するための構成規則 (formation rule) すなわち文法は、このようなものです。

   まずは項 (term) を次のように帰納的に定義します。(1) 変数は項です。(2)n 変数関数 f と項 $t_1, ..., t_n$ に対して、$ft_1...t_n$ は項です。定数記号は 0 変数関数なので項となります。

   次に、n 変数の述語記号 P と n 個の項 $t_1, ..., t_n$ に対して、$Pt_1...t_n$ は原子論理式 (atomic formula) であると定義します。

   そして、論理式 (formula) は次のように帰納的に定義されます。(1) 原子論理式は論理式です。(2) ψ と ϕ を論理式、x を変数記号として、

   $$\neg\psi, \psi \wedge \phi, \psi \vee \phi, \psi \to \phi, \psi \leftrightarrow \phi, \forall x\psi, \exists x\psi$$

   は論理式です。

3. 公理には論理公理と非論理公理があります。

   論理公理 (logical axiom) の選び方にはいろいろな流儀があるようですが、その一例を示します。

   (a) トートロジー (tautology) すなわち恒真命題（真である、という論理式）

   (b) 命題論理の公理として、以下のウカシェビッチ公理系 (Łukasievicz's axiom system) （ϕ, ψ, θ には任意の論理式が入るため、無限の公理を含む公理型となる）

   - $\phi \to (\psi \to \phi)$

- $(\phi \to (\psi \to \theta)) \to ((\phi \to \psi) \to (\phi \to \theta))$
- $(\neg\phi \to \neg\psi) \to (\psi \to \phi)$

(c) 一階述語論理の公理

- 等式の公理 (axiom of equality): 変数 x について $x = x$
- 普遍例化の公理型 (axiom scheme for universal instantiation): ϕ_t^x を ϕ の変数 x に項を代入した論理式として、$\forall x\phi \to \phi_t^x$
- 存在一般化の公理型 (axiom scheme for existential generalization): $\phi_t^x \to \exists x\, \phi$

非論理公理 (non-logical axiom) は、議論領域に関する公理です。たとえば集合を議論領域とするときには、 公理的集合論における **ZF** 公理系か、それに選択公理を加えた **ZFC** 公理系が使われます。自然数を議論領域とするときに、次に説明するロビンソン算術やペアノ算術、後に説明する **2** 階算術のような理論体系が選ばれると、それに応じた公理系が設定されます。

4. 推論規則にはいろいろなものがありますが、よく使われる方法はモーダス・ポーネンス (modus ponens) と全称化 (generalization) の 2 つを推論規則とします。モーダス・ポーネンスは、ψ と $\psi \to \phi$ から ϕ を導出するという規則です。全称化は、x が変数のとき ψ から $\forall x\psi$ を導出するという規則です。

5.3.3 算術の理論体系

自然数と演算を議論領域とした一階述語論理として、いくつかの算術の理論体系があります。その中でも基本的なものとして、ロビンソン算術 (Robinson arithmetic) Q とペアノ算術 (Peano arithmetic) PA について説明します。

1891 年にイタリアの数学者ジュゼッペ・ペアノ (Giuseppe Peano, 1858–1932) が自然数を公理化した ペアノの公理[23] (Peano axioms) を作りまし

[23] http://mathworld.wolfram.com/PeanosAxioms.html

た。自然数と演算（加算と乗算）を議論領域とした一階述語論理に、非論理的公理としてペアノの公理を入れたものがペアノ算術 PA で、ペアノ算術から帰納法を除いたものがロビンソン算術 Q です。Q と PA で使う非論理記号は、定数記号 0、関数記号 S（後者関数）と $+$（加算）と $\cdot$（乗算）です。論理記号には等号 $=$ を含めます。Q の非論理公理は、以下の 7 つです。

1. $\forall x \neg(S(x) = 0)$ （0 は他の数の後者ではない。）
2. $\forall x \forall y(S(x) = S(y) \to x = y)$ （x と y の後者が等しければ、x と y は等しい。）
3. $\forall x(\neg(x = 0) \to \exists y(x = S(y)))$ （任意の数は 0 であるか別の数の後者である。）
4. $\forall x(x + 0 = x)$
5. $\forall x \forall y(x + S(y) = S(x + y))$ （4 と 5 で加算を帰納的に定義）
6. $\forall x(x \cdot 0 = 0)$
7. $\forall x \forall y(x \cdot S(y) = (x \cdot y) + x)$ （6 と 7 で乗算を帰納的に定義）

そして、PA の非論理公理は、Q の公理に次の帰納法の公理を加えたものです（ただし、Q の 3 番目の公理は、帰納法の公理から証明できるので不要）。

帰納法の公理: $\forall \vec{y}[\phi(0, \vec{y}) \wedge \forall x(\phi(x, \vec{y}) \to \phi(S(x), \vec{y}))) \to \forall x \phi(x, \vec{y})]$

ここで、$\phi(x, \vec{y})$ は PA の任意の論理式で、$\vec{y} = y_1, y_2, , ..., y_k (k \geq 0)$ です。この公理は、いわゆる数学的帰納法を意味しています。論理式 $\phi(x, \vec{y})$ の取り方は任意なので、PA は Q に無限に多くの公理を加えた体系になります。つまり、一階述語論理では ϕ を量化して $\forall \phi$ のような表記ができないので、個々の論理式がすべて公理となっている、と考えています。また、ペアノの公理には加算と乗算の定義はなく、後者から帰納法によって定義しますが、それには二階述語論理を使うため、一階述語論理の上に構築されている PA には、Q と同様に加算と乗算の公理が入っています。

Q では $3 \cdot 2 + 1 = 7$ のような数値に関する命題は証明できますが（ここで、自然数は $0, S(0), S(S(0)), ...$ を略記したものと考える）、変数が入った命題については証明できないものが多く、PA では $\forall x(0 + x = x)$ のよう

な、Q では証明ができない変数の入った多くの命題が証明できます。ただし不完全性定理があるため、自然数のすべての性質が PA で証明できるわけではありません。

このように Q よりも PA の方が多くの命題を証明できますので、Q よりも PA の方が証明論的に強い理論体系、つまり証明力が強い理論体系ということになります。

5.3.4 順序数解析

Q よりも PA の方が証明力が強い、ということを説明しました。それでは、証明力の強さを比較するための物差しはあるでしょうか。実は、順序数が良い物差しになります。そして、ある理論体系の証明力の強さを順序数と関連づける方法が順序数解析です。

理論体系 T の証明論的順序数 (proof theoretic ordinal) は、その理論では整礎であることが証明できないような最小の順序数です。たとえば、PA の証明論的順序数は ϵ_0 です。したがって、PA では ϵ_0 よりも小さい順序数 α が整礎であること、つまり α からの無限降下列を作ることができないことを証明できますが、ϵ_0 からの無限降下列を作ることができないことの証明はできません。

PA では帰納法が使えましたが、それを順序数のような整礎集合に拡張したものが、超限帰納法 (transfinite induction) です。順序数 α に対して命題 $P(\alpha)$ を考えます。

> 「すべての $\beta < \alpha$ に対して $P(\beta)$ が真であれば、$P(\alpha)$ は真である」ことが示されれば、P はすべての順序数で真になる

とするのが超限帰納法です。

超限帰納法が使えれば、ϵ_0 以下のすべての順序数が整礎であることから、超限帰納法によって ϵ_0 が整礎であると証明できます。しかし PA には超限帰納法がないため、その証明ができません。PA よりも強い理論体系であればその証明ができるものがあり、その理論体系にはやはりまた証明論的順序数 α が存在します。

順序数解析の研究がどのように進められてきたか、新井 (2005) [24] は以下のように解説しています。PA の証明論的順序数が ϵ_0 であることの証明を与えたのは、ドイツの論理学者・数学者のゲルハルト・ゲンツェン (Gerhard Gentzen, 1909–1945) です。まずは PA における証明から、推論規則の流れを図示した証明図を書きます。その証明図には帰納法の公理が含まれていますが、帰納法の公理は実際には無限の公理が含まれている公理型です。それを有限の公理に書き換える操作と、カット除去という操作を繰り返すことで、論理的な証明図になります。ゲンツェンは、PA の証明を論理的な証明図にするためには、ϵ_0 までの帰納法が必要であることを示しました。

その後、竹内外史は ϵ_0 よりも大きな順序構造を証明図において見出して、証明論における順序数図形 (ordinal diagram) を作りました。その証明図の限界が、後の章で紹介する「竹内・フェファーマン・ブーフホルツ順序数」(p.186) と名付けられる順序数です。

このように、ϵ_0 よりも大きい順序数は、ある理論体系における証明論的順序数を解析するために研究をされてきた面もあるようです。

[24] 新井敏康 (2005) 「Hilbert の第 2 問題に関する証明論の展開」数学 57(2): 113-126. doi:10.11429/sugaku1947.57.113

第 6 章
ペアノ算術の限界

本章では $F[\epsilon_0]$ 以下の大きさの再帰関数について扱います。ペアノ算術の証明論的順序数は ϵ_0 であり、$F[\epsilon_0]$ はペアノ算術で全域性（定義域がすべての自然数になること、つまりすべての自然数 n に対して $f(n)$ が計算されること）を証明可能ないかなる関数も支配することが証明されています。そして、$F[\epsilon_0]$ の全域性はペアノ算術では証明できません。つまり、$F[\epsilon_0]$ はペアノ算術で定義できる関数の増加速度の限界です。本章では、その限界に到達した関数を紹介します。

6.1 $F[\epsilon_0](n)$ の計算

まずは、ワイナー階層で $F[\epsilon_0](n)$ を計算してみます。

$$
\begin{aligned}
F[\epsilon_0](0) &= F[1](0) = H[\omega](0) = H[0](0) = 0 \\
F[\epsilon_0](1) &= F[\omega](1) = F[1](1) = H[\omega](1) \\
&= H[1](1) = 2 \\
F[\epsilon_0](2) &= F[\omega^\omega](2) = F[\omega^2](2) = F[\omega\omega](2) \\
&= F[\omega 2](2) = F[\omega+\omega](2) \\
&= F[\omega+2](2) (\approx A(1,2,2)) \\
&= F[\omega+1]^2(2) = F[\omega+1](F[\omega+1](2)) \\
&= F[\omega+1](F[\omega](F[\omega](2)))
\end{aligned}
$$

$$
\begin{aligned}
&= F[\omega+1](F[\omega](F[2](2))) \\
&= F[\omega+1](F[\omega](8)) \\
&= F[\omega+1](F[8](8)) \\
&\approx F[\omega+1](2\uparrow^7 8) \\
&\approx A(1,1,2\uparrow^7 8) \\
F[\epsilon_0](3) &= F[\omega^{\omega^\omega}](3) = F[\omega^{\omega^3}](3) \\
&= F[\omega^{\omega^2\omega}](3) = F[\omega^{\omega^2 3}](3) \\
&= F[\omega^{\omega^2 2+\omega\omega}](3) \\
&= F[\omega^{\omega^2 2+\omega 2+3}](3) \\
&= F[\omega^{\omega^2 2+\omega 2+2}\omega](3) \\
&= F[\omega^{\omega^2 2+\omega 2+2}3](3) \\
&= F[\omega^{\omega^2 2+\omega 2+2}2+\omega^{\omega^2 2+\omega 2+2}](3) \\
&= F[\omega^{\omega^2 2+\omega 2+2}2+\omega^{\omega^2 2+\omega 2+1}3](3) \\
&= F[\omega^{\omega^2 2+\omega 2+2}2+\omega^{\omega^2 2+\omega 2+1}2+\omega^{\omega^2 2+\omega 2+1}](3) \\
&= F[\omega^{\omega^2 2+\omega 2+2}2+\omega^{\omega^2 2+\omega 2+1}2+\omega^{\omega^2 2+\omega 2}3](3) \\
&= F[\omega^{\omega^2 2+\omega 2+2}2+\omega^{\omega^2 2+\omega 2+1}2+\omega^{\omega^2 2+\omega 2}2+\omega^{\omega^2 2+\omega 2}](3) \\
&= F[\omega^{\omega^2 2+\omega 2+2}2+\omega^{\omega^2 2+\omega 2+1}2+\omega^{\omega^2 2+\omega 2}2+\omega^{\omega^2 2+\omega+\omega}](3) \\
&= F[\omega^{\omega^2 2+\omega 2+2}2+\omega^{\omega^2 2+\omega 2+1}2+\omega^{\omega^2 2+\omega 2}2+\omega^{\omega^2 2+\omega+3}](3) \\
&= F[\omega^{\omega^2 2+\omega 2+2}2+\omega^{\omega^2 2+\omega 2+1}2+\omega^{\omega^2 2+\omega 2}2+\omega^{\omega^2 2+\omega+2}3](3) \\
&= F[\omega^{\omega^2 2+\omega 2+2}2+\omega^{\omega^2 2+\omega 2+1}2+\omega^{\omega^2 2+\omega 2}2+\omega^{\omega^2 2+\omega+2}2 \\
&\quad +\omega^{\omega^2 2+\omega+1}2+\omega^{\omega^2 2+\omega}2+\omega^{\omega^2 2+2}2+\omega^{\omega^2 2+1}2+\omega^{\omega^2 2}2 \\
&\quad +\omega^{\omega^2+\omega 2+2}2+\omega^{\omega^2+\omega 2+1}2+\omega^{\omega^2+\omega 2}2+\omega^{\omega^2+\omega+2}2 \\
&\quad +\omega^{\omega^2+\omega+1}2+\omega^{\omega^2+\omega}2+\omega^{\omega^2+2}2+\omega^{\omega^2+1}2+\omega^{\omega^2}2 \\
&\quad +\omega^{\omega 2+2}2+\omega^{\omega 2+1}2+\omega^{\omega 2}2+\omega^{\omega+2}2+\omega^{\omega+1}2+\omega^{\omega}2 \\
&\quad +\omega^2 2+\omega 2+3](3)
\end{aligned}
$$

このように、$F[\epsilon_0](3)$ の時点で絶望的に計算に終わりが見えないことが分かります。

6.2 ドル関数の角括弧表記

Googology Wiki ユーザーの Wythagoras が 2013 年に開発したドル関数[1] (dollar function) は、最も基本的な表記である角括弧表記 (bracket notation) が比較的分かりやすい定義で $F[\epsilon_0]$ の強さを持っています。

【定義】ドル関数の角括弧表記

X は何でも良い。
a は\$の前の数である。

1. $a\$ = a$ (\$の後に何もないとき)
2. $a\$Xb = (a+b)\$X$
3. $[Xb] = [Xb-1][Xb-1]...[Xb-1][Xb-1]$ (a 個の X)
4. $X0 = X$ (X が空でないとき)
5. $[0] = a$

右から左へ向かって数を探して、最初に見つかった数をアクティブな数とし、その数の右にある角括弧をアクティブな角括弧とする。ルール 3 以降についてはアクティブな数と角括弧に対して計算を実行する。

この計算ルールにしたがって計算をすると、

$$\begin{aligned}
a\$[0] &= a\$a = 2a\$ = 2a = F[1](a)\\
a\$[1] &= a\$\underbrace{[0][0]...[0]}_{a} = 2^a a = F[2](a)\\
a\$[2] &= a\$\underbrace{[1][1]...[1]}_{a} = F[3](a)\\
a\$[b] &= F[b+1](a)\\
a\$\,[[0]] &= a\$[a] > F[a](a) = F[\omega](a)\\
a\$[[0]1] &= a\$\underbrace{[[0]][[0]]...[[0]]}_{a} > F[\omega+1](a)
\end{aligned}$$

[1] 巨大数研究 Wiki - ドル関数 http://ja.googology.wikia.com/wiki/ドル関数

$$
\begin{aligned}
a\$[[0]2] &= a\$\underbrace{[[0]1][[0]1]...[[0]1]}_{a} > F[\omega+2](a)\\
a\$[[0][0]] &= a\$[[0]a] > F[\omega\times 2](a)\\
a\$\,[[1]] &= a\$\underbrace{[[0][0]...[0]]}_{a} > F[\omega^2](a)\\
a\$[[1]1] &= a\$\underbrace{[[1]][[1]]...[[1]]}_{a} > F[\omega^2+1](a)\\
a\$[[1][0]] &= a\$[[1]a] > F[\omega^2+\omega](a)\\
a\$[[1][0][0]3] &> F[\omega^2+\omega\times 2+3](a)\\
a\$[[1][1]] &> F[\omega^2\times 2](a)\\
a\$\,[[2]] &= a\$\underbrace{[[1][1]...[1]]}_{a} > F[\omega^3](a)\\
a\$[[b]c] &> F[\omega^{b+1}+c](a) > F[\omega^b+c](a)\\
a\$[[[0]]] &= a\$[[a]] > F[\omega^\omega](a)\\
a\$[[[b]c]d] &> F[\omega^{\omega^{b+1}+c}+d](a) > F[\omega^{\omega^b+c}+d](a)\\
a\$\,[[[[0]]]] &> F[\omega^{\omega^\omega}](a)\\
a\$\,[[[[[0]]]]] &> F[\omega^{\omega^{\omega^\omega}}](a)
\end{aligned}
$$

といったように、角括弧の入れ子を 1 つ増やすと ω の指数タワーが 1 つ増えるので、これは $F[\epsilon_0]$ の強さを持つ関数となります。さらに、拡張角括弧表記では各括弧に下付き文字のレベルをつけて、たとえば

$$a\$[[0]_2] = a\$\,[[...[[0]]...]]\,(a+1\text{ 重の角括弧})$$

のようになります（レベルの展開方法の定義は省略します)。このことを使うと、

$$a\$[[0]_2] \quad\approx\quad F[\varepsilon_0](a)$$

と $F[\epsilon_0]$ の強さを持つ関数を簡潔に書くことができます。

このように、ドル関数は拡張角括弧表記、線形配列表記とさらに大きな関数が定義されていますが、本書で取り上げるのは角括弧表記までとします。

6.3 ふぃっしゅ数バージョン5

バージョン3の次は、バージョン4ではなくバージョン5の説明をします。巨大数探索スレッドでは、バージョン4が先に登場するのですが、本書では章を進めるにしたがってより大きな数が登場するように記述しているため、バージョン5よりも大きいバージョン4の説明は後回しにします。

「関数から関数への写像を考える」という考えをさらにすすめると、「関数から関数への写像」から「関数から関数への写像」への写像を考えることができます。では、このような写像に対して写像を適用する回数の数え上げを定義することはできないでしょうか。このような考察を進めた結果、以下の様なふぃっしゅ数バージョン5が定義されました[2]。

【定義】ふぃっしゅ数バージョン5

1. 集合 $M_n(n=0,1,2,...)$ を以下のように定める。

   M_0=自然数の集合

   M_{n+1}=写像 $M_n \to M_n$ 全体の集合

   M_n の元を M_n 変換と呼ぶ

2. M_n 変換 $m(n)(n \geq 1)$ を以下のように定める。

   $f_n \in M_n$ に対して、$m(n+1)(f_n)=g_n$ を以下で定める。

   $f_{n-1} \in M_{n-1}$ に対して、$g_n(f_{n-1})=g_{n-1}$ を以下で定める。

   $f_{n-2} \in M_{n-2}$ に対して、$g_{n-1}(f_{n-2})=g_{n-2}$ を以下で定める。

   $\vdots$

   $f_0 \in M_0$ に対して、$g_1(f_0)=g_0$ を以下で定める。

   g_0=$(..((f_n^{f_0} f_{n-1})f_{n-2})...f_1)f_0$

[2] http://ja.googology.wikia.com/wiki/ふぃっしゅ数バージョン5

すなわち

$$\begin{aligned} m(1)(f_0) &= {f_0}^{f_0} \\ (m(2)f_1)f_0 &= ({f_1}^{f_0})(f_0) \\ (..((m(n+1)f_n)f_{n-1})...f_1)f_0 &:= (..({f_n}^{f_0}f_{n-1})...f_1)f_0 \end{aligned}$$

3. ふぃっしゅ関数 $F_5(x)$ を以下のように定める。

$$F_5(x) := ((..((m(x)m(x-1))m(x-2))...m(2))m(1))(x)$$

4. ふぃっしゅ数 $F_5 := F_5^{63}(3)$ とする。

定義 2 より、$m(1)(x) = x^x$ です。また、

$$(m(2)f)(x) = (f^x)(x)$$

となりますから、$m(2)$ は F_3 における $s(1)$ の定義と一致し、

$$m(2) = s(1)$$

となります。さらに、

$$\begin{aligned} ((m(3)m(2))f)(x) &= (m(2)^x f)(x) \\ &= (s(1)^x f)(x) \\ &= (s(2)f)(x) \end{aligned}$$

すなわち、$m(3)m(2) = s(2)$

$$\begin{aligned} ((m(3)^2 m(2))f)(x) &= ((m(3)(m(3)m(2)))f)(x) \\ &= ((m(3)m(2))^x f)(x) \\ &= (s(2)^x f)(x) \\ &= (s(3)f)(x) \end{aligned}$$

すなわち、$m(3)^2 m(2) = s(3)$

$$m(3)^n m(2) = s(n+1)$$

$$m(4)m(3)m(2) \quad = \quad ss(1)$$

といったように、計算ができます。すなわち、$m(3)$ は操作の対角化で、$m(2) = s(1)$ の原始再帰に対して、$m(3)$ を繰り返し適用することで $m(3)^n m(2) = s(n+1)$ の多重再帰が生成されます。それを数え上げるのが $m(4)$ ということです。

急増加関数との対応を考えると、$m(2)$ の定義から $f(x) \approx F[\alpha](x)$ の時、$m(2)f(x) \approx F[\alpha+1](x)$ となりますが、これを $m(2) = F[+1]$ と便宜的に表記することとします。すると、

$$\begin{aligned}
m(3)m(2) &= s(2) = F[+\omega] \\
m(3)m(3)m(2) &= s(3) = F[+\omega^2] \\
m(3)^a m(2) &= s(a) = F[+\omega^a] \\
m(4)m(3)m(2) &= ss(1) = s(x) = F[+\omega^\omega]
\end{aligned}$$

となります。さらに、

$$\begin{aligned}
m(3)[m(4)m(3)]m(2) &= ss(2) = F[+\omega^{\omega+1}] \\
m(3)^2[m(4)m(3)]m(2) &= ss(3) = F[+\omega^{\omega+2}] \\
m(3)^a[m(4)m(3)]m(2) &= ss(a+1) = F[+\omega^{\omega+a}] \\
\big[m(4)m(3)\big]^2 m(2) &= ss(n) = F[+\omega^{\omega\times 2}]
\end{aligned}$$

ここまでが、F_3 の定義で記述できるところです。さらに、

$$\begin{aligned}
\big[m(4)m(3)\big]^2 m(2) &= F\big[+\omega^{\omega\times 2}\big] \\
\big[m(4)m(3)\big]^3 m(2) &= F\big[+\omega^{\omega\times 3}\big] \\
\big[m(4)m(3)\big]^a m(2) &= F\big[+\omega^{\omega\times a}\big] \\
\big[m(4)^2 m(3)\big]m(2) &= F\big[+\omega^{\omega^2}\big] \\
m(3)\big[m(4)^2 m(3)\big]m(2) &= F\big[+\omega^{\omega^2+1}\big] \\
\big[m(4)m(3)\big]\big[m(4)^2 m(3)\big]m(2) &= F\big[+\omega^{\omega^2+\omega}\big] \\
\big[m(4)m(3)\big]^2\big[m(4)^2 m(3)\big]m(2) &= F\big[+\omega^{\omega^2+\omega\times 2}\big] \\
\big[m(4)m(3)\big]^3\big[m(4)^2 m(3)\big]m(2) &= F\big[+\omega^{\omega^2+\omega\times 3}\big]
\end{aligned}$$

$$
\begin{aligned}
\big[m(4)^2m(3)\big]^2m(2) &= F\big[+\omega^{\omega^2\times 2}\big](x)\\
\big[m(4)^2m(3)\big]^3m(2) &= F\big[+\omega^{\omega^2\times 3}\big](x)\\
m(4)^3m(3)m(2) &= F\big[+\omega^{\omega^3}\big]\\
m(4)^4m(3)m(2) &= F\big[+\omega^{\omega^4}\big]\\
m(5)m(4)m(3)m(2) &= F\big[+\omega^{\omega^\omega}\big]
\end{aligned}
$$

このように計算されます。

これまでの計算から、

$$
\begin{aligned}
m(3)m(2) &= F[+\omega]\\
m(4)m(3)m(2) &= F[+\omega^\omega]\\
m(5)m(4)m(3)m(2) &= F[+\omega^{\omega^\omega}]
\end{aligned}
$$

となり、同様に

$$
\begin{aligned}
m(6)m(5)m(4)m(3)m(2) &= F[+\omega\hat{}\hat{}4]\\
m(7)m(6)m(5)m(4)m(3)m(2) &= F[+\omega\hat{}\hat{}5]\\
&\vdots
\end{aligned}
$$

となることが期待されます。このとき、

$$F_5(x) \approx F[\epsilon_0]$$

となります。したがって、

$$F_5 \approx F[\epsilon_0+1](63)$$

となります。

6.4　ヒドラゲーム

本節ではヒドラゲームについて記します。その前に、グッドスタイン数列の話の続きをします。グッドスタイン数列の話 (2.2 節、p.57) が、ずっ

と放置されてきました。ようやく続きを話すことができます。n で始まるグッドスタイン数列が終了するまでのステップ数を返すグッドスタイン関数 $G(n)$ の大きさは、なんと $H[\epsilon_0]$ のレベルなのです。数列の定義は原始再帰なのに、関数はこんなに大きくなる、実に面白い関数です。

グッドスタイン数列は急激に増加しますが、やがて減少に転じ必ず 0 となって数列が終わることが証明されています。これをグッドスタインの定理 (Goodstein's theorem) と言います[3]。なぜ、必ず 0 になるのでしょうか。それは次のように証明します。

グッドスタイン数列の遺伝的記法で、数の底のところを順序数 ω で書き換えます。このとき、n^k の和で書かれていればそのまま ω^k に書き換えるだけですが、$a_k n^k$ の和で書かれている場合は、$\omega^k \times a_k$ と書き換えます。

19 ではじまるグッドスタイン数列を、そのように書き換えてみます。

$$
\begin{aligned}
G_0(19) &= 2^{2^2} + 2 + 1 \to \omega^{\omega^\omega} + \omega + 1 \\
G_1(19) &= 3^{3^3} + 3 \to \omega^{\omega^\omega} + \omega \\
G_2(19) &= 4^{4^4} + 3 \to \omega^{\omega^\omega} + 3 \\
G_3(19) &= 5^{5^5} + 2 \to \omega^{\omega^\omega} + 2 \\
G_4(19) &= 6^{6^6} + 1 \to \omega^{\omega^\omega} + 1 \\
G_5(19) &= 7^{7^7} \to \omega^{\omega^\omega}
\end{aligned}
$$

対応する順序数を見ると、どんどん小さくなっています。これは、グッドスタイン数列における 2 つの演算のうち、底を変換する演算は底を ω にしてしまったため変わらず、1 を引く演算によって、より小さい順序数となるためです。$G_6(19)$ については、非常に長い順序数の式が並ぶことになりますが、$\Sigma(\omega^{\Sigma(\omega^a \times b)} \times c)$ の形に書けます。もともとの数列を n^k の和の形で書いていれば、$\Sigma\omega^{\Sigma\omega^a}$ の形になり、ω^{ω^ω} よりも小さくなっています。

順序数は整礎なので、無限降下列をとることができません。つまり、順序数の降下列は、必ず有限ステップで 0 になります。したがって、必ずグッドスタイン数列は有限ステップで値が 0 になって終了します。

[3] Goodstein, R. L. (1944). On the restricted ordinal theorem. *Journal of Symbolic Logic* 9 (2): 33-41. doi:10.2307/2266486

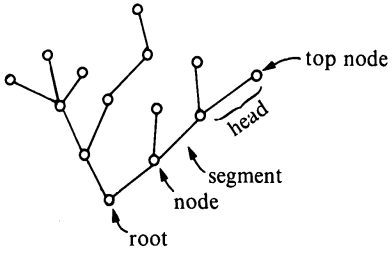

図 6.1: ヒドラの構造。Kirby and Paris (1982) から引用。

さて、このようにしてグッドスタインの定理が証明されましたが、一方でカービーとパリスは、ペアノ算術をもってしてはグッドスタインの定理は証明できない、ということを証明しました[4] 。ある特定のグッドスタイン数列が必ず 0 になる、ということは証明できますが、すべてのグッドスタイン数列が必ず 0 になる、ということを言うためには超限再帰が必要となり、ペアノ算術の理論体系だけでは証明ができないのです。

カービーらは論文でこの証明をするときに、**ヒドラゲーム** (Hydra battle) を考えました。ヒドラゲームとは、ギリシア神話に登場するヘラクレス (Hercules) とヒドラ (Hydra) の戦いにヒントを得たもので、ヒドラ（ヒュドラーとも書く）は巨大な胴体とたくさんの首を持ち、1 本の首を切り落としても、すぐにそこから新しい 2 本の首が生えてきます。

そこで次のようなヒドラを考えます。ヒドラは図 6.1 のように有限の樹の構造を持っていて、複数の**ノード** (node) が**セグメント** (segment) で結ばれています。樹の根元は**ルート** (root) と呼び、すべてのノードは下に 1 つずつセグメントを降りて行くと、必ずルートに到達します。そして一番上のノードを**トップノード** (top node) と呼び、トップノードにつながれているセグメントを**首** (head) と呼びます。

[4] Kirby, L. and Paris, J. (1982) Accessible independence results for Peano arithmetic. *Bulletin London Mathematical Society* 14(4): 285–293. doi:10.1112/blms/14.4.285

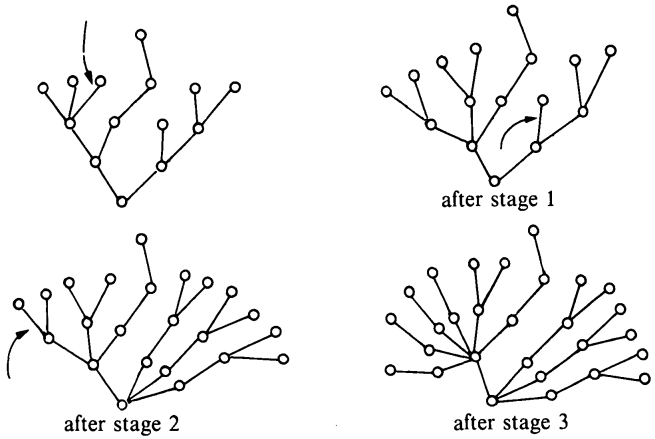

図 6.2: ヒドラゲームの進行。Kirby and Paris (1982) から引用。

ヒドラゲームはこのようなルールです。ヘラクレスは、ヒドラの首を 1 回につき 1 つ切り落とします。n 回目に切り落としたときに、ヒドラは n 個の頭が生えます。その生え方は、今、切り落とした首からセグメントを 1 つ下に降ります。そのセグメントから上に伸びている木の形と同じ形の木の構造が、今度はそこからさらに 1 つセグメントを下に降りたノードから n 個コピーされます。ここで、それ以上ノードが下がれないときには、首が生えません。

ヒドラゲームは図 6.2 のように進行します。図では切り落とす首が矢印で示されています。左上の図が最初のヒドラの構造で、首を切り落とすと、after stage 1 の形になります。これは切り落とした首の 1 つ下のノードから上を見たときの V 字構造が 1 つ、さらに 1 つ下のノードからコピーされました。次に、stage 2 では切り落とした首の 2 つ下のルートから、今度は 2 つコピーされます。次は 3 つ、次は 4 つとコピーされる数が増えるので、ヒドラはどんどん増殖します。

ヘラクレスは有限の回数首を切ることで、ルート以外のすべてを切り落とすことができればこのゲームに勝ちます。どのような順番で首を切れば、勝つことができるでしょうか。実はどの順番で切っても勝つことが分かり

ます。どうしてでしょうか。

ヒドラの構造で、各ノードを順序数に割り当てます。トップノードには 0 を、他のノードには、その上のノードに割り当てられる順序数 $\alpha_1 \geq \cdots \geq \alpha_n$ に対して、$\omega^{\alpha_1} + \cdots \omega^{\alpha_n}$ を割り当てます。図 6.1 のヒドラの構造図で、一番左上のノードから下に降りると、$0 \to 3 \to \omega^\omega + \omega^3 \to \omega^{\omega^\omega + \omega^3} + \omega^{\omega^2+1}$ のように割り当てられます（カントールの標準形に準じて大きい順序数から順に足すこととします)。このとき、ルートに割り当てられた順序数が、ヒドラに対応づけられる順序数です。

ヘラクレスがどのような順番で首を切っても、必ずヒドラの順序数は小さくなります。グッドスタイン数列のときには、順序数が小さくなる道順が一つに決まりましたが、ヒドラゲームでは自由に順路を選ぶことができます。グッドスタイン数列もヒドラゲームもともに、最初は爆発的に増加するけれど最後はなくなってしまうこと、順序数と対応付けると順序数が小さくなること、といった意味で共通しています。

カービーとパリスは、ヒドラゲームと対応づけながら、$G(n)$ が $H[\epsilon_0]$ と同じだけの増加速度を持つこと、もっと正確には、ϵ_0 よりも小さい全ての順序数 α に対して、グッドスタイン関数 $G(n)$ が $H[\alpha]$ よりも増加速度が速いことを証明しました。

グッドスタイン関数については、Caicedo (2007) [5] が、

$$n = 2^{m_1} + 2^{m_2} + \cdots + 2^{m_k} (m_1 > m_2 > \cdots > m_k)$$

に対して

$$G(n) = F[R_2^\omega(m_1)](F[R_2^\omega(m_2)](\cdots(F[R_2^\omega(m_k)](3))\cdots)) - 2$$

となることを示しました。ここで、$R_2^\omega(n)$ は n を 2 を底とした遺伝的記法で書いて、2 を ω に変えたものです。したがって、たとえば $n = 13 = 2^3 + 2^2 + 2^0$ に対して

$$\begin{aligned} G(13) &= F[R_2^\omega(3)](F[R_2^\omega(2)](F[R_2^\omega(0)](3))) - 2 \\ &= F[\omega + 1](F[\omega](F[0](3))) - 2 \end{aligned}$$

[5] Caicedo, A. (2007) Goodstein's function. *Revista Colombiana de Matemáticas* 41 (2): 381–391. http://ref.scielo.org/2c53my

$$\begin{aligned} &= F[\omega+1](F[\omega](4))-2 \\ &= F[\omega+1](F[4](4))-2 \end{aligned}$$

$n = 2 \uparrow\uparrow 5 = 2^{65536}$ に対しては

$$\begin{aligned} G(2^{65536}) &= F[R_2^{\omega}(2^{2^{2^2}})](3)-2 \\ &= F[\omega^{\omega^{\omega\omega}}](3)-2 \\ &= F[\omega^{\omega^{\omega^3}}](3)-2 \end{aligned}$$

のように計算されます。

ふぃっしゅ数バージョン 5 の $m(n)$ 変換の構造は、ヒドラゲームと対応する順序数の関係に対応します。$m(2) \to m(3) \to m(4)$ と操作の対角化を繰り返すことは、ヒドラのセグメントを上に上がって行くことに相当します。下から数えて n 番目のノードが、$m(n)$ に対応します。F_5 と急増加関数の対応（p. 153）で示した

$$[m(4)m(3)]^3[m(4)^2m(3)]m(2) \approx F[+\omega^{\omega^2+\omega\times 3}]$$

の計算において、$[m(4)m(3)]^3$ の部分は $\omega \times 3$ に対応し、$m(4)^2$ は 2 に、$m(4)^2m(3)$ は ω^2 に対応しています。そしてハーディ階層によって

$$[m(4)m(3)]^3[m(4)^2m(3)]m(2)m(1) \approx H[\omega^{\omega^{\omega^2+\omega\times 3}}]$$

と近似できます。F_5 の定義では $m(1)(x) = x^x$ ですが、$m(1)(x) = x+1$ としても再帰の階層は変わらず、$m(1)(x) = x+1$ とした $m(n)$ 変換はハーディ階層とほとんど一致します。

図 6.1 のヒドラツリーでは、一番左上のトップノードはルートから上に 3 個上がったところなので $m(3)$ です。その下のノードは、上に $m(3)$ が 3 つついていて自身が $m(2)$ なので、$m(3)^3m(2)$ になります。その 1 つ下のノードは、$m(3)^3m(2)$ と $m(4)m(3)m(2)$ がついている $m(1)$ ノードなので $[m(3)^3m(2)][m(4)m(3)m(2)]m(1)$ です。そして一番下のルートノードは

$$[m(2)[m(3)^2m(2)]m(1)]\ [[m(3)^3m(2)][m(4)m(3)m(2)]m(1)]\ (x)$$

となります。この関数は対応する順序数のハーディ階層

$$H[\omega^{\omega^{\omega}+\omega^3}+\omega^{\omega^2+1}](x)$$

で近似できます。

6.5　原始数列数

原始数列数[6] (primitive sequence number) は、バシク[7] が考案した巨大数で、巨大数探索スレッド 10 に投稿された BASIC のプログラムによって計算される $F[\epsilon_0+1](10)$ 程度の大きさの数です。バシクは、原始数列数のプログラムに続いてペア数列数 (p.205) のプログラムを投稿しました。原始数列数を生み出すアルゴリズムを原始数列システム (primitive sequence system) と言い、ベクレミシェフの虫[8] (Beklemishev's worms) [9] とよく似ていて、$F[\epsilon_0](n)$ の強さです。

【定義】原始数列システム

非負整数のリスト $S=(S_0,S_1,\ldots,S_n)$ が原始数列である。非負整数 n から非負整数 $S[n]$ への関数 $S[n]$ を次のように定める。 $f(n)=n^2$ とおく。

- $()[n]=n$
- $S_n=0$ のとき、 $S[n]=(S_0,S_1,\ldots,S_{n-1})[f(n)]$
- $S_n>0$ のとき、数列の良い部分 g と悪い部分 b を次のように決める。ここで、S_n は良い部分にも悪い部分にも属さない。$i<n, S_i<S_n$ をみたす非負整数 i が存在するかどうかを考える。
  - i が存在しないときは $g=(S_0,\ldots,S_{n-1}), b=()$ とする。

[6] http://ja.googology.wikia.com/wiki/原始数列数
[7] http://ja.googology.wikia.com/wiki/ユーザー:BashicuHyudora
[8] http://ja.googology.wikia.com/wiki/ベクレミシェフの虫
[9] Beklemishev, L. D. The worm principle. *Logic Colloquium* 2002 (Münster, 2002), 75–95, Lecture Notes in Logic, 27, Association for Symbolic Logic, La Jolla, CA., USA, 2006.

- i が存在するときは $k = \max\{i; i < n, S_i < S_n\}$ とおき、$g = (S_0, \ldots, S_{k-1}), b = (S_k, \ldots, S_{n-1})$ とする。

そして、$S[n] = (g+\underbrace{b+b+\cdots+b+b}_{f(n)+1\text{ 個の }b})[f(n)]$ とする。ここで、+ は数列の連結であり、たとえば $(0,3,2)+(1,4,5) = (0,3,2,1,4,5)$ である。

ここで計算方法の理解のために計算例を示します。$f(n) = n^2$ とすると大変なので $f(n) = n$ とします。また、数列の「悪い部分」に下線を引きます。「良い部分」はその左です。

$$
\begin{aligned}
& (0,1,\underline{2},3)[2] \\
= \ & (0,\underline{1,2,2},2)[2] \\
= \ & (0,1,2,2,1,2,2,\underline{1,2},2)[2] \\
= \ & (0,1,2,2,1,2,2,1,2,1,2,\underline{1},2)[2] \\
= \ & (\underline{0,1,2,2,1,2,2,1,2,1,2,1,1},1)[2] \\
= \ & (0,1,2,2,1,2,2,1,2,1,2,1,1,0,1,2,2,1,2,2,1,2,1,2,1,1, \\
& \underline{0,1,2,2,1,2,2,1,2,1,2,1},1)[2]
\end{aligned}
$$

数列の最後が 0 のときは $f(n) = n+1$、それ以外は $S[n] = (g+\underbrace{b+\cdots+b}_{n\text{ 個の }b})[n]$ と定義を変えると、原始数列はハーディ階層と完全に一致します。まず、数列 (0) は $(0)[n] = n+1$ となるためハーディ階層では $H[1](n)$ に対応し、順序数 1 に対応します。そして数列 (0,0) は順序数 2 に対応します。数列の最後が 0 のときは $f(n) = n+1$ としたため、数列の最後の 0 が消えて $n+1$ が計算されることが、ハーディ階層の $H[\alpha+1](n) = H[\alpha](n+1)$ と対応しています。

最後が 0 以外のときは極限順序数と対応しています。たとえば (0,1) という数列を考えると、「良い部分」はなく、「悪い部分」は 0 となります。そして、悪い部分を n 個つなげることから、

$$
\begin{aligned}
(0,1)[1] &= (0)[1] = 1 \\
(0,1)[2] &= (0,0)[2] = 2
\end{aligned}
$$

$$\begin{aligned}(0,1)[3] &= (0,0,0)[3] = 3\\ (0,1)[4] &= (0,0,0,0)[4] = 4\end{aligned}$$

のように計算され、$1,2,3,4,5,...$ のような列ができ、これが ω の基本列と一致します。ハーディ階層の $H[\omega](n) = H[n](n)$ に対応させて書くと

$$H[\omega](n) = (0,1)[n] = \underbrace{(0,0,\ldots,0)}_{n}[n] = H[n](n)$$

のようになります。このように、数列の「悪い部分」が増えていく様子が、基本列を列記する形と一致します。同様に考えると、原始数列の計算がハーディ階層の計算と一致します。たとえば、$(0,1,2)[3] = (0,1,1,1)[3]$ という式の展開は $H[\omega^\omega](3) = H[\omega^3](3)$ という式の展開と一致し、それぞれが対応します。$(0,1,2)[3]$ と $H[\omega^\omega](3)$ の計算をそれぞれ進めていく過程を、原始数列とハーディ階層を対応づけながら示します。以下の式において、左辺と右辺がそれぞれ上から下へ計算が進められます。

$$\begin{aligned}(0,1,2)[3] &= H[\omega^\omega](3)\\ (0,1,1,1)[3] &= H[\omega^3](3)\\ (0,1,1,0,1,1,0,1,1)[3] &= H[\omega^2 3](3)\\ (0,1,1,0,1,1,0,1,0,1,0,1)[3] &= H[\omega^2 2+\omega 3](3)\\ (0,1,1,0,1,1,0,1,0,1,0,0,0)[3] &= H[\omega^2 2+\omega 2+3](3)\\ (0,1,1,0,1,1,0,1,0,1,0,0)[4] &= H[\omega^2 2+\omega 2+2](4)\\ (0,1,1,0,1,1,0,1,0,1,0)[5] &= H[\omega^2 2+\omega 2+1](5)\\ (0,1,1,0,1,1,0,1,0,1)[6] &= H[\omega^2 2+\omega 2](6)\\ (0,1,1,0,1,1,0,1,0,0,0,0,0,0)[6] &= H[\omega^2 2+\omega+6](6)\\ (0,1,1,0,1,1,0,1)[12] &= H[\omega^2 2+\omega](12)\\ (0,1,1,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0)[12] &= H[\omega^2 2+12](12)\\ (0,1,1,0,1,1)[24] &= H[\omega^2 2](24)\end{aligned}$$

原始数列の定義はハーディ階層とは異なりますが、ハーディ階層の繰り返しの回数を増やした構造なので、巨大数論的なスケールでは、ハーディ

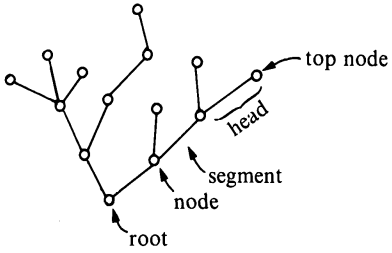

図 6.3: (図 6.1 再掲) ヒドラの構造。Kirby and Paris (1982) から引用。

階層で近似できると考えることができます。そしてハーディ階層で近似したときの順序数を α としたとき、原始数列そのものが順序数 α をあらわすものとします。このように考えると、原始数列は ϵ_0 よりも小さい順序数 α をもれなく対応づけることができます。

原始数列は、バシクがカントールの標準形、ヒドラゲーム、ふぃっしゅ数バージョン 5 などを参考にして作成したものなので、次のようにヒドラツリーの順序数と対応づけることができます。

1. 順序数 α のヒドラツリーを書く。
2. ルートノード (root) からスタートする。
3. ルートノードから上に 1 つノードを上がるときには、数列の最後に新しい要素 0 を加える。
4. ノードを 1 つ上に上がるときには、数列の最後に「最後尾の要素の値 + 1」の値の要素を加える。
5. ノードの端点まで来たので n 個下の枝分かれのノードまで下がり、別の枝 (segment) に進むときは、数列の最後に「最後尾の要素の値 $-n+1$」の値の要素を加える。
6. すなわち各項の数は対応するノードのルートノードを起点とした高さから 1 を引いた値となる。

たとえば図 6.1 のヒドラツリーでは、 $\omega^{\omega^{\omega}}$ の部分は $(0,1,2,3)$ というように表されます。 その後ノードを 3 本降りて、次の枝 (segment) へ移って $(1,2,2,2)$ を得て、列に追加されます。その後、同様にして $(0,1,2,2,1)$ が追加されます。

したがって図 6.1 のヒドラツリーは数列 $(0,1,2,3,1,2,2,2,0,1,2,2,1)$ と対応します。すなわち $\omega^{\omega^{\omega}+\omega^3}+\omega^{\omega^2+1}=(0,1,2,3,1,2,2,2,0,1,2,2,1)$ です。

このようにすると、次のように原始数列が順序数と対応付けられます。

$$
\begin{aligned}
1 &= (0)\\
2 &= (0,0)\\
3 &= (0,0,0)\\
\omega &= (0,1)\\
\omega+2 &= (0,1,0,0)\\
\omega\cdot 2 &= (0,1,0,1)\\
\omega^2 &= (0,1,1)\\
\omega^2+\omega &= (0,1,1,0,1)\\
\omega^3 &= (0,1,1,1)\\
\omega^{\omega} &= (0,1,2)\\
\omega^{\omega^{\omega}} &= (0,1,2,3)\\
\omega^{\omega^{\omega^{\omega}}} &= (0,1,2,3,4)\\
\omega^{\omega^{(\omega^{\omega}+1)}} &= (0,1,2,3,4,2)\\
\omega^{\omega^{\omega^{\omega^{\omega}}}} &= (0,1,2,3,4,5)
\end{aligned}
$$

したがって $f(n)=(0,1,2,...n)[n]$ とすれば $f(n)\approx H[\epsilon_0](n)\approx F[\epsilon_0](n)$ となり、$f^{10}(n)(10)\approx F[\epsilon_0+1](10)$ 程度の大きさの数をプログラムによって定義したものが原始数列数です。

この原始数列の定義を元に、巨大数研究 Wiki でミカヅキモが原始数列の定義の改良版を考えて、その定義の下で、原始数列 S を添え字に持つ急

増加関数 $F[S](n)$ を、順序数の知識を全く使わずに定義しました。以下にミカヅキモの定義をそのまま書きます。

【定義】原始数列の改良版

以下、+は数列の結合を意味する。

1. 空の列 () は原始数列である。
2. S を高々 1 個の 0 を含む原始数列とするとき、$w(S) = (0) + S'$ も原始数列である。ただし、S' は S の各成分に 1 を加えてできる数列を意味するものとする。($S = ()$ のときは、()' = () とする。)
3. $S_1, S_2, \cdots, S_n$ が原始数列であり、辞書式順序のもとで $S_1 \geq S_2 \geq \cdots \geq S_n$ であるとき (たとえば、$(0,1,2,3) \geq (0,1,2,2,2,0,1,2,3,4) \geq (0,1,2)$)、$S_1 + S_2 + \cdots + S_n$ も原始数列である。
4. 1〜3 によって原始数列であるとされるもののみが原始数列である。

w は順序数で言うと α を ω^α に移す写像 (1 項演算) に対応しており、ヒドラツリーで言うと根に接続されている辺に点を 1 つ加える操作に対応しています。+ は順序数で言うと順序数における +、ヒドラツリーで言うと n 個のヒドラツリーを横に並べる n 項演算に対応しています。

たとえば

$$
\begin{aligned}
(0) &= w(()) \\
(0,1) &= w((0)) = w(w(())) \\
(0,1,2,\cdots,n) &= w(w(\cdots(w(()))\cdots)) \ (w \text{ が } n-1 \text{ 個}) \\
(0,0) &= (0) + (0) \\
(0,1,1) &= w((0,0)) \\
(0,1,2,2) &= w((0,1,1)) \\
(0,1,2,3,1,2,2,0,0) &= (0,1,2,3,1,2,2) + (0,0)
\end{aligned}
$$

$$\begin{aligned} &= w((0,1,2,0,1,1)) + (0,0) \\ &= w((0,1,2) + (0,1,1)) + (0,0) \end{aligned}$$

などは原始数列です。

【定義】極限原始数列

原始数列 S の末尾の数が 0 でないとき、S は極限原始数列であるという。

【補題】

S が極限原始数列のとき、有限個の原始数列 $S_1, \cdots, S_k$ が存在して、$S = w(S_1) + \cdots + w(S_k)$ と表せる。しかもこの表し方は一意。

たとえば、$S = (0,1,2,3,1,2,2,2,0,1,2,2,1)$ は $S = w((0,1,2,0,1,1,1)) + w((0,1,1,0))$ と表せます。

【定義】極限原始数列の基本列

極限原始数列 S と非負整数 n が与えられたとき、原始数列 $S[n]$ を以下で帰納的に定める。$S[0], S[1], S[2], \cdots$ を S の基本列と呼ぶ。

1. $(w(S_1) + \cdots + w(S_k))[n] = w(S_1) + \cdots + w(S_{k-1}) + w(S_k)[n]$
2. $w(S + (0))[n] = w(S)n = w(S) + \cdots + w(S)$($n$ 個の $w(S)$)
3. $w(S)[n] = w(S[n])$　(S: 極限原始数列)

たとえば、$(0,1,2,2)[3] = (0,1,2,1,2,1,2)$ です。実際、

$$\begin{aligned} (0,1,2,2)[3] &= w((0,1,1))[3] = w((0,1,1)[3]) = w(w((0,0))[3]) \\ &= w(w((0) + (0))[3]) = w(w((0))3) = w((0,1)3) \\ &= w((0,1) + (0,1) + (0,1)) = w((0,1,0,1,0,1)) \\ &= (0,1,2,1,2,1,2) \end{aligned}$$

と計算できます。

【定義】改良版原始数列による急増加関数の定義

原始数列 S が与えられたとき、急増加関数 $F[S](n)$ を以下で定める。

1. $F[()](n) = n + 1$
2. $F[S + (0)] = F[S]^n(n)$
3. $F[S](n) = F[S[n]](n)$ （S: 極限原始数列）

これは ϵ_0 未満の順序数 α に対する急増加関数 $F[\alpha](n)$ と全く同じものです。

6.6 多重リストアッカーマン関数

たろうは、多変数アッカーマン関数を 2 重リストに拡張して、2 重リストアッカーマン関数を定義しました[10] 。多変数アッカーマンの n 個目の引数が a であることを $[n, a]$ と表記して、[ ] の中を多変数化することで拡張しました。これは $F[\omega^{\omega^{\omega}}](n)$ 程度の増大度の関数となります。

【定義】**2** 重リストアッカーマン関数

- □：0 個以上の 0
- X：0 個以上の 0 以上の整数
- Y, Y_1, Y_2：0 個以上の 0 以上の整数リスト
- a, b, c：0 以上の整数

十分大きな整数 N に対し、

$$\mathrm{index}[..., b_3, b_2, b_1, b_0, a_0] = ... + N^3 \cdot b_3 + N^2 \cdot b_2 + N \cdot b_1 + b_0$$

とし、Ak() の中のリストは index の大きい順に表記する。また、index

[10] http://ja.googology.wikia.com/wiki/2 重リストアッカーマン関数

が同じとなるリストは Ak() 内に複数含まないとする。

$$
\begin{aligned}
\mathrm{Ak}(Y_1,[X,0],Y_2) &= \mathrm{Ak}(Y_1,Y_2)\\
\mathrm{Ak}(Y_1,[\square,X],Y_2) &= \mathrm{Ak}(Y_1,[X],Y_2)\\
\mathrm{Ak}([a]) &= a+1\\
\mathrm{Ak}(Y,[1,b+1]) &= \mathrm{Ak}(Y,[1,b],[1])\\
\mathrm{Ak}(Y,[1,b+1],[a+1]) &= \mathrm{Ak}(Y,[1,b],[\mathrm{Ak}(Y,[1,b+1],[a])])\\
\mathrm{Ak}(Y,[\mathrm{X},c+1,b+1],[a]) &= \mathrm{Ak}(Y,[X,c+1,b],[X,c,a],[a])\\
&\quad \text{ここで } X\neq\square \text{ or } c\neq 0\\
\mathrm{Ak}(Y,[\mathrm{X},c+1,0,\square,b+1],[a]) &= \mathrm{Ak}(Y,[X,c+1,0,\square,b],\\
&\quad [X,c,a,\square,a],[a])
\end{aligned}
$$

以下のように計算されています。

$$
\begin{aligned}
\mathrm{Ak}([n,1],[n]) &\approx F[\omega^{\omega}](n)\\
\mathrm{Ak}([1,0,1],[n]) &\approx F[\omega^{\omega}](n)\\
\mathrm{Ak}([a,0,1],[n]) &\approx F[\omega^{(\omega\cdot a)}](n)\\
\mathrm{Ak}([n,0,1],[n]) &\approx F[\omega^{\omega^2}](n)\\
\mathrm{Ak}([1,0,0,1],[n]) &\approx F[\omega^{\omega^2}](n)\\
\mathrm{Ak}([a,0,0,1],[n]) &\approx F[\omega^{\omega^2\cdot a}](n)\\
\mathrm{Ak}([n,0,0,1],[n]) &\approx F[\omega^{\omega^3}](n)\\
\mathrm{Ak}([1,0,0,0,1],[n]) &\approx F[\omega^{\omega^3}](n)\\
\mathrm{Ak}([1,0,0,0,0,1],[n]) &\approx F[\omega^{\omega^4}](n)\\
\mathrm{Ak}([1,0,0,0,0,0,1],[n]) &\approx F[\omega^{\omega^5}](n)\\
\mathrm{Ak}(...,[3,a_3],[2,a_2],[1,a_1],[0,a_0]) &\approx \mathrm{Ak}(a_3,a_2,a_1,a0)\\
\mathrm{Ak}([...,b_3,b_2,b_1,b_0,a],[n]) &\approx F[\omega^{\cdots+\omega^3\cdot b_3+\omega^2\cdot b_2+\omega\cdot b_1+b_0}\cdot a](n)\\
\mathrm{Ak}([n\text{ 個の }1],[n]) &\approx F[\omega^{\omega^{\omega}}](n)
\end{aligned}
$$

さらに、同様の拡張を繰り返すことで多重リストアッカーマン関数を定

義できるとしました[11]。具体的な定義は与えられていませんが、その大きさは以下のように計算されています。

3 重リスト

$$\begin{aligned}
\mathrm{Ak}([[1,0,1],[1]],[[n]]) &\approx F[\omega^{\omega^{\omega}}](n)\\
\mathrm{Ak}([[1,0,0,1],[1]],[[n]]) &\approx F[\omega^{\omega^{\omega^2}}](n)\\
\mathrm{Ak}([[1,0,0,0,1],[1]],[[n]]) &\approx F[\omega^{\omega^{\omega^3}}](n)\\
\mathrm{Ak}([...,[3,a_3],[2,a_2],[1,a_1],[0,a_0]],[[n]]) &= \mathrm{Ak}([...,a_3,a_2,a_1,a_0],[n])\\
\mathrm{Ak}([[n\text{ 個の }1],[1]],[[n]]) &\approx F[\omega^{\omega^{\omega^{\omega}}}](n)
\end{aligned}$$

4 重リスト

$$\begin{aligned}
\mathrm{Ak}([[[1,0,1],[1]],[[1]]],[[[n]]]) &\approx F[\omega\text{^^}4](n)\\
\mathrm{Ak}([[[1,0,0,1],[1]],[[1]]],[[[n]]]) &\approx F[\omega^{\omega^{\omega^{\omega^2}}}](n)\\
\mathrm{Ak}([[[1,0,0,0,1],[1]],[[1]]],[[[n]]]) &\approx F[\omega^{\omega^{\omega^{\omega^3}}}](n)\\
\mathrm{Ak}([[...,[2,a_2],[1,a_1],[0,a_0]]],[[[n]]]) &= \mathrm{Ak}([[...,a_2,a_1,a_0]],[[n]])\\
\mathrm{Ak}([[[n\text{ 個の }1],[1]],[[1]]],[[[n]]]) &\approx F[\omega\text{^^}5](n)
\end{aligned}$$

n 重リストとすることで $F[\epsilon_0](n)$ に到達

6.7 バードのネスト配列表記

バードは、バードの線形表記 (p.107) の次に、$F[\omega^{\omega^{\omega}}]$ に相当するバードの多次元配列表記 (Bird's multi-dimensional array notation) と、$F[\omega^{\omega^{\omega^{\omega}}}]$ に相当するバードの超次元配列表記 (Bird's hyper-dimensional array notation) を定義しました。次に、超次元をネストするバードのネスト配列表記 (Bird's nested array notation) を考えて、これが $F[\epsilon_0]$ に到達しました。本書では詳

[11] http://ja.googology.wikia.com/wiki/多重リストアッカーマン関数

細な計算ルールは省きますが、次元セパレータ [ ] をネストさせます。バードの計算では

$$\begin{aligned}
[1\,[2]\,2] &= \omega^{\omega^{\omega}} \\
[1\,[3]\,2] &= \omega^{\omega^{\omega^2}} \\
[1\,[n+1]\,2] &= \omega^{\omega^{\omega^n}} \\
[1\,[1,2]\,2] &= \omega^{\omega^{\omega^{\omega}}} = \omega\text{^^}4 \\
[1\,[1\,[2]\,2]\,2] &= \omega\text{^^}5
\end{aligned}$$

といったように、入れ子（ネスト）のレベルが上がると ω^^n の n が上がって、入れ子レベルを対角化すると ϵ_0 になります。次元セパレータをネストさせる様子は、ちょうど多重リストアッカーマン関数のような構造となっています。

6.8　BEAF のテトレーション配列

BEAF の配列表記 (p.108) は、数を横一列に並べる一次元の配列でした。バウアーズは、これをさらに高次元の配列へと拡張しました。BEAF 入門[12]では、どのようにその拡張を考えていけば良いのかが、順序立てて解説されています。以下に BEAF の定義[13]を記します。まずは用語の定義です。

1. 「基数」 (b) は配列の 1 番目の要素である。
2. 「プライム」 (p) は配列の 2 番目の要素である。
3. 「パイロット」はプライムの次の最初の 1 ではない要素である。パイロットは 3 番目以降の要素となる。
4. 「副操縦士」はパイロットの 1 つ前の要素である。パイロットが行の中で 1 番目の要素であれば、副操縦士は存在しない。
5. 「構造」は配列の一部で、配列よりも低次元なグループによって構成されるものである。構造は、要素 (X^0 と書く)、行 (X^1 と書く)、平

[12]巨大数研究 Wiki - BEAF 入門 http://ja.googology.wikia.com/wiki/BEAF 入門
[13]巨大数研究 Wiki - BEAF http://ja.googology.wikia.com/wiki/BEAF

面 (X^2)、3 次元の領域 (X^3)、4 次元のフルーン (X^4)、さらに高次元の構造 (X^5, X^6 等)、そして $X \uparrow\uparrow 3$ のようなテトレーション構造、といった可能性がある。さらに、そこから先はペンテーション構造、ヘキセーション構造、... と続く。

6. 「前の要素」は、パイロットと同じ行にあり、パイロットよりも前にある要素である。「前の行」は、パイロットと同じ平面にあり、パイロットよりも前にある行である。「前の平面」は、パイロットと同じ領域にあり、パイロットよりも前にある平面である。同様に定義を続けることができる。これらは「前の構造」と呼ばれる。

7. 構造 S の「プライムブロック」は、構造を表記する記号の X をすべて p に置き換えたものである。たとえば、もし $S = X^3$ であれば、プライムブロックは p^3、すなわち一片の長さが p の立方体となる。X^X 構造のプライムブロックは p^p、すなわち一片が p の p 次元超立方体となる。

8. 「飛行機」は、パイロットと、すべての前の要素と、すべての前の構造のプライムブロックを含んだものである。

9. 「乗客」は飛行機の中のパイロットと副操縦士以外の要素である。

10. 配列 A の値は $v(A)$ と表記する。

そして、次のように計算ルールが定められます。

BEAF の計算ルール

1. プライムルール: もし $p = 1$ であれば、 $v(A) = b$ とする。
2. 初期ルール: もしパイロットがなければ、 $v(A) = b^p$ とする。
3. 破滅ルール: 1 も 2 もあてはまらない場合には、次のようにする。
   (a) パイロットの値を 1 減らす。
   (b) 副操縦士の値を元の配列のプライムを 1 減らしたものに置き換える。

(c) すべての乗客を b にする。

(d) 配列のそれ以外の要素は変化しない。

配列表記のルールは、このルールと一致することを確認してみてください。ここで、BEAF では有限長の配列は、その後に 1 が無限に続いている配列と同じと考えるため、$\{a\} = \{a,1\} = \{a,1,1,...\}$ です。したがって、$\{a\}$ のプライムは 1 であると考えます。

さて、このように定義すると、配列を高次元に拡張することができます。逆に言えば、高次元の配列が定義されなければ「構造」の定義は意味をなしません。まずは 2 次元の配列について、急増加関数による近似計算の結果を示します。計算の詳細については省略します。

$$\left\{\begin{array}{ccccc} 5 & 5 & 5 & 5 & 5 \\ 2 & & & & \end{array}\right\} \approx F[\omega^{\omega} \times 2](5)$$

$$\left\{\begin{array}{ccccc} 5 & 5 & 5 & 5 & 5 \\ 5 & & & & \end{array}\right\} \approx F[\omega^{\omega} \times \omega](5) = F[\omega^{\omega+1}](5)$$

$$\left\{\begin{array}{ccccc} 5 & 5 & 5 & 5 & 5 \\ 5 & 5 & & & \end{array}\right\} \approx F[\omega^{\omega+1} \times \omega](5) = F[\omega^{\omega+2}](5)$$

$$\left\{\begin{array}{ccccc} 5 & 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 & 5 \end{array}\right\} \approx F[\omega^{\omega+\omega}](5) = F[\omega^{\omega\times 2}](5)$$

$$\left\{\begin{array}{ccccc} 5 & 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 & 5 \end{array}\right\} \approx F[\omega^{\omega\times 3}](5)$$

$$\left\{\begin{array}{ccccc} 5 & 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 & 5 \end{array}\right\} \approx F[\omega^{\omega\times\omega}](5) = F[\omega^{\omega^2}](5)$$

次に、このように高次元の配列を一列で表記するために、(m) という表記を導入します。ここで m は次元の数をあらわし、(1) は 1 次元の区切り

なので次の行に数がうつることを示し、(2) は 2 次元の区切りなので次の平面に数がうつることを示します。たとえば

$$\{5,5,5,5,5(1)2\} = \left\{ \begin{array}{ccccc} 5 & 5 & 5 & 5 & 5 \\ 2 & & & & \end{array} \right\} \approx F[\omega^{\omega} \times 2](5)$$

ということです。$\{5,5(2)2\}$ を BEAF の計算ルールにしたがって計算すると、2 がパイロットとなり、副操縦士はなく、プライムブロックは p^2、すなわち一片の長さが $p = 5$ の平面 (5 行 5 列の平面) となり、その要素が乗客となっていっせいに 5 に変わるため、5 行 5 列の 5 となります。また、BEAF では**配列次元演算子**[14] (“array of” operator) という表記も使います。これは、たとえば $a^n\&b$ を「一辺の長さ a の n 次元超立方体配列中に b が充填されている」と解釈します。これらの表記を使うと、

$$\{5,5(2)2\} = \left\{ \begin{array}{ccccc} 5 & 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 & 5 \end{array} \right\} = 5^2\&5 \approx F[\omega^{\omega^2}](5)$$

となります。ここで、配列次元演算子は次元の構造をあらわしているので、そのまま計算してはいけないことに注意してください。つまり、$5^2\&5$ は $25\&5$ のように計算できません。

2 次元配列が $F[\omega^{\omega^2}]$ の強さとなるように、n 次元配列は $F[\omega^{\omega^n}]$ の強さとなります。100 次元配列の例としてゴンギュラス (gongulus) を示します。

【定義】ゴンギュラス

ゴンギュラスは、BEAF で $\{10,10(100)2\} = 10^{100}\&10$ と等しい数である。$10^{100}\&10$ は 1 辺が 10 の 100 次元超立方体に 10 が充填されている状態で、全部で 10 がグーゴル $= 10^{100}$ 個ある。急増加関数で $F[\omega^{\omega^{100}}](10)$ と近似される。

[14] http://ja.googology.wikia.com/wiki/配列次元演算子

ここから先のテトレーション空間 (tetration space) の話、すなわちテトレーション配列 (tetration array) は、難しくなってきます。バウアーズ本人による BEAF の説明[15] では、テトレーション空間を次のように説明しています。多次元空間を X^n と表記すると、X が直線、X^2 が平面、X^{100} が 100 次元空間です。超次元 (super dimension) は X^{X^n} で、トリメンション (trimension) は $X^{X^{X^n}}$、クアドラメンション (quadramension) は $X^{X^{X^{X^n}}}$ のように続きます。そしてデュラトリ[16] (dulatri) が図解されています。デュラトリは 3 を使った X^{2X} の構造の超次元配列で、「一辺の長さ 3 の 3 次元立方体配列中に 3 が充填されている」ものを 1 つの「立方体要素」と考えて（つまり、1 つの立方体要素に $3^3 = 27$ 個の 3 がある）、一辺の長さ 3 の 3 次元立方体配列中に「立方体要素」が充填されているものがデュラトリです。したがって、そこには $(3^3)^3 = 3^{2\times3} = 729$ 個の 3 が充填されていることになります。このような構造を配列次元演算子で書くときには、構造を分かりやすくするために X をそのまま使います。また、次元の区切りは $(0,2)$ と表記されます。

【定義】デュラトリ

デュラトリは BEAF で $\{3,3(0,2)2\} = X^{2X}\&3$ と書かれる数で、$X = 3$ と評価される。急増加関数では $F[\omega^{\omega^{\omega^2}}](3)$ と近似される。

$X^{X^{X^{X^X}}}$ 次元を $X \uparrow\uparrow 5$ 次元であると表記して、$a \uparrow\uparrow b\& c$ を $a \uparrow\uparrow b$ 次元の c といったように考えることができます。バウアーズ自身は、このようなテトレーション配列以上の BEAF の計算方法について、あまり具体的な計算方法の説明をしていません。ちなみに、テトレーション配列の次はペンテーション配列になりますが、その説明は「その仕組みを理解しようとする人は偏頭痛になるかもしれない」としています。とにかく、テトレーションやペンテーションを考えることができるのと同じように、次元の構造にもテトレーション次元やペンテーション次元を考えることができるのだ、としています。その定義に急増加関数と対応づけて説明を与えたのは、Googology Wiki の人たちです。

[15] Exploding Array Function http://www.polytope.net/hedrondude/array.htm
[16] 巨大数研究 Wiki - デュラトリ http://ja.googology.wikia.com/wiki/デュラトリ

テトレーション配列は $F[\epsilon_0]$ の強さを持ちます。例としてゴッパトス[17] (goppatoth) はこのような数です。

【定義】ゴッパトス

ゴッパトスは、BEAF で 10 ↑↑ 100& 10 と書かれる数である。10 ↑↑ 100 個の 10 が、100 次元のテトレーションハイパー立方体に埋められている。急増加関数では $F[\varepsilon_0](100)$ と近似される。

このように、テトレーション配列の具体的な計算方法は非常に複雑で、$F[\varepsilon_0]$ レベルの関数で比較すれば、計算のアルゴリズムはふぃっしゅ数バージョン 5 やドル関数の方がずっと簡単です。BEAF の優れているところは「配列次元演算子」のような抽象化力にあります。つまり、テトレーション次元をテトレーション記号と配列次元演算子を使って表記してしまうことで、あまり具体的な計算方法に気を煩わせることなく $F[\varepsilon_0]$ レベルの数を簡潔に表現できて、さらに具体的に想像することはできないようなペンテーション次元等の「高次の概念」へと拡張していきます。つまり、ここから先の拡張が BEAF の本領発揮となりますが、その話については次章で続けます。

6.9 巨大数生成プログラム

本章の最後に、グッドスタイン数列を使った巨大数を生み出すプログラムを紹介します。Googology Wiki で、Deedlit11 が C 言語あるいはその他の言語によってなるべく短い文字数で $F[\epsilon_0]$ の増加速度を持つ関数を記述する、という競技を開催しました[18]。

巨大数ベイクオフ大会 (p.226) に準じて、プログラム言語には制限を設けない次のようなルールです。

1. $F[\omega^{\omega^{\omega^{\omega}}}](4)$ よりも大きな数を出力するプログラムを作成する。この数には特別な意味はない。$F[\epsilon_0]$ レベルの関数がなければ作れない数で、それほど大きくない数を選んだ。

[17] 巨大数研究 Wiki - ゴッパトス http://ja.googology.wikia.com/wiki/ゴッパトス
[18] Googology Wiki - User blog:Deedlit11/Bignum Bakeoff appetizer

2. 記憶容量の制限はなく、任意の整数を扱うことができるものとする。したがって、MAXINT のような参照はできない。

3. あるプログラム言語における優勝は、ホワイトスペースを除いた文字数が最小のプログラムである。

この競技で、Deedlit11 自身によるグッドスタイン数列を使った 101 文字の C 言語プログラム、そしてそれを元にした Sai2000 による 83 文字の Python 3 プログラムが作成されました。Python プログラムを掲載します。

```
i = 9 ** 9 ** 9
b = 2

B=lambda y,x=0:y and y%b*(b*b)**B(x)+B(y/b,x+1)
while i:
    i = B(i)-1
    b *= b
print(b)
```

これは、グッドスタイン関数 $G(9^{9^9})$ を計算するプログラムです。$G(2^{65536})$ で $F[\omega^{\omega^{\omega^{\omega}}}]$ のレベルに到達するので (p.159)、$G(9^{9^9})$ は $F[\omega^{\omega^{\omega^{\omega}}}](4)$ を超えます。

第 7 章

再帰関数

本章では $F[\epsilon_0]$ よりも大きな関数と巨大数を扱います。そのために、まず ϵ_0 よりも大きな順序数を表記するための、ヴェブレン関数と順序数崩壊関数について説明をします。また、2 階算術およびその部分体系と、その証明論的順序数について説明をします。それから具体的な巨大数を生み出すシステムとその強さについて、解析ができている範囲で紹介します。

7.1 ヴェブレン関数

ϵ_0 は、$\alpha = \omega^\alpha$ が成り立つ最小の順序数です。ϵ_0 よりも大きく $\alpha = \omega^\alpha$ が成り立つ最小の順序数は ϵ_1 と書かれ、$a[0] = \epsilon_0 + 1, a[n+1] = \omega^{a[n]}$ を基本列に持つ極限順序数です。その基本列は次のように計算されます。

$$
\begin{aligned}
a[0] &= \epsilon_0 + 1 \\
a[1] &= \omega^{\epsilon_0+1} = \omega^{\epsilon_0}\omega = \epsilon_0\omega \\
a[2] &= \omega^{\epsilon_0\omega} = (\omega^{\epsilon_0})^\omega = {\epsilon_0}^\omega \\
a[3] &= \omega^{{\epsilon_0}^\omega} = \omega^{{\epsilon_0}^{1+\omega}} = \omega^{\epsilon_0{\epsilon_0}^\omega} = (\omega^{\epsilon_0})^{{\epsilon_0}^\omega} = {\epsilon_0}^{{\epsilon_0}^\omega} \\
a[4] &= \omega^{{\epsilon_0}^{{\epsilon_0}^\omega}} = \omega^{{\epsilon_0}^{1+{\epsilon_0}^\omega}} = \omega^{\epsilon_0{\epsilon_0}^{{\epsilon_0}^\omega}} = {\epsilon_0}^{{\epsilon_0}^{{\epsilon_0}^\omega}}
\end{aligned}
$$

この基本列と $b[i] = {\epsilon_0}\text{^^}i$、つまり $b[0] = 1, b[1] = \epsilon_0, b[2] = {\epsilon_0}\text{^^}2 = {\epsilon_0}^{\epsilon_0}$, $b[3] = {\epsilon_0}\text{^^}3, \cdots$ の基本列を比較すると、$i > 1$ で $b[i-1] < a[i] < b[i]$ となるため収束先は一致し、$a[\omega] = b[\omega]$ となります。したがって、

$$\epsilon_1 = a[\omega] = b[\omega] = \epsilon_0{}^{\wedge\wedge}\omega$$

となります。このように、基本列の取り方は一通りではありません。そして $\omega^{\epsilon_1} = \epsilon_1$ が成り立ちます。$\alpha = \omega^\alpha$ が成り立つ順序数は、順番に ϵ_0, ϵ_1, ϵ_2, ... と定義されます。つまり、

$$\begin{aligned}\epsilon_{i+1} &= \lim(\epsilon_i + 1, \omega^{\epsilon_i+1}, \omega^{\omega^{\epsilon_i+1}}, \cdots) \\ &= \epsilon_i{}^{\wedge\wedge}\omega\end{aligned}$$

となります。次に、ϵ_0, ϵ_1, ϵ_2, $\cdots$ の基本列を取ると、極限順序数 ϵ_ω が定義され、これは $\alpha = \omega^\alpha$ が成り立つ ω 番目の順序数です。

さらに基本列の定義を繰り返し、ϵ_ω,ϵ_{ω^ω},$\epsilon_{\omega^{\omega^\omega}}$,... という基本列を考えると、$\alpha = \omega^\alpha$ が成り立つ ϵ_0 番目の順序数 ϵ_{ϵ_0} が定義されます。

そして $\epsilon_0, \epsilon_{\epsilon_0}, \epsilon_{\epsilon_{\epsilon_0}}, \epsilon_{\epsilon_{\epsilon_{\epsilon_0}}}$, ... といった基本列を取ると、極限順序数 ζ_0（ζ：ゼータ; ツェータ）が定義されます。ζ_0 は、$\alpha = \epsilon_\alpha$ が成り立つ最小の順序数です。ここで、この順序数は Googology Wiki では ζ_0 とされていますが、η_0 と書かれることもあります。ϵ_0 よりも大きい順序数の記号はあまり統一されていないようなので、混乱しないためには注意が必要です。

アメリカの数学者オズワルド・ヴェブレン (Oswald Veblen, 1880–1960) は 1908 年の論文で、以下のヴェブレン関数[1] (Veblen function) $\phi_\alpha(\beta)$ を定義しました。

【定義】ヴェブレン関数

$\phi_0(\beta) = \omega^\beta$
$\phi_{\alpha+1}(\beta) =$「$\phi_\alpha(\gamma) = \gamma$ となる γ」 のうち β 番目のもの
$\phi_\alpha(\beta) =$ 「すべての $\alpha' < \alpha$ に対し $\phi_{\alpha'}(\gamma) = \gamma$ となる γ」 のうち β 番目のもの (α は極限順序数)
$\phi_\alpha(\beta) < \phi_\gamma(\delta)$ となるのは、「$\alpha = \gamma$ かつ $\beta < \delta$」であるか、または「$\alpha < \gamma$ かつ $\beta < \phi_\gamma(\delta)$」であるか、または「$\alpha > \gamma$ かつ $\phi_\alpha(\beta) < \delta$」のときだけである。

[1] Veblen, O. (1908), Continuous increasing functions of finite and transfinite ordinals. *Transactions of the American Mathematical Society* 9 (3): 280–292 doi:10.1090/S0002-9947-1908-1500814-9

ヴェブレン関数の定義 $\phi_{\alpha+1}(\beta)$ =「$\phi_\alpha(\gamma)=\gamma$ となる β 番目の数」は、「ある順序数 γ より小さい数に $+, \phi_0, \phi_1, \cdots, \phi_\alpha$ を何回適用してもやはり γ より小さい」という条件を満たす γ のうち β 番目のものとしても同じことです。「 」の条件は、「γ より小さい数は $+, \phi_0, \phi_1, \cdots, \phi_\alpha$ という演算について閉じている」と言い換えることもできます。

(1) $\phi_0(\alpha)$ の計算

まずは、$\phi_0(\alpha)=\omega^\alpha$ について考えます。

$\alpha=0$ のときは $\omega^0=1$ より小さい数は 0 しか存在せず、0 をいくら足し合わせても 1 よりは小さいことから、1 が「その数より小さい数は+について閉じている」ような 0 番目の (最初の) 数ということになります。

$\alpha=1$ のときは、$\omega^1=\omega$ より小さい数 (つまり自然数) をいくら足し合わせても ω よりは小さいということになります。逆に考えると、自然数をいくら足し合わせても到達できないような最小の数を $\phi_0(1)=\omega$ として定義していることになります。

$\alpha=2$ のときは、$\phi_0(2)$ は 2 番目の数であるため 1 番目の数である ω よりは大きな数となります。なので、自然数や ω をいくら足し合わせても到達できないような最小の数を考えると、それは $\omega\times n$ の収束先である ω^2 ということになります。

同様に、$\phi_0(3)=\omega^3$, $\phi_0(4)=\omega^4$, $\cdots$, $\phi_0(n)=\omega^n$ となり、さらに

$$\begin{aligned}\phi_0(\phi_0(1)) &= \phi_0(\omega)=\omega^\omega\\ \phi_0(\phi_0(\phi_0(1))) &= \phi_0(\omega^\omega)=\omega^{\omega^\omega}\end{aligned}$$

のように、ヴェブレン関数によって順序数が定義されます。

(2) $\phi_1(\alpha)$ の計算

次に $\phi_1(0)$ を考えます。これは、「$\phi_0(\alpha)=\alpha$ となる α のうち最初のもの」つまり「$\omega^\alpha=\alpha$ となる最初の α」です。これが ϵ_0 の定義でした。また、ϵ_0 より小さい数に+や ϕ_0 を何回適用しても ϵ_0 より小さいということになります。0 に+や ϕ_0 を何回適用しても到達できないような数のうち、最小のものを $\phi_1(0)=\epsilon_0$ と定義していると言うこともできます。

$\phi_1(1) = \epsilon_1$ は、ϵ_0 以下の数 (ϵ_0 も含む) に+や ϕ_0 を何回適用しても到達できない最小の数ということになります。$\epsilon_1 = \epsilon_0{}^{\wedge\wedge}\omega$ ですから、ϵ_0 に冪乗を繰り返しても到達できない数です。

以下、$\phi_1(2) = \epsilon_2$, $\phi_1(3) = \epsilon_3$, $\cdots$, $\phi_1(\alpha) = \epsilon_\alpha$ と定義され、さらに

$$\begin{aligned} \phi_1(\phi_1(0)) &= \phi_1(\epsilon_0) = \epsilon_{\epsilon_0} \\ \phi_1(\phi_1(\phi_1(0))) &= \phi_1(\epsilon_{\epsilon_0}) = \epsilon_{\epsilon_{\epsilon_0}} \end{aligned}$$

と計算することができます。

(3) $\phi_2(\alpha)$ の計算

$\phi_2(0)$ は、「$\phi_1(\gamma) = \gamma$ となる γ のうち最初のもの」つまり、「$\epsilon_\gamma = \gamma$ となる最初の γ」となります。0 に+と ϕ_0 と ϕ_1 を何回適用しても到達できない最小の数です。そのような順序数は、ζ_0 になります。

そして、$\epsilon_\gamma = \gamma$ となる α 番目の順序数が、$\phi_2(\alpha) = \zeta_\alpha$ で定義されます。

このように、+だけでは到達できない数を ϕ_0 で定義し、+と ϕ_0 だけでは到達できない数を ϕ_1 で定義し、+と ϕ_0 と ϕ_1 だけでは到達できない数を ϕ_2 で定義し、…と続けていきます。この定義を続けると、ヴェブレン関数を何回使っても到達できない最小の順序数、すなわち $\phi_\alpha(0) = \alpha$ を満たす最小の順序数が定義されます。これを Γ_0 と書いて、ソロモン・フェファーマン (Solomon Feferman, 1928–2016) とクルト・シュッテ (Kurt Schütte, 1909–1998) の名前を取ってフェファーマン・シュッテの順序数 (Feferman–Schütte ordinal) と呼ばれます。ここで、シュッテは Γ_0 が可述的 (predicative) な順序数の限界だとしました。

ヴェブレンは 2 変数関数の定義を多変数 $\phi(\alpha_n, \alpha_{n-1}, \cdots, \alpha_0)$ に拡張しています。多変数ヴェブレン関数に対して、

$$\begin{aligned} \phi(1,0,0) &= \Gamma_0 \\ \phi(1,0,\gamma) &= \Gamma_\gamma \\ \phi(1,0,0,0) &= \text{アッカーマン順序数 (Ackermann ordinal)} \end{aligned}$$

$$\phi(1,\underbrace{0,\cdots,0}_{\omega\text{個の }0}) \quad = \quad \text{小ヴェブレン順序数 (small Veblen ordinal)}$$

となります。

ヴェブレンはさらに、変数の数を超限的にたくさん増やしたとき (transfinitely many variables) にも、ヴェブレン関数を定義できるとしています。この定義の限界が大ヴェブレン順序数 (large Veblen ordinal) です。

7.2 順序数崩壊関数

2 変数ヴェブレン関数の限界であるフェファーマン・シュッテの順序数よりも大きな順序数を表記するときには、順序数崩壊関数 (ordinal collapsing function) がよく使われます。順序数崩壊関数には様々なバージョンがあり、通常は ψ（プサイ）あるいは θ（シータ）が記号として使われます。同じ記号であっても様々な定義があるので、どの定義によるものなのかを確認する必要があります。

7.2.1 非可算順序数と共終数

順序数崩壊関数は、非可算順序数（基数）を使って大きな可算順序数を定義する関数です。急増加関数 $F[\alpha]$ で、α が非可算な順序数になることはありませんが、大きな可算順序数 α を作るために非可算順序数を使います。有限の自然数を得るために無限の順序数を導入し、さらに大きな可算の順序数を得るために非可算の順序数を導入する、というところは面白いだろうと思います。有限の話をしているはずなのに、なぜかどんどん大きな無限の話が出てきます。図 7.1 に、順序数崩壊関数によって非可算順序数から大きな自然数である巨大数を定義するまでの流れをまとめました。

まずは順序数崩壊関数で使う非可算順序数について簡単にまとめておきます。

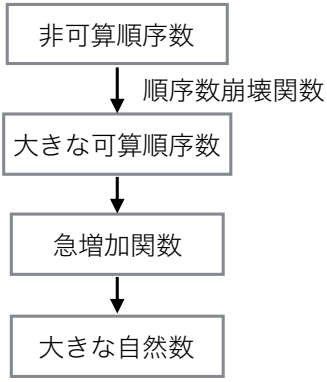

図 7.1: 非可算順序数による巨大数の定義

集合 A の基数とは、A との間に全単射が存在する順序数の中で最小のもので、可算無限集合の基数は ω となり、アレフ 0 ($\aleph_0$) とも書く、ということはすでに説明しました。詳細は省きますが、連続体仮説によれば $\aleph_0$ と連続体濃度 $2^{\aleph_0}$ の間には基数が存在しないので、$\aleph_0$ の次の基数である $\aleph_1$ は $2^{\aleph_0}$ となります。$\aleph_1 = \omega_1$ は最小の非可算順序数で、すべての可算順序数の集合です。同様に、$\omega_2 = \aleph_2 = 2^{\aleph_1}$ と定義をすることができます。連続体仮説を使わずに書けば、$\omega_{\alpha+1}$ は濃度が ω_α よりも大きい最小の基数です。さらに極限順序数 α に対して ω_α をすべての $\beta < \alpha$ に対する ω_β の上限であるとすれば、一般に ω_α が定義できます。$\omega_\omega, \omega_{\omega_\omega}, \omega_{\omega_{\omega_\omega}}, \ldots$ と定義を続けることができて、そのような基本列を持つ順序数は $\alpha = \omega_\alpha$ となる最小の順序数でオメガ収束点[2] (omega fixed point) となります。Googology Wiki ではオメガ収束点を $\psi_I(0)$ と表記しています。これは、オメガ収束点よりもさらに大きい最小の到達不能基数 (inaccessible cardinal) I を崩壊させてオメガ収束点を作っています。このように、非可算基数から非可算基数への崩壊をさせることもあります。

順序数崩壊関数では、ω を大文字で Ω と表記することがよくあります。すなわち、$\Omega_\alpha = \omega_\alpha$ です。ただし、Ω は ω ではなくて ω_1 です。

[2] http://ja.googology.wikia.com/wiki/オメガ収束点

ここで、Ω が最小の再帰的ではない順序数であるチャーチ・クリーネ順序数[3] (Church-Kleene ordinal) ω_1^{CK} を意味することもあります。再帰的順序数 (recursive ordinal) は、再帰的な整列集合の順序型です。ω_1^{CK} は再帰的順序数の上限、すなわち最小の再帰的ではない順序数です。再帰的な関係は可算なので、再帰的順序数も可算です。したがって ω_1^{CK} は可算です。ω_1^{CK} は多くの場合に ω_1 と似たような挙動を示すため、$\Omega = \omega_1^{CK}, \Omega_\alpha = \omega_\alpha^{CK}$ と定義をしてもうまくいきます。ただし、そのように説明をしてしまうと「非可算」という概念を使って説明をできなくなってしまいます。Ω は非可算順序数であるとされることが多いので、本書でもそのようにします。

次に、共終数 (cofinality) という概念についても説明をしておきます。半順序集合 A の共終数 $\mathrm{cf}(A)$ とは、A と共終な部分集合の基数の中で最小のものです。ここで A の部分集合 B が A と共終であるとは、「A の任意の要素 a に対して、$a \leq b$ となるような B の要素 b が存在する」という意味です。順序数 α の共終数は、α と共終な部分集合の順序型である順序数 δ の中で最小のものです。極限順序数 α に対して、δ までの順序数でインデックスされた順序数の単調増加列で、α に収束するものが存在します。たとえば $\alpha = \omega^3$ の共終数は ω です。なぜならば、自然数 n に対して $\alpha[n] = \omega^2 \cdot n$ という数列を与えれば、$\alpha = \omega^3$ に収束するためです。

順序数の基本列を定めるとは、ω までの順序数でインデックスすることにほかなりませんから、共終数が ω である順序数は基本列を定めることができますが、共終数が非可算の順序数に対しては基本列を定めることができません。0 の共終数は 0 で、後続順序数の共終数は 1 です。可算な極限順序数の共終数は ω です。非可算な極限順序数の共終数は、ω となる時と非可算になる時があります。たとえば $\omega_1 + \omega$ は非可算ですが、$\alpha[n] = \omega_1 + n$ という数列でインデックスできるので、共終数は ω になります。

7.2.2 フェファーマンの θ 関数

フェファーマンの θ 関数 (Feferman's θ-function) は、順序数から順序数への関数 $\theta_\alpha(\beta)$ です。2 変数関数 $\theta_\alpha(\beta)$ とみなされて、$\theta(\alpha, \beta)$ や $\theta\alpha\beta$ とも

[3] http://ja.googology.wikia.com/wiki/チャーチ・クリーネ順序数

書かれます。次のような定義です。

【定義】フェファーマンの θ 関数

$$
\begin{aligned}
C_0(\alpha,\beta) &= \beta \cup \{0, \omega_1, \omega_2, \ldots, \omega_\omega\} \\
C_{n+1}(\alpha,\beta) &= \{\gamma+\delta, \theta_\xi(\eta) \mid \gamma,\delta,\xi,\eta \in C_n(\alpha,\beta); \xi < \alpha\} \\
C(\alpha,\beta) &= \bigcup_{n<\omega} C_n(\alpha,\beta) \\
\theta_\alpha(\beta) &= \min\{\gamma \mid \gamma \notin C(\alpha,\gamma) \wedge \forall \delta < \beta : \theta_\alpha(\delta) < \gamma\}
\end{aligned}
$$

この定義はこういう意味です。$C(\alpha,\beta)$ は、順序数 $0, \omega_1, \omega_2, \ldots, \omega_\omega$ と β よりも小さい順序数から、有限回の「加算」「すでに定義されている θ 関数、すなわち $\xi < \alpha$ に対する θ_ξ 関数」を適用した順序数すべての集合です。そして $\theta_\alpha(\beta)$ は $C(\alpha,\beta)$ に含まれない β 番目の順序数です。

ここで $\omega_1, \omega_2, \ldots, \omega_\omega$ を無視してこの定義を読むと、ヴェブレン関数と同じで、その限界はフェファーマン・シュッテの順序数 Γ_0 です。そして、$\alpha \leq \Gamma_0$ であれば、フェファーマンの θ 関数は 2 変数ヴェブレン関数と一致して $\theta_\alpha(\beta) = \phi_\alpha(\beta)$ となります。なぜならば $\phi_\alpha(\beta)$ や Γ_0 のような可算順序数を作るためには、$\omega_1, \omega_2, \ldots, \omega_\omega$ といった非可算順序数は大きすぎて材料としては役に立たないからです。Γ_0 よりも大きい ω_1, ω_1+1 などは $C(\alpha,\beta)$ に属しますが、$\theta_\alpha(\beta)$ は $C(\alpha,\beta)$ に属さない β 番目の順序数という定義なので、Γ_0 よりも大きい順序数が $C(\alpha,\beta)$ に属していても Γ_0 が含まれていなければ $\theta_\alpha(0) = \Gamma_0$ となるわけです。

ここで、ヴェブレン関数と違うのは 2 番目の式で θ_ξ 関数の適用をするときに、$\xi \in C_n(\alpha,\beta)$ の条件が入っていることです。そのために、$\alpha > \Gamma_0$ であっても θ_{Γ_0} 関数の適用ができません。すなわち $\theta_{\Gamma_0}(0) = \phi_{\Gamma_0}(0) = \Gamma_0$ となりますが、$\theta_{\Gamma_0+1}(0)$ の値はこれよりも大きくなりません。よって $\Gamma_0 \leq \alpha \leq \Omega$ において $\theta_\alpha(0) = \Gamma_0$ です。つまり $\theta_\Omega(0) = \Gamma_0$ です。さらに順序数講座[4] で詳しく解説しているように、$\theta_\Omega(0)$ は $\alpha = \theta_\alpha(0)$ を満たす最小の順序数となり、$\theta_\Omega(\beta)$ は $\alpha = \theta_\alpha(0)$ を満たす β 番目の順序数となります。

[4] http://ja.googology.wikia.com/wiki/ユーザーブログ:Kyodaisuu/順序数講座

それでは $\theta_{\Omega+1}(0)$ はどうなるでしょうか。$C(\alpha,\beta)$ の定義に $\Omega=\omega_1$ を入れておいたため、$C(\Omega+1)$ の定義で $\theta_\Omega(\beta)=\Gamma_\beta$ を使うことができます。つまり $C(\Omega+1,0)$ は、$0,\omega_1,\omega_2,\ldots,\omega_\omega$ から、加算、$\theta_\alpha(\alpha\leq\Omega)$ 関数で構成できるすべての順序数が含まれ、その関数の中には Γ_β が含まれます。そして $\theta_{\Omega+1}(0)$ は、そのように構成できない最小の順序数なので $\alpha=\Gamma_\alpha$ を満たす最小の順序数 $\phi(1,1,\beta)$（多変数ヴェブレン関数）となります。

ここで $\theta_\Omega(0)=\Gamma_0$ の定義のように、ある順序数の定義に自分自身よりも大きな順序数を使うときに、その定義は非可述的 (impredicative) であると言います。通常は非可述的な定義とは自分自身を定義に含む定義ですが、実際には $\theta_\Omega(0)=\Gamma_0$ を定義することは $\theta_{\Gamma_0}(0)=\Gamma_0$ を定義することと同じなので、自分自身を定義に含めることと同じになります。順序数崩壊関数の「崩壊」の意味は、このように非可算順序数から、可算順序数へと崩しているという意味なので、非可述的な定義が順序数崩壊関数の本質となります。

このような計算を続けてみます。

$$
\begin{aligned}
\theta_\Omega(\beta) &= \Gamma_\beta=\phi(1,0,\beta)\,(\alpha=\phi_\alpha(0)\text{ を満たす }\beta\text{ 番目の順序数})\\
\theta_{\Omega+1}(\beta) &= \phi(1,1,\beta)\,(\alpha=\phi(1,0,\alpha)\text{ を満たす }\beta\text{ 番目の順序数})\\
\theta_{\Omega+2}(\beta) &= \phi(1,2,\beta)\\
\theta_{\Omega+\omega}(\beta) &= \phi(1,\omega,\beta)\\
\theta_{\Omega 2}(\beta) &= \theta_{\Omega+\Omega}(\beta)=\phi(2,0,\beta)\,(\alpha=\phi(1,\alpha,\beta)\\
\theta_{\Omega\omega}(\beta) &= \phi(\omega,0,\beta)\\
\theta_{\Omega^2}(0) &= \theta_{\Omega\Omega}(0)=\phi(1,0,0,0)\,(\text{アッカーマン順序数})\\
\theta_{\Omega^2+1}(\beta) &= \phi(1,0,1,\beta)\\
\theta_{\Omega^2+\Omega}(\beta) &= \phi(1,1,0,\beta)\\
\theta_{\Omega^2 2}(\beta) &= \theta_{\Omega^2+\Omega\Omega}(0)=\phi(2,0,0,\beta)\\
\theta_{\Omega^2 3}(\beta) &= \phi(3,0,0,\beta)\\
\theta_{\Omega^3}(\beta) &= \theta_{\Omega^2\Omega}(0)=\phi(1,0,0,0,\beta)\\
\theta_{\Omega^3+1}(\beta) &= \phi(1,0,0,1,\beta)\\
\theta_{\Omega^3+\Omega}(\beta) &= \phi(1,0,1,0,\beta)\\
\theta_{\Omega^3+\Omega^2}(\beta) &= \phi(1,1,0,0,\beta)
\end{aligned}
$$

$$
\begin{aligned}
\theta_{\Omega^3 2}(\beta) &= \phi(2,0,0,0,\beta) \\
\theta_{\Omega^3 \omega}(\beta) &= \phi(\omega,0,0,0,\beta) \\
\theta_{\Omega^4}(\beta) &= \phi(1,0,0,0,0,\beta) \\
\theta_{\Omega^\omega}(0) &= \text{小ヴェブレン順序数 (多変数ヴェブレン関数の限界)} \\
\theta_{\Omega^\Omega}(0) &= \text{大ヴェブレン順序数 (超限変数ヴェブレン関数の限界)}
\end{aligned}
$$

ここで、たとえば Ω^2 という非可算順序数は $\theta_0(\Omega 2) = \omega^{\omega^2} = (\omega^\Omega)^2 = \Omega^2$ のように、$\omega_1 = \Omega$ と θ_0 関数から作ることができます。つまり上の計算は $\theta_{\theta_0(\Omega 2)}(0) = \theta_{\Omega^2}(0) = \phi(1,0,0,0)$ のような計算過程を省略したものです。さらに $\theta_0^2(\Omega 2) = \Omega^\Omega$, $\theta_0^3(\Omega 2) = \Omega^{\Omega^\Omega}$, ... のような基本列を持つ非可算順序数 $\epsilon_{\Omega+1}$ を考えると、$\theta_1(\Omega+1) = \epsilon_{\Omega+1}$ となります。そして $\theta_{\epsilon_{\Omega+1}}(0)$ をバッハマン・ハワード順序数 (Bachmann-Howard ordinal) と言います。これは $\theta_\Omega(0), \theta_{\Omega^\Omega}(0), \theta_{\Omega^{\Omega^\Omega}}(0), ...$ が基本列です。そして

$$
\begin{aligned}
\theta_{\theta_1(\Omega+1)}(0) &= \theta_{\epsilon_{\Omega+1}}(0)\,(\text{バッハマン・ハワード順序数}) \\
\theta_{\theta_2(\Omega+1)}(0) &= \theta_{\zeta_{\Omega+1}}(0)
\end{aligned}
$$

のようにさらに大きな順序数が定義できますが、やがて Ω だけでは作ることができないような限界の順序数にぶち当たり、$\omega_2 = \Omega_2$ を使った $\theta_{\Omega_2}(0)$ は、そのような限界の順序数となります。さらに、$\omega_3, \omega_4, ...$ と順序数を導入します。

このように、新しい非可算順序数を導入することで、それまでの非可算順序数だけでは作ることができないような大きな可算順序数を作ることができるようになります。そして Ω_ω を使った $\theta_{\theta_1(\Omega_\omega+1)}(0) = \theta_{\epsilon_{\Omega_\omega+1}}(0)$ が、竹内外史の順序数図形 (ordinal diagram) [5] [6] や、次のブーフホルツの ψ 関数の限界と等しくなり、 竹内・フェファーマン・ブーフホルツ順序数[7] (Takeuti-Feferman-Buchholz ordinal) と呼ばれます。フェファーマンの θ 関数の限界は、これよりももう少し大きいところにあります。

---

[5] Takeuti, G. (1960) Ordinal diagrams II. *Journal of the Mathematical Society of Japan* 12: 385–391. doi:10.2969/jmsj/01240385

[6] 八杉満利子 (1974) 「Ordinal Diagram について」数学 26(2): 121–136. doi:10.11429/sugaku1947.26.121

[7] http://ja.googology.wikia.com/wiki/竹内・フェファーマン・ブーフホルツ順序数

7.2.3 ブーフホルツの ψ 関数

ドイツの数理論理学者ヴィルフリート・ブーフホルツ (Wilfried Buchholz) が定義した順序数崩壊関数を、ブーフホルツの ψ 関数[8] (Buchholz's ψ function) と言います。ブーフホルツの ψ 関数は、$\nu \leq \omega$ と順序数 α に対して $\psi_\nu(\alpha)$ によって順序数を定義する関数で、非可算順序数 $\Omega_\nu = \omega_\nu$ を定義に用いています。定義は省略しますので、脚注の原著論文を参照してください。

ブーフホルツは $1 \leq \nu \leq \omega$ のときに $\psi_0(\epsilon_{\Omega_\nu+1}) = \theta_{\epsilon_{\Omega_\nu+1}}(0)$ が成り立つことを証明しました。つまり、ブーフホルツの ψ 関数はフェファーマンの θ 関数と同等の強さを持ち、竹内・フェファーマン・ブーフホルツ順序数までを表記できます。

7.2.4 マドールの ψ 関数

ブーフホルツの ψ 関数は数学的には美しいが素人には分かりにくいとして、フランスの数学者ダヴィド・マドール (David Madore; Wikipedia ユーザー名 Gro-Tsen) が順序数崩壊関数を説明するために定義して 2013 年 2 月 13 日に Wikipedia の記事 “Ordinal collapsing function” に書いた次のような関数を、マドールの ψ 関数 (Madore's ψ function) と言います。

【定義】マドールの ψ 関数

すべての $\beta < \alpha$ に対して $\psi(\beta)$ が定義されているときに、$\psi(\alpha)$ を以下によって定義する。$\Omega = \omega_1$ （最小の非可算順序数）である。

$$
\begin{aligned}
C_0(\alpha) &= \{0, 1, \omega, \Omega\} \\
C_{n+1}(\alpha) &= \{\beta_1 + \beta_2, \beta_1\beta_2, \beta_1^{\beta_2}, \psi(\beta) \mid \beta_1, \beta_2, \beta \in C_n(\alpha); \beta < \alpha\} \\
C(\alpha) &= \bigcup_{n<\omega} C_n(\alpha) \\
\psi(\alpha) &= \min\{\beta \in \Omega \mid \beta \notin C(\alpha)\}
\end{aligned}
$$

[8] Buchholz, W. (1986) A new system of proof-theoretic ordinal functions. *Annals of Pure and Applied Logic* 32, 195–207. doi:10.1016/0168-0072(86)90052-7

まず、$C(\alpha)$ を $0, 1, \omega, \Omega$ から加算、乗算、冪乗と $\beta < \alpha$ に対する $\psi(\beta)$ のみを有限回使って定義できるすべての順序数の集合とします。$\psi(\alpha)$ は $C(\alpha)$ に属さない最小の順序数です。つまりもっと大雑把に言えば、$\psi(\alpha)$ は $0, 1, \omega, \Omega$ から加算、乗算、冪乗と、α よりも小さいすでに定義された順序数に対する ψ 関数で表記できない最小の順序数です。

Wikipedia に分かりやすく解説が書かれていることから、多くの人がマドールの ψ 関数を使うようになり、メジャーになってきています。

まず $C(0)$ について考えます。これは $0, 1, \omega, \Omega$ から、加算、乗算、冪乗で定義できない最小の順序数ということです。ϵ_0 よりも小さい順序数は、すべて $0, 1, \omega$ から加算、乗算、冪乗で定義できます。ϵ_0 は定義できません。Ω は大きすぎて ϵ_0 作りには役立ちません。したがって $\psi(0) = \epsilon_0$ となります。

次に $C(1)$ について考えると、これは $0, 1, \omega, \Omega, \epsilon_0$ から加算、乗算、冪乗で定義できない最小の順序数となります。$C(1)$ は ϵ_1 よりも小さいすべての順序数を含み、ϵ_1 を含まないため、$\psi(1) = \epsilon_1$ となります。同様に $\psi(\alpha) = \epsilon_\alpha$ であることを帰納的に証明できますが、その証明が有効なのは $\alpha < \epsilon_\alpha$ が成立している間だけです。$\alpha = \epsilon_\alpha$ となる最小の順序数は $\zeta_0 = \phi_2(0)$ なので、$\alpha < \zeta_0$ に対して $\psi(\alpha) = \epsilon_\alpha$ となります。そして、$\psi(\zeta_0) = \zeta_0$ となりますが、$\psi(\zeta_0 + 1)$ は ζ_0 よりも大きくなりません。ζ_0 は $\phi_1(\alpha) = \epsilon_\alpha$ 関数の有限回の繰り返しでは定義できないためです。$\alpha \leq \Omega$ において、ζ_0 は $C(\alpha)$ の要素にならないため、$\zeta_0 \leq \alpha \leq \Omega$ に対して $\psi(\alpha) = \zeta_0$ となります。

$\psi(\Omega + 1)$ の定義では、$C(\alpha)$ の定義に Ω が入っているため、$\psi(\Omega) = \zeta_0$ を使うことができます。すなわち $C(\Omega + 1)$ は、$0, 1, \omega, \Omega, \zeta_0$ から加算、乗算、冪乗、$\phi_1(\alpha) = \epsilon_\alpha$ 関数で構成できるすべての順序数が含まれます。$\psi(\Omega + 1)$ は、そのように構成できない最小の順序数なので $\epsilon_{\zeta_0 + 1}$ となります。このように、マドールの順序数崩壊関数では ζ_0 が最小の非可述的な順序数になります。フェファーマンの順序数崩壊関数から、シュッテは Γ_0 から非可述的な順序数になると説明をしましたが、どの順序数から非可述的となるかは順序数崩壊関数の定義にもよるということです。

同様に、次のように計算が進みます。

$$
\begin{aligned}
\psi(\Omega+2) &= \epsilon_{\zeta_0+2}\\
\psi(\Omega+\omega) &= \epsilon_{\zeta_0+\omega}\\
\psi(\Omega+\zeta_0) &= \epsilon_{\zeta_0+\zeta_0}=\epsilon_{\zeta_0 2}\\
\psi(\Omega 2) &= \psi(\Omega+\Omega)=\zeta_1=\phi_2(1)\\
\psi(\Omega 3) &= \zeta_2=\phi_2(2)\\
\psi(\Omega\omega) &= \zeta_\omega=\phi_2(\omega)\\
\psi(\Omega^2) &= \psi(\Omega\Omega)=\phi_3(0)\\
\psi(\Omega^3) &= \phi_4(0)\\
\psi(\Omega^\omega) &= \phi_\omega(0)\\
\psi(\Omega^\Omega) &= \Gamma_0=\theta_\Omega(0)\\
\psi(\Omega^\Omega+\Omega) &= \phi_{\Gamma_0+1}(0)\\
\psi(\Omega^\Omega+\Omega^2) &= \phi_{\Gamma_0+2}(0)\\
\psi(\Omega^\Omega+\Omega^\omega) &= \phi_{\Gamma_0+\omega}(0)\\
\psi(\Omega^\Omega 2) &= \psi(\Omega^\Omega+\Omega^\Omega)=\Gamma_1=\theta_\Omega(1)\\
\psi(\Omega^\Omega 3) &= \Gamma_2=\phi(1,0,2)=\theta_\Omega(2)\\
\psi(\Omega^\Omega\omega) &= \Gamma_\omega=\phi(1,0,\omega)=\theta_\Omega(\omega)\\
\psi(\Omega^{\Omega+1}) &= \psi(\Omega^\Omega\Omega)=\phi(1,1,0)=\theta_{\Omega+1}(0)\\
\psi(\Omega^{\Omega+2}) &= \phi(1,2,0)=\theta_{\Omega+2}(0)\\
\psi(\Omega^{\Omega+\omega}) &= \phi(1,\omega,0)=\theta_{\Omega+\omega}(0)\\
\psi(\Omega^{\Omega 2}) &= \psi(\Omega^{\Omega+\Omega})=\phi(2,0,0)=\theta_{\Omega 2}(0)\\
\psi(\Omega^{\Omega 3}) &= \phi(3,0,0)=\theta_{\Omega 3}(0)\\
\psi(\Omega^{\Omega 4}) &= \phi(4,0,0)=\theta_{\Omega 4}(0)\\
\psi(\Omega^{\Omega\omega}) &= \phi(\omega,0,0)=\theta_{\Omega\omega}(0)\\
\psi(\Omega^{\Omega^2}) &= \phi(1,0,0,0)=\theta_{\Omega^2}(0)\\
\psi(\Omega^{\Omega^3}) &= \phi(1,0,0,0,0)=\theta_{\Omega^3}(0)\\
\psi(\Omega^{\Omega^\omega}) &= \theta_{\Omega^\omega}(0)\ \text{(小ヴェブレン順序数)}
\end{aligned}
$$

$$\begin{aligned}\psi(\Omega^{\Omega^{\Omega}}) &= \theta_{\Omega^{\Omega}}(0)\,(\text{大ヴェブレン順序数})\\ \psi(\Omega^{\Omega^{\Omega^{\Omega}}}) &= \theta_{\Omega^{\Omega^{\Omega}}}(0)\\ \psi(\epsilon_{\Omega+1}) &= \theta_{\epsilon_{\Omega+1}}(0)\,(\text{バッハマン・ハワード順序数})\end{aligned}$$

この定義では、バッハマン・ハワード順序数より大きくはなりません。

ここで、ψ 関数の「正則な」定義を考えます。たとえば $1+\omega$ という順序数は定義できますが、これはカントール標準形ではないため「正則」ではないと考えます。$\alpha+\beta$ の表記で $\alpha<\beta$ であれば正則ではありません。そして正則な ψ 関数では、ψ 関数の内側に入っている引数は常に外側の関数よりも小さくなります。たとえば $\psi(\psi(\Omega)+1)$ はこの条件を満たさないため、正則な表記ではありません。

バッハマン・ハワード順序数よりも小さい順序数 α の基本列を取る方法について考えます。まずは α を ψ 関数で

$$\alpha=\delta_1^{\beta_1}\gamma_1+\cdots+\delta_k^{\beta_k}\gamma_k$$

の形で正則に表記します。ここで δ_k は ω か $\psi(\alpha)$ です。γ_k が極限順序数であればその基本列を取り、後続順序数であれば $\delta_k^{\beta_k}\gamma_k=\delta_k^{\beta_k}(\gamma_k-1)+\delta_k^{\beta_k}$ とすれば、$\delta_k^{\beta_k}$ の基本列を定めることに帰着します。そして β_k が極限順序数であればその基本列を取り、後続順序数であれば $\delta_k^{\beta_k}=\delta_k^{\beta_k-1}\delta_k$ とすることで、δ_k の基本列を定める問題に帰着します。$\delta_k=\omega$ であれば基本列 $0,1,2,...$ を取り、$\delta_k=\psi(0)$ であれば基本列 $1,\omega,\omega^{\omega},...$ を取り、$\delta_k=\psi(\alpha+1)$ であれば基本列

$$\psi(\alpha),\psi(\alpha)^{\psi(\alpha)},\psi(\alpha)^{\psi(\alpha)^{\psi(\alpha)}},...$$

を取ります。

残りは $\delta_k=\psi(\alpha)$ で α が極限順序数のときです。α の共終数が ω であれば、α の基本列を取ればいいので簡単です。問題は α の共終数が非可算のときにどのように基本列を決めるか、というところです。

α の共終数が非可算のときは、α への基本列を取ることができません。そこで $\rho<\alpha$ かつ $\psi(\rho)=\psi(\alpha)$ となる最小の可算順序数 ρ を見つけます。そのために、まずは α の基本列を取ろうとしたときに、Ω への基本列を取る

必要が出てきて基本列が取れなくなる箇所を見つけて、その Ω を γ と書き換えて他の Ω と区別して、$h(\gamma)$ という関数を考えます。たとえば $\alpha = \Omega 2$ に対しては、$\Omega + \Omega$ の 2 番目の Ω の基本列を取ろうとして取れないことから、2 番目の Ω を γ に変えて $h(\gamma) = \Omega + \gamma$ とします。そして $h(\gamma)$ に $\gamma = \psi(\xi)$ を入れた $h(\psi(\xi))$ を考えると、$\xi = h(\psi(\xi))$ となる最小の順序数が ρ となります。このとき ρ の基本列は、

$$\begin{aligned} \rho[0] &= 0 \\ \rho[n+1] &= h(\psi(\rho[n])) \end{aligned}$$

すなわち、$0, h(\psi(0)), h(\psi(h(\psi(0)))), ...$ となります。$\psi(\alpha) = \psi(\rho)$ の正則な基本列は、この基本列に ψ 関数をかぶせて

$$\begin{aligned} \psi(\alpha)[0] &= \psi(0) \\ \psi(\alpha)[n+1] &= \psi(h(\psi(\alpha)[n])) \end{aligned}$$

すなわち

$$\psi(0), \psi(h(\psi(0))), \psi(h(\psi(h(\psi(0))))), ...$$

となります。

たとえば $\alpha = \Omega 2$ であれば、$h(\gamma) = \Omega + \gamma$ なので、$\psi(\Omega 2)$ の基本列は

$$\psi(0), \psi(\Omega + \psi(0)), \psi(\Omega + \psi(\Omega + \psi(0))), ...$$

です。また $\alpha = \Omega^{\Omega^3 + \Omega^2 4}$ のときは、$h(\gamma) = \Omega^{\Omega^3 + \Omega^2 3 + \Omega\gamma}$ なので、$\psi(\alpha)$ の基本列は

$$\begin{aligned} \psi(\alpha)[0] &= \psi(0) \\ \psi(\alpha)[1] &= \psi(h(\psi(\alpha)[0])) = \psi(\Omega^{\Omega^3 + \Omega^2 3 + \Omega\psi(0)}) \\ \psi(\alpha)[2] &= \psi(h(\psi(\alpha)[1])) = \psi(\Omega^{\Omega^3 + \Omega^2 3 + \Omega\psi(\Omega^{\Omega^3 + \Omega^2 3 + \Omega\psi(0)})}) \end{aligned}$$

となります。

さて、ここまでの定義ではバッハマン・ハワード順序数よりも大きな順序数を表記することはできません。$\psi(\epsilon_{\Omega+1}+1)$ としても、$\epsilon_{\Omega+1}$ を含む $C(\alpha)$ が存在しないため、$\psi(\epsilon_{\Omega+1})$ よりも大きくはならないのです。そこで、さらに大きな順序数を表記するために、2 つ目の非可算順序数 $\Omega_2 = \omega_2$ を導入して、さらに新しく ψ_1 という関数を定義することで（定義の詳細は省略します）、$\psi_1(0) = \epsilon_{\Omega+1}$ となって $\psi(\psi_1(0))$ がバッハマン・ハワード順序数となり、さらに $\psi(\psi_1(\epsilon_{\Omega_2+1}))$ までの可算順序数を作ることができるシステムができます。さらに、定義を修正することで $\psi(\psi_1(\Omega_2))$ と書かずに $\psi(\Omega_2)$ と書けば可算順序数が得られるようにすることができます。

同様に Ω_ω までの非可算順序数を導入して、ψ を ψ_0 とすることで、ブーフホルツの ψ 関数と本質的に同等な定義が得られます。その限界は竹内・フェファーマン・ブーフホルツ順序数 $\psi_0(\epsilon_{\Omega_\omega+1}) = \theta_{\epsilon_{\Omega_\omega+1}}(0)$ となります。この順序数の名前は、マドールが書いた Wikipedia の記事で名づけられたものです。

7.2.5 ワイヤーマンの ϑ 関数

ワイヤーマンの ϑ 関数 [9] (Weiermann's ϑ function) は、次のように定義されます。

【定義】ワイヤーマンの ϑ 関数

$$
\begin{aligned}
C_0(\alpha,\beta) &= \beta \cup \{0, \Omega\} \\
C_{n+1}(\alpha,\beta) &= \{\gamma+\delta, \omega^\gamma, \vartheta(\eta) \mid \gamma,\delta,\eta \in C_n(\alpha,\beta); \eta < \alpha\} \\
C(\alpha,\beta) &= \bigcup_{n<\omega} C_n(\alpha,\beta) \\
\vartheta(\alpha) &= \min\{\beta < \Omega \mid C(\alpha,\beta) \cap \Omega \subseteq \beta \wedge \alpha \in C(\alpha,\beta)\}
\end{aligned}
$$

これは、定義は少し複雑ですが 1 変数で簡潔に表記できます。Googology

[9] Rathjen, M. and Weiermann, A. (1993) Proof theoretic investigations of Kruskal's theorem. *Annals of Pure and Applied Logic* 60(1), 49–88. doi:10.1016/0168-0072(93)90192-G

Wiki では、フェファーマンの θ 関数と区別するために、ギリシャ文字 θ のフォントに ϑ を使っているので、本書でもそのようにします。

この定義ですべての $\alpha < \epsilon_{\Omega+1}$ に対して $\vartheta(\alpha)$ が定義されます。ここで $C(\alpha,\beta)$ は「0」「β よりも小さい順序数」「Ω」から、有限回の「加算」「ω^γ」「$\vartheta(\eta)$（すでに $\vartheta(\eta)$ が定義されている場合に限る）」を適用した順序数すべての集合です。そして $\vartheta(\alpha)$ は $\alpha \in C(\alpha,\beta)$ であり、かつ $C(\alpha,\beta)$ に属するいかなる可算順序数よりも大きい順序数 β の中で最小のものです。

この定義を使うと、たとえば次のようになります。

$$\begin{aligned}
\vartheta(\Omega^3) &= \theta_{\Omega^3}(0) = \psi(\Omega^{\Omega^2}) = \phi(1,0,0,0)\\
\vartheta(\Omega^\omega) &= \theta_{\Omega^\omega}(0) = \psi(\Omega^{\Omega^\omega})\,(\text{小ヴェブレン順序数})\\
\vartheta(\Omega^\Omega) &= \theta_{\Omega^\Omega}(0) = \psi(\Omega^{\Omega^\Omega})\,(\text{大ヴェブレン順序数})\\
\vartheta(\epsilon_{\Omega+1}) &= \theta_{\epsilon_{\Omega+1}}(0) = \psi(\epsilon_{\Omega+1})\,(\text{バッハマン・ハワード順序数})
\end{aligned}$$

ワイヤーマンの ϑ 関数では、バッハマン・ハワード順序数よりも大きな順序数を表記することはできません。ただし Googology Wiki では次に紹介する順序数崩壊関数の拡張による表記もされているようです。

7.2.6 順序数崩壊関数の拡張

Googology Wiki ユーザーの Deedlit11 は、フェファーマンの $\theta_\alpha(\beta)$ 関数、ワイヤーマンの $\vartheta(\alpha)$ 関数、ブーフホルツの $\psi_0(\alpha)$ 関数を、それぞれ $\alpha = \psi_I(0)$（オメガ収束点）まで拡張できることを Googology Wiki のブログで解説しています。まずは、ワイヤーマンの ϑ 関数には $\Omega_\nu(\nu \leq \omega)$ を導入して他の 2 つと同等の強さにしてから、C の定義に Ω_α を入れて、$\theta_{\Omega_\Omega}(\beta)$ が $\alpha = \theta_{\Omega_\alpha}(0)$ となる β 番目の順序数、$\vartheta(\Omega_\Omega)$ が $\alpha = \vartheta(\Omega_\alpha)$ となる最小の順序数、$\psi_0(\Omega_\Omega)$ が $\alpha = \psi_0(\Omega_\alpha)$ となる最小の順序数となるように定義します。すると、それぞれの順序数崩壊関数によって、オメガ収束点までの順序数を崩壊させた可算順序数を定義できます。Deedlit11 は、さらにオメガ収束点の先の弱到達不能基数や弱マーロ基数のような基数を崩壊する方法について書いています。

7.3　2 階算術

7.3.1　逆数学

公理から定理を証明するのが通常の数学の手順ですが、それを逆に考えて、ある定理を証明するためにどのような公理が必要か、と考える**逆数学** (reverse mathemetics) を、アメリカの数理論理学者のハーヴェイ・フリードマン (Harvey Friedman) が提唱し、スティーブ・シンプソン (Steve Simpson) が確立しました[10]。

順序数解析の節 (p.139) では、ある理論体系である定理が証明できるかどうかを調べるということは、ある形式体系に定められている言語で、公理をスタートとして推論規則を繰り返し適用して定理を示すことができるかどうかを調べることであると説明しました。理論体系 A では証明できない定理が理論体系 B では証明できるときに、理論体系 B は理論体系 A よりも証明力が強いと言います。それでは、ある定理を証明するために「最低限必要な強さを持った理論体系」は何であろうか？ と考えるのが逆数学です。

7.3.2　2 階算術

逆数学では通常 **2 階算術** (second-order arithmetic) Z_2 の部分体系を研究対象とします。逆数学で ZFC のような集合論の公理をベースとしない理由は、集合論は表現力が高すぎて、逆数学で必要な「弱い理論体系」を作りにくいためです。2 階算術は、数学におけるかなりの定理を証明できる強力な理論体系です。そこで 2 階算術の部分体系である RCA_0 という理論体系を基本の理論体系として、RCA_0 に公理を加えることで色々な強さの理論体系を作ります。

自然数を議論領域とするペアノ算術は **1 階算術** (first-order arithmetic) と言われますが、2 階算術は議論領域が数と数の集合へと広がります。そのために、変数には数変数と集合変数の 2 種類があり、次のような記号を使います。

[10] Simpson, S. G. (2009) Subsystems of second order arithmetic, Perspectives in Logic (2nd ed.). Cambridge University Press, England.

1. 定数記号ゼロ 0
2. 等号、不等号、集合の要素と呼ばれる 3 つの二項関係の記号 $=, <, \in$。最初の 2 つは数の間の関係、最後の 1 つは数と集合の間の関係
3. 数から数へ写す、後者と呼ばれる単項の関数記号 S
4. 2 組の数を 1 つの数へ写す、それぞれ加法、乗法と呼ばれる 2 つの二項関数記号 $+(a,b), \cdot(a,b)$ 。 それらは $a+b, a\cdot b$ とも表示される。

2 階算術の公理は、ロビンソン算術に以下の公理を加えたものです。

1. **2 階帰納公理** (second-order induction axiom): $\forall X((0 \in X \wedge \forall x(x \in X \to Sx \in X)) \to \forall x(x \in X))$ すなわち、集合 X が 0 を要素に持ち、X のすべての要素の後者が X の要素であるときに、すべての自然数は X の要素である。
2. **内包の公理型** (axiom schema of comprehension): X が自由変数でないすべての論理式 ϕ ごとに $\exists X \forall x(x \in X \leftrightarrow \phi(x))$ 。すべての自然数に関する述語ごとに自然数の部分集合 (すなわち ϕ を満たすこれらの自然数からなる集合) が定義されることを意味する。

内包の公理型の「内包」とは何でしょうか。集合論において「内包」は「外延」に対する語で、たとえば集合の要素を列記することで集合を記すことは「外延」で、条件 $P(x)$ があったときに $\{x \mid P(x)\}$ のように、ある条件を満たすかどうかで集合を表記する方法が「内包」です。つまり述語 ϕ によって「ϕ を満たすすべての自然数」という集合を定義できる、ということが「内包」という意味です。

ZF 公理系では、内包の公理型は**制限付き内包の公理型** (axiom schema of restricted comprehension) あるいは**分出の公理型** (axiom schema of specification) として登場します。**置換の公理型** (axiom schema of replacement) から分出の公理型を証明できるため、現代では ZF 公理系から分出公理が省かれることも多いのですが、ゲーデルは分出公理が集合論における最も重要な公理であると考えていたようです。次のような公理です。

$$\forall X \exists A \forall x(x \in A \leftrightarrow (x \in X \wedge \phi(x)))$$

「制限付き」内包の公理型としているのは、ここで述語 ϕ を適用する範囲を $X \wedge \phi(x)$ のように集合 X の部分集合に制限していることを意味します。これは、$\phi(x) = \neg(x \in x)$ とするようなラッセルのパラドックスを回避するためです。2 階算術における内包の公理型では、述語を適用する範囲を自然数に制限しているので、パラドックスは回避されています。ここで内包公理は「すべての ϕ」に対する公理ですが、$\forall\phi$ という二階述語論理の表現は一階述語論理ではできないので、すべての ϕ に対する公理型となっています。

7.3.3　論理式の階層

次に 2 階算術の部分体系について説明したいのですが、そのためには論理式の階層について説明をしておく必要があります。まず、ペアノ算術には**算術的階層** (arithmetical hierarchy) があります。

一階述語論理におけるすべての論理式は、**冠頭標準形** (prenex normal form) という量化子が論理式の先頭に集められた形に書き直すことができます。たとえば

$$\forall x((\exists y\phi) \vee ((\forall z\psi) \to \rho))$$

という論理式は冠頭標準形ではありませんが、これと等価な冠頭標準形の論理式

$$\forall x\exists y\exists z(\phi \vee (\psi \to \rho))$$

に書き換えることができます。ある論理式の算術的階層は、この冠頭標準形の論理式に書き直してから考えます。

算術的階層は Σ^0_n あるいは Π^0_n と書かれます。ここで n は自然数です。論理式が量化子を含まないときは Σ^0_0 であり、また Π^0_0 です。ここで有界量化子 $\forall n < t$ と $\exists n < t$ は量化子として数えません。そして Σ^0_n と Π^0_n は以下のように帰納的に定義されます。

1. ψ が Π^0_n であるとき、$\exists n_1 \cdots \exists n_k\psi$ と論理的に等価な式 ϕ は、階層 Σ^0_{n+1} に相当する。

2. ψ が Σ^0_n であるとき、$\forall n_l \cdots \forall n_k\psi$ と論理的に等価な式 ϕ は、階層 Π^0_{n+1} に相当する。

たとえば ψ が量化子を含まないとき、$\exists w \forall x \forall y \exists z \psi$ について考えると、ψ は Π^0_0、$\exists z \psi$ は Σ^0_1、$\forall x \forall y \exists z \psi$ は Π^0_2、そして $\exists w \forall x \forall y \exists z \psi$ は Σ^0_3 となります。

2 階算術では、同様に解析的階層 (analytical hierarchy) で集合量化子の階層を考えます。集合の量化子を含まない式は $\Sigma^1_0 = \Pi^1_0 = \Delta^1_0$ です。算術的階層と同様に、集合量化子に関する階層 Σ^1_n と Π^1_n を帰納的に定義します。ここで Σ^1_n あるいは Π^1_n に属する式は、n よりも大きい m に対して、Σ^1_m と Π^1_m に属します。自然数の集合が Σ^1_n 式で定義できれば Σ^1_n に属し、Π^1_n 式で定義できれば Π^1_n に属し、Σ^1_n と Π^1_n の両方に属すときに Δ^1_n に属します。

7.3.4 2 階算術の部分体系と証明論的順序数

2 階算術の部分体系には、ビッグ 5 といわれる代表的なものがあります。以下に、ビッグ 5 の理論体系を次第に強くなるように簡単に紹介し、それぞれの証明論的順序数を書きます。ここでは、おおまかな概念を書くにとどめます。正確かつ詳細な解説は本書の著者には無理なので、先に引用したスティーブ・シンプソンの本を参照してください。ここでそれぞれの名前に下付き文字の 0 がついている理由は、完全な 2 階帰納公理を使うことができずに、帰納法の公理が制限されていることを示しています。

1. 逆数学の基本体系 RCA_0 の証明論的順序数は ω^ω です。RCA は再帰的内包公理 (recursive comprehension axiom) を意味します。「ロビンソン算術の公理」＋「Σ^0_1 論理式に対する帰納法の公理」＋「Δ^0_1 論理式に対する内包公理」です。ここで、Δ^0_1 は Σ^0_1 または Π^0_1 のいずれかです。RCA_0 は弱い理論体系に見えますが、多くの数学的な定理を証明できます。自然数、整数、有理数に関する様々な性質や、実数に関する基本的な性質、実数の連続関数における中間値の定理を証明できます。これが逆数学の基本体系なので、逆数学の理論体系はすべて RCA_0 に公理を加えることでできます。

2. WKL_0 の証明論的順序数は ω^ω です。RCA_0 に弱いケーニッヒの補題 (weak form of König's lemma) を加えたものです（WKL の名前の由来)。証明論的順序数は RCA_0 と同じですが、RCA_0 では証明できない多くの数学の定理を証明できます。

3. ACA_0 の証明論的順序数は ϵ_0 です。これは、RCA_0 にすべての 1 階算術の式（Σ^0_n の式）に対する内包公理、すなわち算術的内包公理 (arithmetical comprehension axiom) を加えたものです。つまり、ACA_0 は任意の算術的な式を満たす自然数の集合を作ることができるということです。ペアノ算術と同じ強さになります。ACA_0 は可述的な理論だと考えることができますが、可述的に証明可能な定理の中にも、ACA_0 で証明できないものがあります。ACA_0 では自然数の基本的な性質のほとんどと、様々な数学の定理を証明できます。たとえば「有界な実数列は収束部分列を持つ」というボルツァーノ・ワイエルシュトラスの定理 (Bolzano-Weierstrass theorem) を証明できます。

4. ATR_0 の証明論的順序数はフェファーマン・シュッテの順序数 Γ_0 です。これは 2 変数ヴェブレン関数の限界であり、可述的な理論体系の限界です。ATR_0 は ACA_0 に算術的超限再帰 (arithmetical transfinite recursion) の公理型を加えたものです。公理の内容を直感的に表現すると、ある自然数の集合に対して、算術的な汎関数 (arithmetical functional) を適用して新しい自然数の集合とする、ということを、ある可算の整列集合に順番に対応させながら（たとえば ϵ_0 回）繰り返すことができる、ということです。

5. Π^1_1-CA_0 の証明論的順序数は $\psi_0(\Omega_\omega)$ です。これは RCA_0 に Π^1_1 式の内包公理型を加えたものです。これはいわゆる非可述的な体系 (impredicative system) です。Π^1_1 式、つまり冠頭標準形で $\forall$ のみを使った式を使って集合 X を定義するということは、その Π^1_1 式の量化子の変数に自分自身 (X) を代入してもその式が成り立つということを意味しています。自分自身を定義に使うということは、非可述的な定義ということになります。つまり非可述的な定義を許す公理が Π^1_1 式の内包公理型です。なお、Π^1_1-CA_0 で ACA_0 や ATR_0 の公理を証明できます。

ビッグ 5 以外にも、色々な 2 階算術の部分体系が解析されています。その中のいくつかを以下に紹介します。ここで、すべて n は 0 か 1 です。

- RCA や WKL のように下付き文字の 0 がないときには、2 階帰納公理が使えます。

- Σ^n_m-IND と Π^n_m-IND と Δ^n_m-IND は、Σ^n_m と Π^n_m と Δ^n_m それぞれの論理式に制限した 1 階の帰納法 (induction) です。Π^1_∞-IND はこれらすべての公理型を足したものです。

- Σ^n_m-TI と Π^n_m-TI と Δ^n_m-TI は、それぞれのクラスの論理式に制限した超限帰納法 (transfinite induction) の公理型です。つまり、それぞれの階層の複雑性を持った $\psi(n)$ 論理式と、自然数に関する整列順序が入っている集合 X について、$\forall n : \psi(n)$ を整列順序によって証明可能ということです。上と同様に、Π^1_∞-TI が定義できます。

- Σ^n_m-CA と Π^n_m-CA と Δ^n_m-CA は、それぞれのクラスの論理式に制限した内包公理 (comprehension axiom) です。すべてを入れた Π^1_∞-CA は、2 階算術 Z_2 と等価です。

- BI は、バー帰納法 (bar induction) の公理型です。ごく簡単に直感的な表現をすると、算術的な論理式であらわされる整礎な二項関係と、任意の論理式に対して、超限帰納による証明が可能であることを意味します。

- Σ^n_m-AC と Π^n_m-AC と Δ^n_m-AC は、ZFC の選択公理 (axiom of choice) を 2 階算術に「取り入れた」公理型です。この公理は、すべての n に対して、それぞれの階層の論理式に対応する $\psi(n, X)$ が成立するような集合 X が存在するのであれば、 $\psi(n, X_n)$ が成立するような集合の列 X_n を作ることができる、というものです。

- Σ^n_m-DC と Π^n_m-DC と Δ^n_m-DC は、従属選択公理 (axiom of dependent choice) という名前の、弱い形の選択公理を 2 階算術に取り入れたものです。すべての n と X に対して、$\psi(n, X, Y)$ が成立するような Y が存在するのであれば、 $\psi(n, X_n, X_{n+1})$ が成立するような集合の列 X_n を作ることができる、という公理です（ややおおざっぱな書き方をしています)。

- Σ^n_m-TR と Π^n_m-TR と Δ^n_m-TR は、それぞれのクラスの論理式に対する超限再帰 (transfinite recursion) です。つまり算術的超限再帰を拡張したものです。直感的な表現をすれば、ある操作を超限的に繰り返すことです。Googology Wiki の Deedlit11 によれば、Π^1_1-TR_0 の証明論的順序数は、順序数崩壊関数をオメガ収束点にまで拡張したものに相当するようです。

7.3.5　タラノフスキーの C 表記

タラノフスキーの **C 表記** (Taranovsky's C notation) は、1983 年にウクライナのキエフに生まれて 1996 年にアメリカに移住し、MIT で数学を専攻したドミトロ・タラノフスキー (Dmytro Taranovsky) が MIT のサーバーに公開した大きな順序数の表記法です[11]。まずは一般的な整列集合、そして順序数に対する崩壊関数 $C(a,b)$ を定義しています。典型的な C 関数の定義では、a が b よりも非常に大きいときに $C(a,b) < a$ となって、C は b よりも大きい a を崩壊させる働きを持っていることから、崩壊関数と呼ばれるとしています。この一般の C 表記に 0 と Ω を導入すれば、バッハマン・ハワード順序数が定義されます。そして、再帰到達不能度 (degree of recursive inaccessibility) やリフレクション度 (degree of reflection) を定義して、それぞれの degree を崩壊する順序数崩壊関数を定義しました。さらに Π^1_n-CA の強さ、すなわち 2 階算術の強さを持つ C 関数を定義しました。

7.4　様々な巨大数と関数

このレベルの巨大数を生み出す関数・システムや巨大数を表記する記法は、いずれも複雑なものばかりです。その強さは、ここまでに示したような順序数に対する順序数階層や順序数解析によって評価されますが、あまりにも強力な理論になってくると、対応する順序数を決めることも難しくなってきます。この章でこれから紹介する巨大数の前半は、対応する順序数が分かっていますが、途中からはまだ対応する順序数もよく分からず、お互いにどちらの方が強いのかはっきりしなくなってきます。よってこの章

[11] Ordinal Notation http://web.mit.edu/dmytro/www/other/OrdinalNotation.htm

で示す巨大数や関数は、必ずしも強さの順番となっているかどうかははっきりしていません。ローダー数以降は、理論体系の強さによって関数の強さが直接評価され、さらにはその理論体系の強さが巨大基数の強さによって評価されるような巨大数も出現します。

7.4.1 ふぃっしゅ数バージョン 6

本書『巨大数論』を書き始めて「まだ書きかけ」版の PDF ファイルを公開した 2007 年 10 月 27 日に、その PDF ファイルの中でふぃっしゅ数バージョン 6 を定義しました。その時期は、巨大数探索スレッドの議論でふぃっしゅ数バージョン 5 が $F[\epsilon_0]$ のレベルに達することが解析されて、そのことを本書の著者が理解できたころです。そこで、バージョン 5 までの考え方をさらに進めて「より抽象度の高い変換」を定義することで、どの程度の大きさの巨大数が作れるかと考えて作ったものです。その大きさは $F[\zeta_0]$ のレベルです。それは大きな数ではありますが、$F[\Gamma_0]$ に相当するような巨大数をこの定義の拡張で作るのはかなり無理そうで、作ったとしても難解な定義になるだろうと思います。

著者としては「ふぃっしゅ数」の計算可能な定義の拡張は、これ以上はあまり面白くないと考えてこのバージョン 6 で止めました。

【定義】ふぃっしゅ数バージョン 6

1. 集合 $M[m,n](m=0,1,\ldots;n=1,2,\ldots)$ を以下のように定める。

   $M[0,1]$=自然数から自然数への関数全体の集合

   $M[m+1,1]=M(m,n)(n=1,2,\ldots)$ の元 1 個ずつを要素に持つ集合、すなわち $M[m+1,1]=M[m,1]\times M[m,2]\times M[m,3]\times\ldots$

      $M[m,1]$ の元は、その要素の要素の…要素である $M[0,1]$ の元と同じ関数の働きを持つ。

   $M[m,n+1]$=写像 $M[m,n]\to M[m,n]$ 全体の集合 $(n=1,2,\ldots)$

2. $M[m,n]$ の元 $m(m,n)$ を以下のように定める[a]。

$$\begin{aligned}
m(0,1)(x) &:= x+1 \\
m(m,n+1)f_nf_{n-1}...f_1(x) &:= {f_n}^x f_{n-1}\dots f_1(x) \\
&\quad (m=0; n=1,2,\dots \\
&\quad \text{or } m=1,2,\dots; n=2,3,\dots) \\
m(m+1,1) &:= [m(m,1), m(m,2), m(m,3), ...] \\
m(m+1,2)[a_1,a_2,...] &:= [b_1,b_2,...] \text{ の } b_n \text{を以下で定める。} \\
b_nf_{n-1}\dots f_1(x) &:= a_ya_{y-1}...a_nf_{n-1}\dots f_1(x) \\
&\quad (y=\max(x,n))
\end{aligned}$$

3. ふぃっしゅ関数 $F_6(x)$ を以下のように定める。

$$F_6(x) := m(x,2)m(x,1)(x)$$

4. ふぃっしゅ数 $F_6 := F_6^{63}(3)$ とする。

[a] ただし、a_i, b_i, f_i は $m(m,i)$ の元とし、厳密な定義の構造は F_5 と同じである。$m(m,n+1)f_nf_{n-1}...f_1 \in M(m,1) = M(0,1)\times\Pi_{0\le k<m,1<l}M(k,l)$ を、$M(0,1)-$成分は与えられた式の通りに、その他の成分は $f_1 \in M(m,1)$ を参照して定義する。$m(m+1,2)$ の式についても同様に定義する。

F_6 の大きさは、次のように計算できます。

$$\begin{aligned}
m(1,1) &= [m_1,m_2,m_3,\cdots](m_i=m(0,i)) \\
&= x+1 \\
m(1,2)m(1,1) &= [a_1,a_2,a_3,\cdots] \text{ とすると} \\
a_1(x) &= m_xm_{x-1}...m_1(x) \text{ より、} F_5 \text{と同様の計算で} \\
a_1 &\approx H[\epsilon_0] \\
\text{すなわち } m(1,2)m(1,1) &\approx H[\epsilon_0] \\
a_2f_1(x) &= m_ym_{y-1}\cdots m_2f_1(x)(y=\max(x,2)) \text{ より} \\
m_2 &= H[\times\omega](H[a]\to H[a\times\omega] \text{ の変換})
\end{aligned}$$

$$\begin{aligned} m_3m_2 &= H[\times\omega^{\omega}] \\ m_3m_3m_2 &= H[\times\omega^{\omega^2}] \\ m_4m_3m_2 &= H[\times\omega^{\omega^{\omega}}] \\ &\vdots \\ a_2 &= H[\times\epsilon_0] \end{aligned}$$

となります。そして

$$\begin{aligned} a_3f_2f_1(x) &= m_ym_{y-1}...m_3f_2f_1(x)(y=\max(x,3)) \text{ より、} \\ f_2 &= H[\times a] \text{ とすると} \\ m_3f_2 &= H[\times a^{\omega}] \\ m_3m_3m_2 &= H[\times a^{\omega^2}] \\ m_4m_3f_2 &= H[\times a^{\omega^{\omega}}] \\ &\vdots \\ a_3f_2 &= H[\times a^{\epsilon_0}] \end{aligned}$$

となるため、a_3は $H[\times a] \to H[\times a^{\epsilon_0}]$ の変換です。

$$\begin{aligned} a_4f_3f_2f_1(x) &= m_ym_{y-1}...m_4f_3f_2f_1(x)(y=\max(x,4)) \text{ より、} \\ f_2 &= H[\times a]、f_3 = H[\times a] \to H[\times a^b] \text{ の変換とすると} \\ f_2 &= H[\times a] \\ f_3f_2 &= H[\times a^b] \\ m_4f_3f_2 &= H[\times a^{b^{\omega}}] \\ m_5m_4f_3f_2 &= H[\times a^{b^{\omega^{\omega}}}] \\ &\vdots \\ a_4f_3f_2 &: H[\times a^{b^{\epsilon_0}}] \end{aligned}$$

となるため、a_4 は $[H[\times a] \to H[\times a^b]] \to [H[\times a] \to H[\times a^{b^{\epsilon_0}}]]$ の変換です。このことから、

$$a_4a_3 \quad : \quad H[\times a] \to H[\times a^{\epsilon_0^{\epsilon_0}}] \text{ の変換}$$

$$a_4a_3a_2 \quad : \quad H[\times{\epsilon_0}^{{\epsilon_0}^{\epsilon_0}}]$$

と計算され、以下同様に a_5 は $[H[\times a] \rightarrow H[\times a^b]] \rightarrow [H[\times a] \rightarrow H[\times a^{b^c}]]$ $\rightarrow [H[\times a] \rightarrow H[\times a^b]] \rightarrow [H[\times a] \rightarrow H[\times a^{b^{c^{\epsilon_0}}}]]$ の変換となって、

$$\begin{aligned} a_5a_4a_3a_2 &: H[\times{\epsilon_0}^{{\epsilon_0}^{{\epsilon_0}^{\epsilon_0}}}] = H[\times{\epsilon_0}^{\wedge\wedge}4] \\ a_6a_5a_4a_3a_2 &: H[\times{\epsilon_0}^{\wedge\wedge}5] \\ a_{n+1}a_n...a_2 &: H[\times{\epsilon_0}^{\wedge\wedge}n] \end{aligned}$$

と計算され、

$$m(1,2)^2m(1,1) \approx H[{\epsilon_0}^{\wedge\wedge}\omega] = H[\epsilon_1]$$

となります。そして

$$m(1,2)^3m(1,1) = [b_1, b_2, b_3, ...]$$

とすると、b_i は上記 a_i の ϵ_0 を ϵ_1 に変えた式となります。したがって、

$$\begin{aligned} m(1,2)^3m(1,1) &\approx H[{\epsilon_1}^{\wedge\wedge}\omega] = H[\epsilon_2] \\ m(1,2)^4m(1,1) &\approx H[{\epsilon_2}^{\wedge\wedge}\omega] = H[\epsilon_3] \end{aligned}$$

といったように、$m(1,2)$ は $H[\epsilon_a]$ の a に 1 を足す効果を持ちます。このことから F_5 の計算と同様の構造で

$$\begin{aligned} m(1,3)m(1,2)m(1,1)(x) &\approx H[\epsilon_\omega] \\ m(1,4)m(1,3)m(1,2)m(1,1)(x) &\approx H[\epsilon_{\omega^\omega}] \\ m(1,5)m(1,4)m(1,3)m(1,2)m(1,1)(x) &\approx H[\epsilon_{\omega^{\omega^\omega}}] \\ m(1,x)m(1,x-1)...m(1,2)m(1,1)(x) &\approx H[\epsilon_{\epsilon_0}] \\ m(2,2)m(2,1) &\approx H[\epsilon_{\epsilon_0}] \end{aligned}$$

となります。そして、$m(2,2)m(2,1) = [f_1, f_2, f_3, \cdots]$ とすると (f_i は $M[1,i]$ の元)

$$f_1 \quad = \quad H[\epsilon_{\epsilon_0}] \text{ 程度の関数を元に持つ } M(1,1)$$

$$\begin{aligned} f_2 &: H[\epsilon_{\times\epsilon_0}](H[\epsilon_a] \to H[\epsilon_{a\times\epsilon_0}] \text{ の変換}) \\ f_3 &: H[\epsilon_{\times a}] \to H[\epsilon_{\times a^{\epsilon_0}}] \text{ の変換} \\ f_4f_3f_2 &: H[\epsilon_{\times\epsilon_0{}^{\epsilon_0}{}^{\epsilon_0}}] \end{aligned}$$

となることから

$$\begin{aligned} m(2,2)^2m(2,1) &\approx H[\epsilon_{\epsilon_1}] \\ m(2,2)^3m(2,1) &\approx H[\epsilon_{\epsilon_2}] \\ m(2,2)^4m(2,1) &\approx H[\epsilon_{\epsilon_3}] \end{aligned}$$

と計算されます。このように、$m(2,2)$ が $H[\epsilon_{\epsilon_a}]$ の a に 1 を足す効果を持つので

$$m(3,2)m(3,1) \approx H[\epsilon_{\epsilon_{\epsilon_0}}]$$

となります。

以上の計算を繰り返すと、

$$\begin{aligned} m(1,2)m(1,1) &\approx H[\epsilon_0] \\ m(2,2)m(2,1) &\approx H[\epsilon_{\epsilon_0}] \\ m(3,2)m(3,1) &\approx H[\epsilon_{\epsilon_{\epsilon_0}}] \\ m(4,2)m(4,1) &\approx H[\epsilon_{\epsilon_{\epsilon_{\epsilon_0}}}] \end{aligned}$$

となり

$$F_6(x) = m(x,2)m(x,1)(x) \approx H[\zeta_0] \approx F[\zeta_0]$$

と計算されます。よって

$$F_6 \approx F[\zeta_0 + 1](63)$$

です。

7.4.2 ペア数列数

ペア数列数 (pair sequence number) は、バシクが 2014 年に巨大数探索スレッドに投稿した BASIC プログラム[12] によって表示される数です。2014

[12] 巨大数探索スレッド 10 https://wc2014.5ch.net/test/read.cgi/math/1384444271/167

年に投稿したプログラムは計算が終了しない（有限の数が表示されない）ことが示されたため、バシクが 2017 年 10 月 2 日に更新をしたバージョンを示します。なお、読みやすくするためのちょっとした書き換え（endif を追加する等）をしました。

```
dim A(Infinity):dim B(Infinity):C=9
for D=0 to 9
  for E=0 to C
    A(E)=E:B(E)=E
  next
  for F=C to 0 step -1
    C=C*C
    for G=0 to F
      if A(F)=0 or A(F-G)<A(F)-H then
        if B(F)=0 then
          I=G:G=F
        else
          H=A(F)-A(F-G)
          if B(F-G)<B(F) then I=G:G=F endif
        endif
      endif
    next
    for J=1 to C*I
      A(F)=A(F-I)+H:B(F)=B(F-I):F=F+1
    next
    H=0
  next
next
print C
```

ペア数列数は $F[\psi_0(\Omega_\omega)+1](10)$ 程度の大きさです。なお、このプログラムは計算アルゴリズムを示すためのもので、無限配列を宣言するといっ

たコードが入っていて、実際に動かせるものではありません。このアルゴリズムをペア数列システム[13] (pair sequence system) と言います。このプログラムは、バシクによる原始数列数 (p.160) のプログラムを拡張したものです。そしてバシク行列 (p.222) へと発展します。なお、バシク行列で紹介する予定のバシク行列計算機のホームページから、ペア数列の計算過程を検証できます。バシク行列計算機にはバシク行列の複数のバージョンがありますが、バシクが 2014 年に発表したプログラムはバージョン 1 に対応し、ここに記載したプログラムはバージョン 2 に対応します。

ペア数列 (pair sequence) は、(0,0)(1,1)(2,2)(3,3)(3,2) のような非負整数ペアの有限長数列です。ペア数列 P は、自然数 n から自然数 $P[n]$ への関数として働き、 (0,0)(1,1)(2,2)(3,3)(3,2)[n] のように書きます。関数 $P[n]$ をハーディ階層によって増加速度を近似したときの順序数を α としたとき、ペア数列 P そのものが順序数 α をあらわすものとします。たとえば $(0,0)(1,1)(2,2)(3,3)(3,2) = \psi(\psi_1(\Omega_2))$ のように表記します。

それでは、ペア数列数のプログラムを追いながら計算アルゴリズムを説明します。このプログラムではペアの 1 つ目を A、2 つ目を B として

$$(A(0),B(0))(A(1),B(0))(A(2),B(2))...(A(F),B(F))$$

のようにペア数列を変数に格納しています。

for D=0 to 9 から next のループ (変数 D のループ) 内で関数 $\mathrm{Pair}(C) = (0,0)(1,1)\dots(C,C)[C]$ を計算して、この計算を 10 回繰り返して、最後に print C で結果を表示しています。変数 D のループの中には、変数 E と変数 F の 2 つのループがあります。最初の E のループは、ペア数列 $(0,0)(1,1)...(C,C)$ を作成しています。次の F のループで、$F = C$ として、関数 $\mathrm{Pair}(F) = (0,0)(1,1)\dots(F,F)[F]$ を計算しています。

F ループの中では、まず最初に $C = C * C$ として、C の値を増やしています。原始数列では $f(n) = n^2$ という関数を定義していたところに相当します。そして、G のループでは原始数列でいうところの「悪い部分」を探して、次の J のループでその「悪い部分」をコピーします。まずは G の

[13]巨大数研究 Wiki - ペア数列数 http://ja.googology.wikia.com/wiki/ペア数列数

ループについて見ると、for ループの中に if が 3 重に入っているので複雑です。パラメータの意味について確認すると、F は「一番右のペア」を示すインデックスで A(F) は「一番右のペアの 1 つ目の数」で B(F) は「一番右のペアの 2 つ目の数」です。if 文が多重になっていて大変なので、if B(F)=0 then に着目して、B(F)=0 が成立するかどうか、つまり「一番右のペアの 2 つ目の数」が 0 であるかどうかによって場合分けをして考えます。

まずは一番右のペアの 2 つ目の数が 0 のときです。これをアルゴリズム A とします。このときは、G のループは

```
for G=0 to F
  if A(F)=0 or A(F-G)<A(F) then
    I=G:G=F
  endif
next
```

と等価になります。ここで BASIC では H=0 で初期化されるため、A(F)-H と書いてあったところを A(F) と書き直しました。このときはペアの 1 つ目の数に関する原始数列のアルゴリズムと同じになります。右端のペアの 1 つ目の数 $A(F)$ よりも、ペアの 1 つ目の数 $A(F-I)$ が小さいようなペアの中で、一番右側にあるペアを探しています（G で検索していますが、見つかった段階で $I=G$ としているので $A(F-I)$ としました）。$A(F-I)$ から $A(F-1)$ までが、原始数列の定義で言うところの「悪い部分」に相当するペアで、I 個のペアがあります。ただし、$A(F)=0$ のときには $I=0$ として、「悪い部分」はなくします。続いて、J のループでは一番右端を取り除き、悪い部分を C 個コピーします。変数 F はコピーされた後の数列の右端の位置となります。つまりアルゴリズム A は原始数列と同じで、以下のように順序数と対応します。

$$\begin{aligned}(0,0) &= 1\\ (0,0)(0,0) &= 2\\ (0,0)(0,0)(0,0) &= 3\\ (0,0)(1,0) &= \omega\end{aligned}$$

$$
\begin{aligned}
(0,0)(1,0)(0,0)(0,0) &= \omega+2\\
(0,0)(1,0)(0,0)(1,0) &= \omega\cdot 2\\
(0,0)(1,0)(1,0) &= \omega^2\\
(0,0)(1,0)(1,0)(0,0)(1,0) &= \omega^2+\omega\\
(0,0)(1,0)(2,0) &= \omega^\omega\\
(0,0)(1,0)(2,0)(3,0) &= \omega^{\omega^\omega}
\end{aligned}
$$

続いてアルゴリズム B、つまり一番右端のペアの 2 つ目の数が 0 ではない場合を見ます。G のループは

```
for G=0 to F
  if A(F)=0 or A(F-G)<A(F)-H then
    H=A(F)-A(F-G)
    if B(F-G)<B(F) then I=G:G=F endif
  endif
next
```

と等価になります。この場合も G のループで「悪い部分」を探しますが、今度は $A(F-I)<A(F)-H$ と $B(F-I)<B(F)$ の両方が成り立つペア、つまりペアの 1 つ目の数がそれよりも右のいかなるペアの 1 つ目の数よりも小さく、かつペアの 2 つ目の数が右端のペアの 2 つ目の数よりも小さいようなペアの中で、一番右端のものを検索します（ここで、バージョン 1 ではペアの 1 つ目の数も 2 つ目の数もともに右端のペアよりも小さいようなペアを探します。ここだけが、バージョン 1 と異なるところです)。そして $H=A(F)-A(F-I)$ として、1 番目の数の差を変数 H に格納します。

そして J のループで「悪い部分」をコピーしますが、そのときに $A(F)=A(F-I)+H$ と H を足すことで、1 つコピーが作られるごとにペアの 1 つ目の数が H ずつ大きくなります。

バージョン 1 とバージョン 2 の違いについて、簡単に説明しておきます。次の数列の「悪い部分」を考えます。

$$(0,0)(1,1)(2,2)(3,0)(2,1)(3,1)(4,1)$$

一番右のペアの 2 つ目が 0 ではないので、アルゴリズム B です。バージョン 1 では、一番右の $(4,1)$ よりも、1 つ目の数も 2 つ目の数もともに小さいような一番右のペアである $(3,0)$ から先の $(3,0)(2,1)(3,1)$ が「悪い部分」となります。このパターンではバージョン 1 の計算が終了しないということを、2018 年 3 月 25 日に Bubby3 が Googology Wiki のブログで指摘したことを受けて、バシクがペア数列のプログラムを書き換えたものが本章で紹介したバージョン 2 のプログラムです。与えられた条件を満たすペアは $(0,0)$ であり、$(0,0)(1,1)(2,2)(3,0)(2,1)(3,1)$ が「悪い部分」となります。

この計算アルゴリズムの仕組みを見るためには、ペア数列 P に対して $P[1], P[2], P[3], ...$ を順番に計算したときの挙動を見るのが分かりやすいです。そこで、まずは $P = (0,0)(1,1)$ についてその計算をしてみます。$f(n) = n^2$ では強すぎて分かりにくいので、$f(n) = n$ としておきます。右端のペアの 2 つ目の数は 1 なので、アルゴリズム B です。$(0,0)(1,1)[n]$ は $(0,0)$ のみが「悪い部分」で、n 個コピーされます。$L = 1$ なので、コピーされるときに 1 ずつ 1 つ目の数が足されていきます。したがって次のようになります。

$$\begin{aligned}(0,0)(1,1)[1] &= (0,0)(1,0)[1]\\ (0,0)(1,1)[2] &= (0,0)(1,0)(2,0)[2]\\ (0,0)(1,1)[3] &= (0,0)(1,0)(2,0)(3,0)[3]\end{aligned}$$

ここで、右辺のペアに対応する順序数を列記すると $\{\omega, \omega^\omega, \omega^{\omega^\omega}, ...\}$ となり、ϵ_0 の基本列ができています。そのことから

$$(0,0)(1,1) = \epsilon_0$$

となります。

次に $(0,0)(1,1)(1,0)$ について考えます。アルゴリズム A なので、1 番目の数だけに着目し、1 よりも小さい数の中で右端は $(0,0)$ となり、悪い部分 $(0,0)(1,1)$ がそのままコピーされます。したがって

$$(0,0)(1,1)(1,0)[1] = (0,0)(1,1)[1](\epsilon_0)$$

$$\begin{aligned}(0,0)(1,1)(1,0)[2] &= (0,0)(1,1)(0,0)(1,1)[2](\epsilon_0 \cdot 2)\\ (0,0)(1,1)(1,0)[3] &= (0,0)(1,1)(0,0)(1,1)(0,0)(1,1)[3](\epsilon_0 \cdot 3)\end{aligned}$$

となります。右端のカッコ内は対応する順序数です。これは $\epsilon_0 \cdot \omega$ の基本列なので

$$(0,0)(1,1)(1,0) = \epsilon_0 \cdot \omega$$

となります。次に、$(0,0)(1,1)(1,0)(1,0)[n]$ について $n = 1,2,3$ を計算すると

$$\begin{aligned}(0,0)(1,1)(1,0) &= \epsilon_0 \cdot \omega\\ (0,0)(1,1)(1,0)(0,0)(1,1)(1,0) &= \epsilon_0 \cdot \omega \cdot 2\\ (0,0)(1,1)(1,0)(0,0)(1,1)(1,0)(0,0)(1,1)(1,0) &= \epsilon_0 \cdot \omega \cdot 3\end{aligned}$$

したがって $(0,0)(1,1)(1,0)(1,0) = \epsilon_0 \cdot \omega^2$ となります。このように数列の末尾に $(1,0)$ を加える事で、順序数は ω 倍されます。

数列の末尾に $(1,0)(2,0)$ を加えると、悪い部分の $(1,0)$ が 1 つずつ追加される基本列ができるため、これは順序数を ω^ω 倍することに相当します。したがって

$$(0,0)(1,1)(1,0)(2,0) = \epsilon_0 \cdot \omega^\omega$$

となります。数列の末尾に $(1,0)(2,0)(3,0)$ を加えると、順序数を ω^{ω^ω} 倍することになり、$(1,0)(2,0)(3,0)(4,0)$ を加えると順序数を $\omega^{\omega^{\omega^\omega}}$ 倍することになり、$(1,0)(2,1)$ を加えると順序数を ϵ_0 倍することになります。したがって $(0,0)(1,1)(1,0)(2,1) = \epsilon_0^2$ です。ここに $(2,0)$ をくっつけて $(0,0)(1,1)(1,0)(2,1)(2,0)$ とすると、今度は $(1,0)(2,1)$ が「悪い部分」となってコピーされて ϵ_0 倍を繰り返すことで、

$$(0,0)(1,1)(1,0)(2,1)(2,0) = \epsilon_0^\omega$$

となります。

$(0,0)(1,1)(1,0)(2,1)(2,0)$ の右端にさらにペアを追加することを考えます。$(1,0)(2,1)$ の追加は ϵ_0 倍で、$(1,0)(2,1)(2,0)$ の追加は ϵ_0^ω 倍、$(2,0)$ を追加すると $(1,0)(2,1)(2,0)$ が「悪い部分」としてコピーされて ϵ_0^ω 倍が繰り返され、ϵ^{ω^2} 倍となります。$(3,0)$ を加えると、今度は $(2,0)$ がコピーさ

れますが、$(2,0)$ がコピーされるごとに「悪い部分」も伸びていくことから、$(3,0)$ は ϵ^{ω^ω} 倍に相当します。すなわち

$$(0,0)(1,1)(1,0)(2,1)(2,0)(3,0) = \epsilon_0^{\omega^\omega}$$

です。ここから先は詳細に記述していくときりがありませんが、

$$(0,0)(1,1)(1,0)(2,1)(2,0)(3,0)(4,0) = \epsilon_0^{\omega^{\omega^\omega}}$$

のように ω の指数が増えていきます。$(4,0)$ の段階の複雑性に到達するまでに、$(1,0)$ の複雑性、$(2,0)$ の複雑性、$(3,0)$ の複雑性といったこれまでの複雑性がすべて内包されているためです。そして

$$(0,0)(1,1)(1,0)(2,1)(2,0)(3,1) = \epsilon_0^{\epsilon_0}$$

となります。

そして $(0,0)(1,1)(1,1)[n]$ を $n = 1,2,3,4$ について計算すると

$$\begin{aligned}
(0,0)(1,1) &= \epsilon_0 \\
(0,0)(1,1)(1,0)(2,1) &= \epsilon_0^2 \\
(0,0)(1,1)(1,0)(2,1)(2,0)(3,1) &= \epsilon_0^{\epsilon_0} \\
(0,0)(1,1)(1,0)(2,1)(2,0)(3,1)(3,0)(4,1) &= \epsilon_0^{\epsilon_0^{\epsilon_0}}
\end{aligned}$$

となり、このことから $(0,0)(1,1)(1,1) = \epsilon_1 = \psi(1)$ となります。

バシクは同様に計算を進めて、以下のような結果を示しました。ここでマドールの ψ 関数を使っています。

$$\begin{aligned}
(0,0)(1,1)(2,0) &= \epsilon_\omega = \psi(\omega) \\
(0,0)(1,1)(2,0)(2,0) &= \epsilon_{\omega^2} = \psi(\omega^2) \\
(0,0)(1,1)(2,0)(3,0) &= \epsilon_{\omega^\omega} = \psi(\omega^\omega) \\
(0,0)(1,1)(2,0)(3,1) &= \epsilon_{\epsilon_0} = \psi(\psi(0)) \\
(0,0)(1,1)(2,0)(3,1)(4,0)(5,1) &= \epsilon_{\epsilon_{\epsilon_0}} = \psi(\psi(\psi(0)))
\end{aligned}$$

$$\begin{aligned}(0,0)(1,1)(2,1) &= \zeta_0 = \phi(2,0) = \psi(\Omega)\\ (0,0)(1,1)(2,1)(3,1) &= \Gamma_0 = \phi(1,0,0) = \psi(\Omega^\Omega)\end{aligned}$$

フェファーマン・シュッテの順序数に到達しました。続けます。

$$\begin{aligned}(0,0)(1,1)(2,1)(3,1)(4,0) &= \psi(\Omega^{\Omega^\omega})\\ (0,0)(1,1)(2,1)(3,1)(4,1) &= \psi(\Omega^{\Omega^\Omega})\\ (0,0)(1,1)(2,1)(3,1)(4,1)(4,0) &= \psi(\Omega^{\Omega^{\Omega\omega}})\\ (0,0)(1,1)(2,1)(3,1)(4,1)(4,1) &= \psi(\Omega^{\Omega^{\Omega^2}})\\ (0,0)(1,1)(2,1)(3,1)(4,1)(5,0) &= \psi(\Omega^{\Omega^{\Omega^\omega}})\\ (0,0)(1,1)(2,1)(3,1)(4,1)(5,1) &= \psi(\Omega^{\Omega^{\Omega^\Omega}})\\ (0,0)(1,1)(2,2) &= \psi(\epsilon_{\Omega+1}) = \psi(\psi_1(0))\end{aligned}$$

$(0,0)(1,1)(2,2)$ で、バッハマン・ハワード順序数になりました。

ここで、$(0,0)(1,1)(2,2)[1]$ を $f(n) = n$ として計算したときのペア数列と対応する順序数の変化について、バシクによる計算を示します。式がどんどん横に長くなるので、途中からペア数列を行列表記にします。行列表記では、ペア数列を

$$(a_{11},a_{21})(a_{12},a_{22})...(a_{1k},a_{2k}) = \begin{pmatrix} a_{11} & a_{12} & ... & a_{1k} \\ a_{21} & a_{22} & ... & a_{2k} \end{pmatrix}$$

のように書き換えます。

$$\begin{aligned}(0,0)(1,1)(2,2) &= \psi(\psi_1(0))\\ (0,0)(1,1)(2,1) &= \psi(\Omega)\\ (0,0)(1,1)(2,0)(3,1) &= \psi(\psi(0))\\ (0,0)(1,1)(2,0)(3,0) &= \psi(\omega^\omega)\\ (0,0)(1,1)(2,0)(2,0) &= \psi(\omega^2)\\ (0,0)(1,1)(2,0)(1,1)(2,0) &= \psi(\omega\cdot 2)\\ (0,0)(1,1)(2,0)(1,1)(1,1) &= \psi(\omega+2)\end{aligned}$$

$$\begin{pmatrix} 0 & 1 & 2 & 1 & 1 & 2 & 3 & 2 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix} = \psi(\omega+1)\cdot\psi(\omega+1)$$

$$\begin{pmatrix} 0 & 1 & 2 & 1 & 1 & 2 & 3 & 2 & 3 & 4 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \end{pmatrix} = \psi(\omega+1)\cdot\psi(\omega)\cdot\psi(\omega)$$

$(0,0)(1,1)(2,2)$ の右に $(3,3)$ を加えるとどうなるでしょうか。右に加えるペアを段階的に増やすと

$$\begin{aligned}
(0,0)(1,1)(2,2)(0,0) &= \psi(\psi_1(0))+1 \\
(0,0)(1,1)(2,2)(1,0) &= \psi(\psi_1(0))\omega \\
(0,0)(1,1)(2,2)(2,0) &= \psi(\psi_1(0)\omega) \\
(0,0)(1,1)(2,2)(3,0) &= \psi(\psi_1(\omega)) \\
(0,0)(1,1)(2,2)(3,0)(4,0) &= \psi(\psi_1(\omega^\omega)) \\
(0,0)(1,1)(2,2)(3,0)(4,1) &= \psi(\psi_1(\psi(0))) = \psi(\psi_1(\epsilon_0)) \\
(0,0)(1,1)(2,2)(3,1) &= \psi(\psi_1(\Omega)) \\
(0,0)(1,1)(2,2)(3,2) &= \psi(\psi_1(\Omega_2)) \\
(0,0)(1,1)(2,2)(3,3) &= \psi(\psi_1(\psi_2(0)))
\end{aligned}$$

のように順次複雑性が上がり、ψ_2 関数を使うことになります。さらに $(4,4)$ を加えると

$$(0,0)(1,1)(2,2)(3,3)(4,4) = \psi(\psi_1(\psi_2(\psi_3(0))))$$

と ψ_3 が使われ、同様に

$$(0,0)(1,1)(2,2)(3,3)...(9,9) = \psi(\psi_1(\psi_2(\psi_3(\psi_4(\psi_5(\psi_6(\psi_7(\psi_8(0)))))))))$$

のようになります。したがって $\mathrm{Pair}(n) = (0,0)(1,1)\ldots(n,n)[n]$ とすると

$$\mathrm{Pair}(n) \approx F[\psi_0(\Omega_\omega)](n)$$

となります。このことから、ペア数列数は $F[\psi_0(\Omega_\omega)+1](10)$ 程度の大きさとなります。

7.4.3 サブキュービックグラフ数

サブキュービックグラフ数[14] (subcubic graph number) は、逆数学の提唱者であるフリードマン (Harvey Friedman) が考案した、グラフ理論を使った巨大数です。

グラフ理論 (graph theory) は、頂点（ノード）と辺（エッジ）で構成されたグラフに関する数学の理論です。グラフ理論では、頂点と辺がどのようにつながっているかだけが問題とされ、頂点の位置（空間座標）や辺の長さといった情報は関知しません。グラフがもつ様々な性質を探求するのがグラフ理論です。グラフ理論で扱うグラフには様々な種類がありますが、ここで考えるサブキュービックグラフとは、各頂点が最大 3 個の辺につながっているグラフです。ここでマルチグラフ（両端の頂点が等しい 2 つ以上の辺を持つグラフ）や、接続されていない部分があるグラフ、辺の両端が同じ頂点となる「ループ」を含むグラフもサブキュービックグラフであるとします。

整数 k に対して、サブキュービックグラフの列 $G_1, G_2, .., G_n$ を次の条件を満たすように定めます。

1. 全てのグラフ G_i が $i + k$ 個以下の頂点を持つ。
2. すべての $i < j$ に対して、G_i は G_j に位相同型的埋め込み可能 (homeomorphically embeddable) ではない（G_i は G_j のグラフマイナーではない）。すなわち、G_j から辺を取り除く、孤立した点を取り除く、辺を縮める（辺を取り除くと同時に、辺の両端にある頂点をくっつけて 1 つの頂点とする）、という操作を繰り返すことで G_i とすることができない。

ロバートソン・シーモアの定理[15] (Robertson-Seymour therem) またの名をグラフマイナー定理 (graph minor theorem) によれば、サブキュービックグラフは位相同型的埋め込み可能性において整列擬順序 (well-quasi-ordering) です。 無限に降下する列 (G_{i+1} が常に G_i のグラフマイナー) を

[14] http://ja.googology.wikia.com/wiki/サブキュービックグラフ数
[15] http://mathworld.wolfram.com/Robertson-SeymourTheorem.html

作ることも、無限に比較不可能な列 (antichain) を作ることも、できないということです。そのことから、上記の条件を満たす無限長の $G_1, G_2, ...$ の列を作ることはできないということが分かります（もしできたとすれば、G_i と G_{i+1} が比較不可能なときに G_{i+1} を除く操作を有限回繰り返すことで無限降下列ができてしまう）。したがって、いかなる非負整数 k に対しても、上記の条件を満たす最長の列が存在します。その最長の列の長さを SCG(k) とします。また、有限のサブキュービックグラフが整列擬順序であることは Π^1_1-CA$_0$ では証明不可能であることが示されています。

フリードマンは、「あるチューリングマシン (p.239) が停止するということが、Π^1_1-CA$_0$ 理論で 2^{2000} 個以内の記号で証明可能であれば、そのようないかなるチューリングマシンの停止するまでのステップ数よりも、SCG(13) は大きい」ことを証明しました。この SCG(13) がサブキュービックグラフ数です。SCG(k) は Π^1_1-CA$_0$ で定義されるいかなる再帰関数よりも急増加し、Π^1_1-CA+BI で証明可能なことから、

$$F[\psi_0(\Omega_\omega)](k) \leq \mathrm{SCG}(k) < F[\psi_0(\varepsilon_{\Omega_\omega+1})](k)$$

です。

ここで、$\psi_0(\Omega_\omega)$ は緩増加関数が急増加関数に（自然な基本列を選んだ場合に）追いつく順序数です。

7.4.4　ブーフホルツのヒドラ

ブーフホルツのヒドラ[16] [17] (Buchholz hydra) は、ブーフホルツ (p.187) がカービーとパリスのヒドラゲーム (p.156) を発展させたものです。ここでは、カービーとパリスのヒドラは「カービーのヒドラ」として、ブーフホルツのヒドラを「ヒドラ」とします。ヒドラは次のような性質を持つ ω までのラベルの付いた木（ツリー）A です。

1. ルート（木の根）には + というラベルがついている。

[16] http://ja.googology.wikia.com/wiki/ブーフホルツのヒドラ

[17] Buchholz, W. (1987) An independence result for (Π^1_1-CA)+BI. *Annals of Pure and Applied Logic* 33 131-155. doi:10.1016/0168-0072(87)90078-9

2. ルート以外のノードには、すべて順序数 $v \leq \omega$ のラベルがついている。

3. ルートの 1 つ上のノードには、すべて 0 のラベルがついている。

カービーのヒドラにおけるヘラクレスとヒドラの戦いになぞらえて、同じ用語を使います。ヘラクレスが、あるヒドラ A の首（すなわちトップノード）δ を切り落とすと、ヒドラは任意の自然数 n を選んで、自分自身を新しい形のヒドラ $A(\delta, n)$ に変形します。ここで τ を A における δ からセグメントを 1 つ降りたノードとして、A^- を δ が切り落とされたときに A に残る部分であるとします。$A(\delta, n)$ は、δ のラベルによって次の 3 パターンに分かれます。

1. δ のラベルが 0 のとき。カービーのヒドラと同じようにする。すなわち τ がルートであれば、$A(\delta, n) = A^-$ とする。そうでなければ、図 7.2 のように A^- における τ から上に伸びている部分木の構造を A^-_τ としたときに、A^- において τ からセグメントを 1 つ降りたノードに、n 個の A^-_τ をコピーして増やしたものが $A(\delta, n)$ である。

2. δ のラベルが $u+1$ のとき。δ の下で、$v \leq u$ のラベルを持つ δ から最も近いノードを ϵ とする。ルートの 1 つ上のノードはすべてラベルが 0 なので、ϵ は必ず存在する。図 7.3 のように、ϵ から上の部分木 A_ϵ について、ϵ のラベルを u に変えて、δ のラベルを 0 に変えたものを B として、A の δ を B で取り換えたものを $A(\delta, n)$ とする。ここで $A(\delta, n)$ は n に依存しない。

3. δ のラベルが ω のとき。単純に δ を $n+1$ でラベルし直す。

ブーフホルツは以下の 3 つの定理を証明しました。

定理 1 ヘラクレスは常に一番右の頭を切り落とすことで、有限回数でヒドラに勝つことができる。すなわち、いかなるヒドラ A と自然数の列 $n_0, n_1, ...$ に対しても、必ず $A(n_0)(n_1)...(n_k) = \oplus$ となるような k が存在する。ここで $\oplus$ はルートのみのヒドラである。

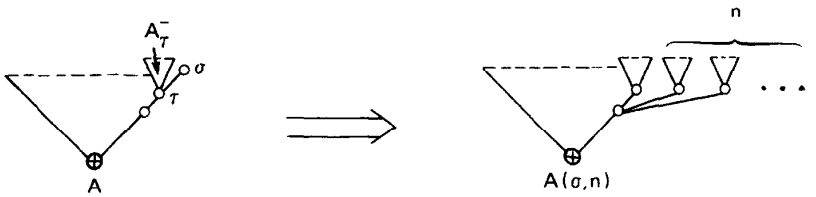

図 7.2: δ のラベルが 0 の時のヒドラの変形。Buchholz (1987) から引用。

定理 2 ヒドラ A の形を定めたときに、そのヒドラと任意の自然数列 $n_0, n_1, ...$ に対して $A(n_0)(n_1)...(n_k) = \oplus$ となるような k が存在するということは、Π^1_1-*CA+BI* で証明可能である。

定理 3 図 *7.4* のように、$+, 0, \underbrace{\omega, \omega, ..., \omega}_{n\text{ 個の}\omega}$ とラベルされた *1* つの枝を持っているヒドラを A^n とする。すべての n に対して、$A^n(1)(2)(3)...(k) = \oplus$ となる k が必ず存在するということは、Π^1_1-*CA+BI* では証明不可能である。

それでは、このヒドラを使ってブーフホルツのヒドラ関数 (Buchholz hydra function) BH(n) を定義します。A^n を定理 3 と同様に定義します。BH(n) を $A^n(1)(2)(3)...(k) = \oplus$ となる最小の k とします。このときに以下が同時に成立します。

1. すべての自然数 n に対して、n の値を固定すれば A^n の形が定まるため、定理 1 と定理 2 により k が存在する。すなわち関数 BH(n) は自然数から自然数への全域関数である。

2. 定理 3 により、関数 BH(n) が自然数から自然数への全域関数であることは Π^1_1-CA+BI では証明不可能である。

この関数の増加率は $F[\psi_0(\varepsilon_{\Omega_\omega+1})](n)$ に匹敵します。つまり Π^1_1-CA+BI の強さをあらわす竹内・フェファーマン・ブーフホルツ順序数の強さとなります。

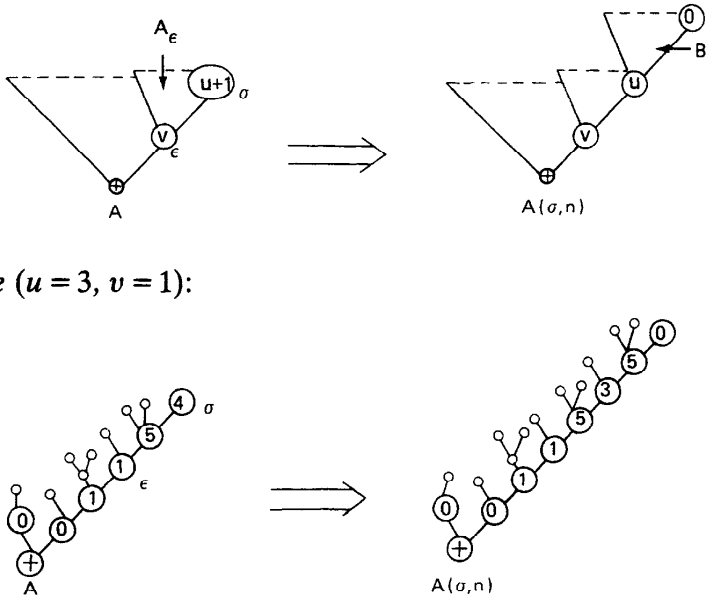

図 7.3: δ のラベルが $u+1$ の時のヒドラの変形。Buchholz (1987) から引用。

また、ω ラベルを除いたときにはペア数列と同じ Π^1_1-CA_0 の強さ、すなわち $F[\psi_0(\Omega_\omega](n)$ になります。原始数列に 2 つ目の自然数を付与したペア数列と、自然数のラベルがついたヒドラは、厳密な計算規則は一致しませんが、各ノードに自然数をラベルすることで証明論的には同等の強め方をしていると解釈することができそうです。

7.4.5 BAN

バードは線形表記 (p.108) から多次元配列表記、超次元配列表記、ネスト配列表記 (p.169) へと発展させて、ϵ_0 のレベルに到達しました。それからさらにそれよりも大きい関数へと発展させて、順序数で評価しています。一連の表記はバードの配列表記 (BAN) です。バッハマン・ハワード順序数レベル、すなわち $F[\psi(\epsilon_{\Omega+1})](n)$ の大きさの H 関数から、さらに強い S

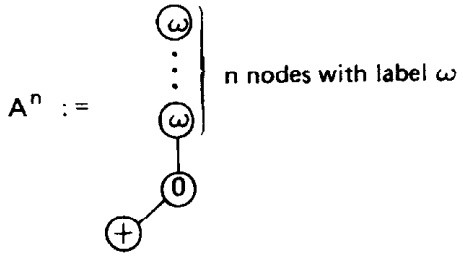

図 7.4: ヒドラ A^n。Buchholz (1987) から引用。

関数、そしてもっと強い U 関数へと強化しました。U 関数は、ブーフホルツの ψ 関数で $F[\psi_0(\Omega_\omega)](n)$ レベルです。U 関数で定義される数が新しいバード数[18] (Bird's number) です。さらに、S 関数は 2014 年に強化され、$F[\psi_0\Omega_\Omega](n)$ のレベルに到達したようです。

7.4.6 BEAF

ジョナサン・バウアーズが開発した BEAF については、これまでに配列表記 (p.108) について解説し、さらに BEAF の基本的な計算ルールと、配列次元演算子を使ったテトレーション配列までの巨大数について解説しました (p.170)。BEAF の特徴は、配列次元演算子で「テトレーション配列」「ペンテーション配列」のような次元そのものを配列にしてしまった、というところにあります。たとえば、バウアーズが名付けたペンテーション配列を使う最小の巨大数トリアクルス (triakulus) は $\{3,3,3\}\&3$ と表記されますが、ここで次元をあらわす $\{3,3,3\}$ を配列次元演算子を使って $3\&3$ と書くことで、配列次元演算子を重ねて $3\&3\&3$ と書くこともできます。

次に、配列表記の定義をレギオン配列 (legion array) によってさらに抽象化します。$n+1$ 番目のレギオン配列はレギオン記号 (legion mark) / によって $\{a,b,c,\cdots/n+1\}$ と書かれ、これを展開すると n 番目のレギオン配列では、配列の区切りが配列次元演算子として展開されます。たとえば、

[18] 巨大数研究 Wiki - バード数 http://ja.googology.wikia.com/wiki/バード数

$\{3, 3/2\} = 3\&3\&3$ のように展開されます。これによって、配列次元演算子を使うことによる強さが、レギオンの段階が高まるごとに入っていきます。

ここから先は、詳しくは BEAF の説明[19]を参照していただくこととして簡単に書きます。**レギオン空間** (legion space) に対する配列次元演算子である**レギオン配列次元演算子** ("legion array of" symbol) && を定義することで、2 重レギオン配列、3 重レギオン配列とします。これをさらに拡張するために、レギオン記号へのレギオン記号とも言える**レギオン記号配列** (legiattic array) を定義し、そこからさらに**レギオン記号配列次元演算子** @ を定義します。

次に**ルギオン空間** (lugion space) を定義して、ルギオン記号、ルギオン記号配列、ルギオン記号配列次元演算子、とレギオンと同じように定義をします。同様にラギオン空間、リギオン空間と高めていきます。ここでレギオン空間を $L1$、ルギオン空間を $L2$、ラギオン空間を $L3$、リギオン空間を $L4$ のように書き、同様に LX 空間を定義できます。さらに L そのものが配列構造を取ります。

レギオン記号配列を使った**ビッグホス** (big hoss) という数は、急増加関数で $\psi(\psi_I(0))$ (オメガ収束点を Deedlit11 の順序数崩壊関数で崩壊させた順序数) のレベルであるという近似が Googology Wiki に以前は掲載されていましたが、その近似は 2015 年 9 月に消されました。この後に紹介する強配列表記を考案した Googology Wiki ユーザーの Hyp cos による解析によれば[20]、BEAF はテトレーション配列よりも上の定義がうまくできていないようです。順序数で $\omega \uparrow\uparrow (\omega+1) = \epsilon_0$ よりも $\omega \uparrow\uparrow \omega+1 = \epsilon_0+1$ の方が強いように、X^^(X+1) よりも X^^X+1 の方が強くなるため、X^(X^^n) という構造は X^X+1 という「あまり強くない構造」に還元されてしまうためだ、としています。

BEAF によって名付けられた最大の数は、BEAF で

$$\{\{L100, 10\}_{10,10}\&L, 10\}_{10,10}$$

[19]巨大数研究 Wiki - BEAF http://ja.googology.wikia.com/wiki/BEAF
[20]http://googology.wikia.com/wiki/User_blog:Hyp_cos/Idea_for_legion

と書かれる数でミーミーミーロッカプーワ・ウンパ[21] (meameamealokka-poowa oompa) という名前です。

7.4.7　バシク行列システム

バシクは原始数列 (p.160) とペア数列 (p.205) を発展させて、トリオ数列、そしてバシク行列システム[22] (Bashicu matrix system) の BASIC プログラムを作成しました。バシク行列は、たとえば

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{pmatrix}$$

のような非負整数を要素に持つ行列で、これを $(a_{11}, a_{21})(a_{12}, a_{22})(a_{13}, a_{23})$ のように列ベクトルの転置行列を並べて表記したものが、バシク行列の数列表記です。バシク行列 BM は、自然数 n から自然数 BM$[n]$ への関数として働き、$(0,0)(1,1)(2,2)(3,3)(3,2)[n]$ のように書きます。

関数 BM$[n]$ が順序数を α のハーディ階層で近似できるとき、バシク行列 BM が順序数 α をあらわしているものとします。つまり次のように表記できます。

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 3 \\ 0 & 1 & 2 & 3 & 2 \end{pmatrix} = (0,0)(1,1)(2,2)(3,3)(3,2) = \psi(\psi_1(\Omega_2))$$

バシクが作成したプログラムは実行を目的としたものではなかったので、計算過程を表示するためのプログラムを本書の著者が作成して、バシク行列計算機[23]を公開しました。そのプログラムはバシクによって動作が検証されました。したがって、バシク行列の定義はバシク行列計算機のソースコード[24]に書かれていることとなります。

バシク行列計算機には Web インターフェイスがあります。プログラムには 4 つの n increment の設定があります。バシク行列のオリジナルの定義は「$n = n * n$」のオプションで、その他のオプションはバリエーションで

[21] http://ja.googology.wikia.com/wiki/ミーミーミーロッカプーワ・ウンパ
[22] http://ja.googology.wikia.com/wiki/バシク行列システム
[23] バシク行列計算機 http://gyafun.jp/ln/basmat.cgi
[24] バシク行列計算機のダウンロード版 https://kyodaisuu.github.io/basmat/

す。Simulate Hardy function のオプションを使って計算すると、その計算は ϵ_0 未満の順序数についてワイナー階層のハーディ関数と完全に一致します。

Googology Wiki のトークページで 2016 年 4 月 28 日に、Hyp cos が計算が終了しない 3 行行列の例があると指摘しました。そこで、バシクは巨大数研究 Wiki の「BASIC 言語による巨大数のまとめ」というブログ記事を更新して、バシク行列バージョン **1** (BM1) からバシク行列バージョン **2** (BM2) へとアップデートしました。BM1 は 2 行行列でも計算が終了しない場合があることがわかりました。2018 年に入ると、多くの人がバシク行列の解析をして、いくつかの別のバージョンが作成されました[25]。

Bubby3 がバージョン 2 の計算が停止しないような 4 行行列を発見して、その指摘を元に 2018 年 9 月 1 日にバシクがバシク行列バージョン **4** (BM4) を定義しました（バージョン 3 はその前にすでに停止しない例が見つけられています)。バシク行列計算機で BM4 の計算ができます。ここで、BM4 は 2018 年 6 月に koteitan が定義した BM2.3 と同じアルゴリズムであることがわかり、koteitan は「上行枝無視方式」と名付けています[26]。本書で紹介したペア数列のプログラムは BM2 を元に作られましたが BM4 でもペア数列のアルゴリズムは同じになります。

バシク行列はあまりにも強力で、解析が困難です。問題は、システムが強力すぎてバシク行列の計算が終了する条件がはっきりとしていないことです。BM4 では n 行行列 $\begin{pmatrix} 0 & 1 \\ 0 & 1 \\ \vdots & \vdots \\ 0 & 1 \end{pmatrix}$ の計算が終了するものと期待されていますが、まだその証明がされていません。そしてその証明が与えられたときに、バシク行列の定義が完成したということになります。計算が終了するのであれば、その増加速度はとても大きいと考えられます。

[25] http://ja.googology.wikia.com/wiki/ユーザー:Kyodaisuu/バシク行列のバージョン
[26] http://ja.googology.wikia.com/wiki/ユーザーブログ:Koteitan/バシク行列の亜種ルールの分類

7.4.8 強配列表記

Googology Wiki ユーザーの Hyp cos は、2013 年に **R** 関数[27](R function) を考案し、改良を続けていましたが、2015 年に R 関数は複雑になりすぎて拡張が難しくなったとして、新しく強配列表記[28][29] (strong array notation) を発表しました。

1. 線形配列表記 (LAN; linear array notation) は、$s(a,b,c,\cdots,z)$ のような表記です。「強配列」であることを示すために s をつけます。多変数アッカーマン関数と似ていますが冪乗を基本としていて、急増加関数の近似は $f(n)=s(\underbrace{n,n,\cdots,n,n}_{n})$ の増加率が ω^ω レベル、すなわち $f(n)\approx F[\omega^\omega](n)$ です。

2. 拡張配列表記 (exAN; extended array notation) は、セパレーター $\{\}$ を導入します。BEAF のテトレーション配列や BAN のネスト配列表記と似ていて、$f(n)=s(n,n\{1\{1\cdots\{1\{1,2\}2\}\cdots2\}2\}2)$ が ϵ_0 レベルの増加率です。

3. 膨張配列表記 (EAN; expanding array notation) は、ドット . をセパレーターとします。$\{1^{\cdot}\}$（右側のセパレーターの左上に上付き文字でドットがついている）は、それ全体がセパレーターと解釈され、それをドットと略記しているものとします。このようなセパレーターの種類そのものが、exAN ルールによって膨張して、$f(n)=\{n,n\{1^{\cdot n}\}2\}$ が ϵ_ω レベルとなります。

   次にグレイヴ・アクセント ‘ (ASCII の文字コードは 96) を導入して、$\{1\text{‘}1\}$ がコンマ、$\{1\text{‘}2\}$ がドット、というようにセパレーターの格が上がることを表記できて、さらに ‘の記号は $\{1^{\text{‘}}\}$ を略記したものであると考えます。 $s(a,b\{1\{1\text{‘}1\text{‘}3^{\text{‘}}\}3\}2)$ といった表記は、$\{1\text{‘}1\text{‘}3^{\text{‘}}\}$ というセパレーターが入った $\{1\{1\text{‘}1\text{‘}3^{\text{‘}}\}3\}$ という表記そのものがセパレーターになっていることになります。計算の順番に関するルールを緻密に定めます。

[27] http://ja.googology.wikia.com/wiki/R 関数シリーズ
[28] http://ja.googology.wikia.com/wiki/強配列表記
[29] Steps Toward Infinity! https://stepstowardinfinity.wordpress.com/array/

$f(n) = s(n, n\{1\{1\{1\cdots\{1\{1,2`\}2`\}\cdots 2`\}2`\}2\}2)$ がバッハマン・ハワード順序数 $\psi_0(\epsilon_{\Omega+1})$ レベルになります。

4. 多重膨張配列表記 (mEAN; multiple expanding array notation) は、グレイヴ・アクセントを連続して並べて書いて n 重グレイヴ・アクセントにします。Hyp cos の定義と計算によれば、n 重グレイヴ・アクセントで $f(n) = s(n, n\{1\underbrace{``\cdots`}_{n\text{ 重}}2\}2)$ という関数は、Π^1_1-CA$_0$ 理論の $\psi_0(\Omega_\omega)$ レベルになります。

5. 第 1 配列落下表記 (pDAN; primary dropping array notation) は、グレイヴ・アクセントがなくなりすべてコンマ , と 2 重コンマ ,, に置き換わります。セパレーターの「レベル」という概念を導入して、レベルを比較しながら計算ルールを定めます。

   $m-1$ 重グレイヴ・アクセントを 持つ $\{A``\cdots`\}$ を $\{A,,m\}$ と書くことで、mEAN の限界である $\psi_0(\Omega_\omega)$ の再帰レベルは $\{1\{1,,1,2\}2\}$ と書くことができてしまいます。$\{1,,2,2\}$ で竹内・フェファーマン・ブーフホルツ順序数の再帰レベルになり、$\{1,,1\{1,,1,,2\}2\}$ で Π^1_1-TR$_0$ 理論のレベル、そこから先はタラノフスキーの C 表記で解析をしています。

   BEAF や BAN を超えていますが、バシク行列は比較不能です。さらにそこから、2 重コンマは $\{1'',\}$ の略記（右側のセパレータの左上にコンマが 2 つついている）で、$\{2'',\}$ は exAN と同様に計算される、といったように高めていきます。

6. 第 2 配列落下表記 (sDAN; secondary dropping array notation) は、グレイヴ・アクセントと 2 重コンマを組み合わせた `,, という記号は、2 重コンマ括弧 $\{\_\_\_\_''\}$ の中で、下のレベルにおける ` と同じように働く、といったような感じで高めていきす。そして、3 重コンマ ,,, を導入します。

7. 配列落下表記 (DAN; dropping array notation) は、pDAN, sDAN の定義を拡張して m 重コンマを導入します。これが R 関数と同程度です。

強配列表記のホームページでは、R 関数よりもさらに強い表記とするために、当初はここから先にさらに NDAN, WDEN, mWDEN, WDmEN, pDDN という 5 つの表記が定義されていました。ところが、NDAN 以降はうまくいかなかったとして 2017 年末に Hyp cos によって取り下げられ、DAN までの 7 つの表記となりました。

7.4.9　ローダー数

ネットニュースの数学グループ (sci.math) と C 言語グループ (comp.lang.c) で、David Meows が 2001 年に巨大数ベイクオフ大会[30] (Bignum Bakeoff contest) を開催しました。この大会は 512 文字以内の大会ルールに定められた方法による C 言語のプログラムで、最大の数を出力するプログラムを作った人が優勝する、というものです。おおまかには以下のようなルールです。

1. main() が返す数が、そのプログラムが出力する数である。有限時間内に main() が終了することが証明できなければ負けである。最大の数を返すプログラムが優勝で、大小の判定できない複数の優勝候補プログラムがある場合にはすべてが優勝となる。

2. ホワイトスペース（空白、タブ文字、復帰、改行、改ページ）は、文字数にカウントしない。

3. C90 すなわち ANSI/ISO 9899-1990 を使う。ただし、メモリは無限にあり、スタックサイズの制限はなく、無限長整数 int を使え、sizeof(int) == 1 である。char, singed char, unsigned char, short int, unsigned short int, unsigned int, long int, unsigned long int も同様で、他の型は使えない。

4. argc, argv, 環境変数を参照できない。

5. ((int)b) < 0 のときに、/ b や % b の演算はできない。

[30] http://djm.cc/dmoews.html

6. 標準ライブラリは、size_t, ptrdiff_t, NULL, offsetof(), jmp_buf, setjmp(), longjmp(), div_t, ldiv_t, calloc(), free(), malloc(), realloc(), bsearch(), qsort(), abs(), div(), labs(), ldiv(), memcpy(), memmove(), strcpy(), strncpy(), strcat(), strncat(), memcmp(), strcmp(), strncmp(), memchr(), strchr(), strcspn(), strpbrk(), strrchr(), strspn(), strstr(), strtok(), memset(), strlen() のみを使える。

この巨大数ベイクオフ大会で優勝したプログラムが、ラルフ・ローダー (Ralph Loader) が作成した **loader.c** というプログラムで、このプログラムによって出力される数が**ローダー数**[31] (Loader's number) です。ローダーは、オックスフォード大学で数学の Ph.D. を取得してから、数年間計算科学の研究者としてラムダ計算等の研究をしていました。2000 年以降はソフトウェアエンジニアとして働いています[32]。

ローダーのプログラムは、フランスのコンピュータ科学者で、現在はスウェーデンのヨーテボリ大学教授のシェリー・コカン (Thierry Coquand) が開発した **CoC** (calculus of constructions) という**型付きラムダ計算** (typed lambda calculus) を対角化します。ここで**ラムダ計算** (lambda calculus) とは、無名関数の抽象表現にラムダ (λ) という記号を用いる形式体系です。たとえば、ある数に 3 を加える関数 f は、通常は $f(x) = x + 3$ と書きますが、ラムダ計算では $\lambda x.x + 3$ という式で表現され、関数の名前である f に相当するものがなくなるので無名関数と言います。ラムダ計算では関数の引数は常に 1 つで、2 つ以上の引数を持つ関数は**カリー化** (currying) という操作によって必ず 1 つの引数を持つ関数に書き換えることができることが知られています。

ラムダ計算に型を導入した型付きラムダ計算では、すべての項 (term) が型 (type) を持っています。型付きラムダ計算で実行できる通常の関数は、項から項への変換規則なので、「項に依存する項」であると考えます。これに「型に依存する項」「型に依存する型」「項に依存する型」を加えたものが CoC です。型付きラムダ理論を、「型に依存する項」「型に依存する型」「項に依存する型」それぞれの有無によって 8 通りに分類して立方体の頂点に

[31] 巨大数研究 Wiki - ローダー数 http://ja.googology.wikia.com/wiki/ローダー数
[32] https://www.linkedin.com/in/ralph-loader-55934025

配したラムダ・キューブ (λ-cube) が描かれることがありますが、CoC はそのすべてを持っているため、ラムダ・キューブに配される型付きラムダ理論の中で、最も表現力が豊かな言語です。

CoC はとても表現力が豊かな言語ですが、どのように書き換えても必ず正規形にたどりついて計算が終了する、つまり強正規化 (strong normalization) するという性質があります。

ちなみに CoC を元にコカンらが作成した証明支援システム Coq は、長い間数学者が証明しようとしてできなかった四色定理 (four color theorem) (証明される前は「四色問題」と言われていた) を証明するために使われた[33] ことで有名になりました。

ローダーのプログラムでは、CoC の文を 2 進数を使って自然数に対応させる規則を作っています。そして $D(n)$ という関数では、n 以下のすべての自然数を CoC の文として解釈して、CoC による推論と正規化をすることにより得られる数を計算し、得られた結果をすべて足します。つまり、CoC の文法を自然数に変換したときに、n 以下となるようなすべての正規化された CoC 式によって得られた数を合計したものが $D(n)$ です。CoC は強正規化するため、この $D(n)$ が計算可能です。

ローダー数は $D^5(99)$ です。$D(99)$ は $2 \uparrow\uparrow 30419$ よりも大きいことから、$D^2(99)$ 以降の計算で $D(n)$ の n に入る数がとても大きくなり、ローダー数の強さは本質的に CoC の証明論的強さに依存することとなります。loader.c は文字数を短くするための処理がされているために読みにくいプログラムですが、ローダーのホームページ[34] には、その元となる読みやすいプログラムがコメント付きで公開されています。そこに書かれているローダーの解説によれば、高階の論理は CoC に翻訳できる（その逆も可能）ので、CoC はすべての「普通の」数学（基本的な解析学、組み合わせ論、等）を形式化するために十分だとしています。そしておおざっぱに考えると、閉じた Σ_1^0 算術の文 $\exists(x)\phi(x)$ を N 文字以内の高階算術の文で証明できると

[33] Gonthier, G. (2008) Formal proof - The four-color theorem. *Notices of the American Mathematical Society* 55 (11):1382–1393. http://www.ams.org/notices/200811/tx081101382p.pdf

[34] http://homepages.ihug.co.nz/~suckfish/

すると、$D(2^N)$ 関数は $\phi(x)$ が成立する最小の x よりも大きい数を返す（証明の CoC への翻訳によって長さ N が変わるので、あくまでもおおざっぱな近似)、とローダーは解説しています。

7.4.10 超越整数と巨大基数による拡張

逆数学の提唱者であるフリードマンは、ある理論体系の証明論的強さに依拠する関数と巨大数を色々と作りました。たとえば、グラフ理論を使った **TREE** 数列は、ACA_0+Π^1_2-BI 理論のいかなる関数も支配して、少なくとも $F[\vartheta(\Omega^\omega\omega)](n)$ 以上です。そしてそれよりも強いサブキュービックグラフ数については、すでに説明しました。ここから先は、フリードマンが考えた関数の中で、特に強力なものをいくつか紹介します。まずは超越整数[35] [36] (transcendental integer) を紹介します。

【定義】超越整数

超越整数 n は、次の条件を満たすような整数すべてである。
チューリングマシン (p.239) M が停止するということを、ZFC で 2^{1000} 個以内の記号で証明できるのであれば、必ず M は n ステップ以内で停止する。

この定義から明らかなように、最小の超越整数が存在します。この定義は ZFC が無矛盾であるときに限り意味があり、それを仮定しています。

ローダー数を計算するチューリングマシンが停止するという証明を ZFC で 2^{1000} 個以内の記号で書くことができれば、超越整数はローダー数よりも大きいことになります。したがって具体的なチューリングマシンと ZFC の証明を書くのは難しいですが、おそらく超越整数はローダー数よりも大きいと考えられます。また ZFC の証明論的順序数は知られていません。2 階算術 Z_2 の証明論的順序数も知られていません。この強さの理論になると、証明論的順序数で比較することは難しく、理論体系の強さそのものを直接比較することになります。

[35] http://ja.googology.wikia.com/wiki/超越整数
[36] https://u.osu.edu/friedman.8/files/2014/01/EnormousInt.12pt.6_1_00-23kmig3.pdf

ここで、超越整数を厳密に定義するためには、ZFC の公理を適用する述語論理の形式体系と、何をもってチューリングマシンの停止性を証明できたとするのかを正確に判定できるような定義を与えることが必要になるだろうと思いますが、そこは省略されています。つまり超越整数は概念を示したものであり、面倒な実装を省略していると考えることができそうです。

フリードマンは超越整数に基づいた関数を定義していないものの、次のように $\mathrm{TR}(T, n)$ を定義することで、理論体系 T の強さに匹敵する計算可能関数ができます。

【定義】超越整数の拡張関数 TR

理論体系 T と整数 n に対し、次の条件を満たす最小の整数を N とするときに、$\mathrm{TR}(T, n) = N$ とする。
チューリングマシン M が停止するということを、T で n 個以内の記号で証明できるのであれば、必ず M は N ステップ以内で停止する。

最小の超越整数は $\mathrm{TR}(\mathrm{ZFC}, 2^{1000})$ です。そして T に ZFC よりも強い公理系を使う理論体系を入れれば、さらに強い関数ができます。理論体系 T の強さを直接的に利用して強い計算可能関数を作るお手軽な方法を示したという点で、超越整数の定義は画期的だと思います。

それでは ZFC よりも強い公理系とは、どのような公理系でしょうか。

ZFC に、ある巨大基数 α が存在するという公理を加えた理論体系は、ZFC よりも強くなります。ここで巨大基数 (large cardinal) のおおまかな定義（その基数が巨大基数であるための必要条件）は、その基数の存在が ZFC と矛盾することが知られておらず、ZFC が無矛盾であると仮定したときに、その基数の存在を ZFC の下では証明できない、つまり「そのような基数は存在しない」という主張が無矛盾であることが証明されている基数です。これまでに紹介した基数の中では、オメガ収束点はまだ巨大基数の仲間入りをしていません。到達不能基数とマーロ基数は巨大基数です。

【定義】理論体系 $Z(\alpha)$

ZFC に「巨大基数 α が存在する」という公理を加えた理論体系を $Z(\alpha)$ とする。

ZFC では α の存在を示すことができないため、$Z(\alpha)$ は、ZFC よりも強くなります。そして巨大基数は**無矛盾性の強さ** (consistency strength) について、厳密な線形順序に従うという経験則があります。このことから、$Z(\alpha)$ の強さは α の無矛盾性の強さによって比較できます。

したがって**超越整数の巨大基数による拡張** $\mathrm{TR}(Z(\alpha), n)$ は、巨大基数 α の強さがそのまま関数の強さとしてあらわれるような計算可能関数となります。本章の最後で紹介する「欲張りクリーク列」による一番強い関数は、$\mathrm{TR}(Z(\alpha), n)$ の α に膨大基数を入れた強さに相当するので、α に膨大基数よりも強い巨大基数を入れれば、「欲張りクリーク列」よりも強い計算可能関数ができます。そこで、α に膨大基数よりも強い**階層内階層基数**[37] (rank-into-rank cardinal) の I0 基数 ρ を入れた **I0 関数** (I0 function) を $\mathrm{I0}(n) = \mathrm{TR}(Z(\rho), n)$ と定義しておきます。

ここまでに出てきた理論体系の強さを、次第に強くなるように並べます。CoC の強さをどこに入れるかは不確定です。

$\mathrm{RCA}_0 < \mathrm{ACA}_0 \approx \mathrm{PA} < \mathrm{ATR}_0 < \Pi^1_1\text{-}\mathrm{CA}_0 < \Pi^1_1\text{-}\mathrm{CA}\text{-}\mathrm{BI} < \Pi^1_1\text{-}\mathrm{TR}_0 < \mathrm{ZFC} < Z(\alpha)$

そして、ここから先は $Z(\alpha)$ の世界に入っていきます。

なお、2 階算術 Z_2 の強さと ZFC を直接比べることはできません。体系が異なるため、Z_2 で証明できて ZFC で証明できないこともあれば、ZFC で証明できて Z_2 で証明できないこともあるためです。ところが、Wikipedia 英語版 “Second-order arithmetic” の記載（タラノフスキーの C 表記を考案したドミトロ・タラノフスキーによる 2012 年 3 月 26 日の編集）によれば、Z_2 に**射影集合の決定性公理** (axiom of projective deteminacy) を加えた Z_2+PD と、ZFC +「n 個の**ウディン基数** (Woodin cardinal) が存在する (n は非負整数)」は同じ証明力の強さとなるようです。

[37] http://ja.googology.wikia.com/wiki/階層内階層基数

7.4.11 有限約束ゲーム

有限約束ゲーム[38] [39] (finite promise games) は、フリードマンが 2009 年に考案した 4 つの関連した 2 人ゲームです。4 つのゲームは、それぞれ「有限区分線形約束ゲーム (FPLCI ゲーム)」「有限多項式約束ゲーム (FPCI ゲーム)」「有限順序不変約束ゲーム (FOICI ゲーム)」「有限線形結合約束ゲーム (FLCI ゲーム)」ですが、ここでは有限区分線形約束ゲーム (finite piecewise linear copy/invert game, 略して FPLCI ゲーム) にしぼって説明します。

関数 $T : \mathbb{N}^k \to \mathbb{N}$ が区分線形 (piecewise linear) であるとは、整数係数の不等式によって条件分けされた、有限個の整数係数の一次関数によって表記できることを意味するものとします。区分線形関数 T に対して、ベクトル $\mathbf{y} \in \mathbb{N}^k$ が x に対する T の逆変換であるとは、$T(\mathbf{y}) = x$ を満たすことを意味するものとします。

区分線形関数 $T : \mathbb{N}^k \to \mathbb{N}$ と、2 つの正の整数 n と s が与えられたときに、有限区分線形約束ゲーム $G(T, n, s)$ を、次のように定義します。

【定義】有限区分線形約束ゲーム

有限区分線形約束ゲーム $G(T, n, s)$ は、マシモとうるかの間の 2 人ゲームである。n ラウンドで終了し、マシモが先手である。

マシモの手番では、$w!$ または $y + z$ となるような x を、$0 \leq x \leq s$ の範囲で選ぶ。ここで y と z はうるかがそれまでに選んだ数でなければならない。x がマシモの提案である。うるかはその提案を受け入れるか拒否するかを選ばなくてはならない。

- 提案を受け入れたときには、うるかは x を選んで、うるかが選んだ数の中からいくつかの数を並べたベクトルは、決して x の T による逆変換と一致することはないと約束する。
- 提案を拒否したときには、x の T による逆変換の中から好きな

[38] http://ja.googology.wikia.com/wiki/有限約束ゲーム
[39] http://cs.nyu.edu/pipermail/fom/2009-September/014007.html

数を選んで、決して x を選ばないと約束する。

うるかが約束を破るとうるかの負けとなる。うるかが n ラウンドすべて約束を破らなければ、うるかの勝ちとなる。ここで、約束はうるかの過去、現在、未来のすべての手番に適用される。

うるかはあらゆる有限区分線形ゲームに対して勝つ戦略があります。そのことは、2 階算術の部分体系である RCA_0 で証明可能です。

さらにこのゲームのルールにおいて、次のルールを追加したゲームを $G_m(T, n, s)$ とします。

うるかは m よりも大きい階乗数の提案を必ず受け入れなければならない。

このとき次の定理が成り立ちます。

うるかはすべての T と n に対して、十分に大きい m と s を設定すれば $G_m(T, n, s)$ に勝つ戦略がある。

このことは SMAH^+ で証明可能ですが、SMAH と同じ強さの部分体系では証明不可能です。ここで SMAH と SMAH^+ は、次のような理論体系です。

- SMAH は ZFC に、すべての正の整数 k に対して「強い k-マーロ基数が存在する」という公理を加えたものである。すべての k に対して加えるので、無限の公理を加えることとなる。
- SMAH^+ は、ZFC に、「すべての正の整数 k に対して強い k-マーロ基数が存在する」という公理を加えたものである。

SMAH と SMAH^+ は一見同じように見えますが、SMAH の証明では有限の公理しか使えないため、SMAH^+ の文を証明するために「すべての k について」という論理を使うことができないという違いがあります。

ここで**マーロ基数** (Mahlo cardinal) は巨大基数です。巨大基数の理論について本書では深く踏み込みませんが、基数 κ が強いマーロ基数であるとは、κ が到達不能で、集合 $\mathrm{U} = \{\lambda < \kappa : \lambda$は到達不能 $\}$ が κ 内で定常集合であることを言います（到達不能と定常の定義は省略します)。さらに、順序数 α に対して α-マーロ基数が定義されますが、その定義は省略します。

FPLCI(a) を、うるかが $G_N(T,n,N)$ に勝つことができる最小の N とします。ただし、以下のすべてが a よりも小さいものとします。

- n
- k (T の定義域は $\mathbb{N}^k$)
- T の区分数
- T の不等式における係数の絶対値
- T の一次関数における係数の絶対値

すると、関数 FPLCI(a) は SMAH の部分体系で証明可能ないかなる再帰関数よりも強い関数となり、より強い SMAH$^+$ で証明可能となります。ZFC よりも強い SMAH を支配する関数なので、FPLCI(a) は小さな a において超越整数を追い越します。また、$G_N(T,n,N)$ の引数である T,n,N の組合せは有限で、それぞれのパターンに対して、うるかの勝敗を計算可能であることから、FPLCI(a) は計算可能関数となり、次章のビジービーバー関数よりは弱い関数となります。

なお、フリードマンが定義した他の 3 つのゲームの中で、2 つのゲームで同様の強さの関数 FPCI(a) と FLCI(a) が定義されます。ゲームの基本構造はそれぞれ同じで、使われる関数の種類が異なり、証明論的な強さは同じで、SMAH を支配して SMAH$^+$ で証明可能な関数となります。

7.4.12　欲張りクリーク列

欲張りクリーク列[40] [41] (greedy clique sequence) は、フリードマンが 2010 年に定義したグラフ理論を使った 3 つの急増加関数 USGCS と USGDCS と USGDCS$_2$ を定義する列です。その定義の前に、関数の強さについて書きます。これらの関数の強さは、次の 4 つの理論で説明されます。

1. SRP 理論は、ZFC に、すべての n に対して「n-stationary Ramsey property を持つラムゼイ基数 (Ramsey cardinal) が存在する」という公理を加えたものである。

[40] http://ja.googology.wikia.com/wiki/欲張りクリーク列
[41] http://www.cs.nyu.edu/pipermail/fom/2010-January/014285.html

2. SRP$^+$ 理論は、ZFC に、「すべての n に対して n-stationary Ramsey property を持つラムゼイ基数が存在する」という公理を加えたものである。

3. HUGE 理論は、ZFC に、すべての n に対して「n-膨大基数 (n-huge cardinal) が存在する」という公理を加えたものである。

4. HUGE$^+$ 理論は、ZFC に、「すべての n に対して n-膨大基数が存在する」という公理を加えたものである。

ここでラムゼイ基数 (Ramsey cardinal) も膨大基数 (huge cardinal) も巨大基数です。これまでに本書で登場した巨大基数の無矛盾性の強さは

$$\text{到達不能} < \text{マーロ} < \text{ラムゼイ} < \text{ウディン} < \text{膨大} < \text{階層内階層}$$

となっています[42]（最後の「基数」は省略)。したがって理論の強さを有限約束ゲームでも含めて比較すると

$$\text{ZFC} < \text{SMAH} < \text{SMAH}^+ < \text{SRP} < \text{SRP}^+ < \text{HUGE} < \text{HUGE}^+$$

となります。

USGCS と USGDCS は、SRP によって証明可能な再帰関数をすべて支配し、SRP$^+$ によって証明可能な再帰関数です。そして USGDCS$_2$ は HUGE によって証明可能な再帰関数をすべて支配し、HUGE$^+$ によって証明可能な再帰関数です。したがって関数の強さは

$$\text{FPLCI} < \text{USGCS}, \text{USGDCS} < \text{USGDCS}_2$$

となります。

それでは欲張りクリーク列の定義を説明します。かなり難解です。まずは用語と記法を定義します。クリーネスター (Kleene star) あるいはクリーネ閉包 (Kleene closure) の記号を使って、$\mathbb{Q}^*$ はすべての有理数のタプル、つまり Q 組 (Q-tuple) の集合を意味するものとします。つまり、$\mathbb{Q}^*$ は、$(4, 13, \frac{2}{3}, -1)$ のような有理数の組をすべて集めた集合です。タプルに 1 か

[42] Cantor's Attic http://cantorsattic.info

らインデックスをつけて下付き文字で記します。たとえば、タプル x は $(x_1, x_2, ..., x_n)$ です。

タプルの連結を角括弧であらわします。たとえば $\langle(0,1),(2,3)\rangle = (0,1,2,3)$ です。

$a \in \mathbb{Q}^*$ に対して、a の上シフト (upper shift) を $\mathrm{us}(a)$ と表記し、a のすべての負ではない要素に 1 を加えたものであると定義します[43]。

$a, b \in \mathbb{Q}^*$ に対して、2 項演算 $\preceq$ と $\prec$ を $a \preceq b \Leftrightarrow \max(a) \leq \max(b)$ および $a \prec b \Leftrightarrow \max(a) < \max(b)$ によって、それぞれ定義します。$a, b \in \mathbb{Q}^*$ が順序等価 (order equivalent) であるとは、a と b が同じ長さであり、かつすべての i と j に対して $a_i < a_j \Leftrightarrow b_i < b_j$ であることと定義します。

集合 $E \subseteq \mathbb{Q}^*$ が順序不変 (order invariant) であるとは、すべての順序等価な x と y に対して $x \in E \Leftrightarrow y \in E$ であることと定義します。H を $\mathbb{Q}^*$ の要素を頂点に持つグラフとします。集合 E を、すべての H の辺 (x, y) に対して、その連結 $\langle x, y\rangle$ が E の要素であるような集合と定義します。このときに、E が順序不変であれば、H は順序不変であると言います。H が順序不変のとき、H は無限の辺を持ちます。

無向グラフ G の部分グラフ A が G のクリーク (clique) であるとは、A が完全グラフである、すなわち、A の任意の 2 頂点間を結ぶ辺が存在するという意味です。有向グラフ G の部分グラフ A が G の下降クリーク (down clique) であるとは、すべての $x \succ y$ を満たす $x, y \in A$ に対して、(x, y) が G の辺であるという意味です（有向グラフなので (x, y) ペアの順番が意味を持ちます)。

上シフト欲張りクリーク列 (USGCS; upper shift greedy clique sequence) を、次のように定義します。

【定義】上シフト欲張りクリーク列

$k \in \mathbb{N}$ と $S \subseteq \mathbb{Q}^*$ が与えられているときに、S の要素を頂点に持つ単純グラフ G と、タプル $x_i \in S$ を要素に持つ空ではないタプル（タプルのタプル） $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ を考える。$\mathbf{x}$ が G の上シフト欲張りクリーク列であるとは、以下のすべての条件が成立することである。

[43] http://www.cs.nyu.edu/pipermail/fom/2009-December/014276.html

1. x_1 は 0 のみである。
2. m を $2 \leq 2m \leq n-1$ を満たす正の整数であるとする。Y を $\langle x_1, x_2, \ldots, x_{2m-1} \rangle$ の連結とし、$y = (Y_m, Y_{m+1}, \ldots, Y_{m+k-1})$ とすると、
   (a) $x_{2m} \preceq y$ かつ $(y, 2m)$ は G の辺ではない。
   (b) $x_{2m+1} = \mathrm{us}(x_{2m})$
3. $\{x_2, x_3, \ldots, x_n\}$ は G のクリークである。

上シフト欲張り下降クリーク列 (USGCDS; upper shift greedy clique down sequence) は、上シフト欲張りクリーク列の定義で、次の箇所を変えたものです。

1. G は有向グラフである。
2. 条件 2(a) は「$x_{2m} = y$」あるいは「$x_{2m} \prec y$ かつ $(y, 2m)$ は G の辺ではない」
3. 条件 3 は $\{x_2, x_3, \ldots, x_n\}$ が G の下降クリークである。

$\mathbf{x} = (x_1, x_2, \ldots, x_n)$ に対するスレッド (thread) は、$(1, 2, \ldots, n)$ の部分列 $(u_1, u_2, \ldots, u_r)$ で、以下のように帰納的に定義されます。

1. $u_1 = 1$
2. u_i が定義されているときに、u_{i+1} を次のように決める。$u_i, u_{i+1}, \ldots, 2^{u_i}$ の中から、$x_j \prec x_{u_i}$ が成立するような j を選ぶ (そのような j が存在しなければ u_{i+1} は定義されない)。そのような j が複数あるときには、その中で $\max(x_j)$ が最大の j を選ぶ。さらにそのような j が複数あるときには、その中で最大の j を選ぶ。そして $u_{i+1} = j$ とする。

スレッド u について $2^{u_r} \leq n$ のときに、そのスレッドは開いたスレッド (open thread あるいは terminal thread) であるとします。

G の上シフト欲張りクリーク列 $\mathbf{x}$ が、開いたスレッド u を持つときに、$\mathbf{x}$ は G の開放上シフト欲張りクリーク列であるとします。そして G の上シフト欲張り下降クリーク列 $\mathbf{x}$ が、開いたスレッド u を持つときに、$\mathbf{x}$ は G の開放上シフト欲張りクリーク下降列であるとします。

いよいよ準備が整い、急増加する関数を定義できます。

【定義】欲張りクリーク関数

1. USGCS(k) は、$S = \mathbb{Q}^k$ としたときに「S の要素を頂点に持つすべての順序不変な単純グラフ G が、最大でも N の長さの開放上シフト欲張りクリーク列を持っている」と言える最小の整数 N である。

2. USGDCS(k) は、$S = \mathbb{Q}^k$ としたときに「S の要素を頂点に持つすべての順序不変な有向グラフ G が、最大でも N の長さの開放上シフト欲張りクリーク下降列を持っている」と言える最小の整数 N である。

3. USGDCS$_2$(k) は、USGDCS(k) の定義において $S = \mathbb{Q}^k \cup \mathbb{Q}^{k+1}$ としたものである。

USGDCS$_2$ 関数は、本書でこれまでに紹介した再帰関数の中で、最も強い関数であると考えられます（超越整数の巨大基数による拡張で示した階層内階層基数の強さを持つ I0 関数を除く）。ただし、バシク行列・強配列表記・ローダー数の D 関数の強さは評価が難しく、これらの中で USGDCS$_2$ 関数よりも強い関数がある可能性はあります。

第 8 章
計算不可能な関数

本章では、いかなる計算可能な関数も支配する計算不可能な関数について扱います。計算や証明をすっぱりとあきらめて、ひたすら大きな数を定義する、という巨大数論の中でも特に不思議な世界に入っていきます。

8.1 ビジービーバー関数

ビジービーバー関数[1] [2] (busy beaver function) あるいはラドのシグマ関数 (Rado's sigma function) は、チューリングマシンから定義される計算不可能関数です。まずはチューリングマシン (Turing machine) について説明します。チューリングマシンは、1936 年にイギリスの数学者、論理学者、コンピュータ科学者のアラン・チューリング (Alan Mathison Turing, 1912–1954) の論文で発表された仮想機械です。無限に長いテープと、その中に格納された情報を読み書きするヘッド、機械の内部状態を記憶するメモリで構成されて、内部状態とヘッドから読み出した情報の組み合わせに応じて、「ヘッド位置のテープに情報を書き込む」「機械の内部状態を変える」「ヘッドを右か左に一つ移動する」のいずれかの動作を、内部状態が停止状態になるまで反復して実行し続けるような機械です。

自然数から自然数への関数 f があって、任意の自然数 n を入力したときに $f(n)$ を出力して停止するチューリングマシンが存在するとき、その関数を計算可能関数 (computable function) と呼び、再帰関数 (recursive

[1] http://ja.googology.wikia.com/wiki/ビジービーバー関数
[2] MathWorld - Busy Beaver http://mathworld.wolfram.com/BusyBeaver.html

function) (p. 71) と同じになります。コンピュータのプログラムで計算できる数は、すべてチューリングマシンで計算できることが知られています。

n 状態 m 記号のチューリングマシンは、次のような機械です。

- マシンは n 個の動作状態と、停止状態を持つ。ここで n は正の整数で、その中の 1 つが初期状態である。
- マシンは 1 つの 2 方向の無限に長いテープを使う。
- テープには m 種類の記号が書かれている。
- マシンには遷移関数があり、入力は (1) マシンの現在の状態と、(2) 現在の位置のテープの状態の 2 つである。
- 遷移関数は、3 つの出力を持つ。(1) テープの現在の位置から読み出された記号を上書きする記号 (2) テープ上を動く方向（左または右）(3) 遷移先の状態（停止する場合もある）

プログラムは入力 2 つと出力 3 つの 5 つ組の情報、つまり（現状態、現記号、次に書く記号、移動方向、次の状態）を有限個並べた表として書くことができます。詳しくは『グッド・マス』[3]という本に、チューリングマシン（チューリング機械）によるプログラムの実例とともに分かりやすく解説されています。

ハンガリーの数学者ティボール・ラド (Tibor Radó, 1895–1965) が、1962 年に n 状態 2 記号のチューリングマシン (n-state, 2-color Turing machine) のビジービーバーゲームを考えて論文で発表しました[4]。

ビジービーバーゲーム (busy beaver game) は、すべて 0 のテープの初期状態から開始して、遷移関数を停止状態になるまで繰り返します。マシンが最終的に停止した場合には、その時のテープ上の 1 の数がマシンが返した数となります。n 状態 2 記号のすべてのチューリングマシンでビジービーバーゲームをしたときに、最大の数を返すものが n 状態のビジービー

[3] Chu-Carroll, M. C. 著 cocoatomo 訳 (2016)『グッド・マス：ギークのための数・論理・計算機科学』オーム社

[4] Rado, T. (1962) On non-computable functions. *Bell System Technical Journal* 41: 877–884. doi:10.1002/j.1538-7305.1962.tb00480.x

バー (n-state busy beaver) であるとされ、最大かどうかは分からないけれど、現在のところは最大となっているものが優勝マシンとされます。問題は、現在優勝マシンとされているものがビジービーバーなのかどうかを、確かめる方法がないことです。それを決定するためには、あらゆる n 状態のチューリングマシンを停止しないことが確定するまで走らせる必要がありますが、どこまで走らせても停止しないことが確定しない（永遠に停止しないのか、いつかは停止するのかが、いつまでも分からない）、という停止性問題 (halting problem) が生じるためです。停止しないということが証明できれば良いのですが、問題はあらゆるチューリングマシンに対して、そのマシンが停止するかどうかを判定するようなアルゴリズムが存在しない、というところにあります。

ビジービーバー関数 $\Sigma(n)$ は、n 状態 2 記号のあらゆるチューリングマシンの中で、最多の 1 を書いて停止するものが書く 1 の個数です。これは、定義はされますが計算不可能で、いかなる再帰的理論でもビジービーバー関数の全域性を証明することができません。前章では「理論体系の証明力の強さ」によって関数の強さを比較しましたが、ビジービーバー関数はあらゆる再帰的な理論体系の証明力の強さを超えてしまいました。

十分に小さい n、すなわち $\Sigma(0) = 0$, $\Sigma(1) = 1$, $\Sigma(2) = 4$, $\Sigma(3) = 6$, $\Sigma(4) = 13$ までは値が分かっていますが、$n > 4$ に対しては、正確な値が計算されていません。$n = 5$ の時は、4098 を返すチューリングマシン（現在の優勝マシン）が見つかりましたが、まだ停止していないチューリングマシンの中で、永遠に停止しないことが証明されていないものが残っているので、それがビジービーバーであるということが確定していません。つまり $\Sigma(5) \geq 4098$ です。$n = 6$ の時は、3.514×10^{18267} という下限が計算されていますが、これもこの値が $\Sigma(6)$ であるかどうかは分かっていません。グラハム数 G との比較については、Googology Wiki ユーザーの Wythagoras が $\Sigma(18) > G$ という結果を示しています。

与えられたプログラミング言語で一定の文字数で書ける最大の数、という関数は、そのプログラミング言語をチューリングマシンに必ず翻訳できることから、本質的にビジービーバー関数と同じです。前章までに定義したすべての計算可能関数によって定義される巨大数は、それを計算するプログラ

ムを書くことができて、チューリングマシンに翻訳可能です。バシク行列のプログラムは BASIC 言語で、ローダー数のプログラムは C 言語で書かれているので、チューリングマシンに翻訳することが可能です。Yedidia and Aeronson (2016) [5] は、フリードマンの欲張りクリーク列を計算する 7910 状態 2 記号のチューリングマシンを構成しました。すなわち ZFC+SRP 理論ではそのチューリングマシンが停止することが証明できないことになります。このことから、$\Sigma(10000)$ は前章までに登場したいかなる巨大数よりも大きいであろうと予想されます。

「n 文字以内で書ける最大の数」あるいは「n 文字以内で書けない最小の数」といったときには、この停止性問題と同じ問題が発生し、計算不可能関数になります。ただし、そのときには「どのような表記法が許されるか」が明確にされていなければなりません。それが明確になされていないと、「15 文字以内で書けない最小の数」が 15 文字で書けてしまう、というような矛盾が生じます。$\Sigma(10000)+1$ を計算するチューリングマシンを 10000 状態以下で作ることができるとすれば、そのチューリングマシンによって $\Sigma(10000)$ よりも大きい数が計算されてしまうことになってしまうので、そのようなチューリングマシンを作ることはできません。そのように考えても、やはりビジービーバー関数が計算できないということが分かります。

Googology Wiki では、ビジービーバー関数を $F[\omega_1^{\mathrm{CK}}](n)$ と急増加関数を使って近似できるとしています。ここで ω_1^{CK} はチャーチ・クリーネ順序数です。たとえば、すべてのチューリングマシンを辞書式順序で並べることで部分再帰関数を $f_0, f_1, f_2, ...$ のように順番に並べることが可能であることを利用すると、チャーチ・クリーネ順序数の基本列を取ることができるためだとしています。本書ではこの考え方を採用して、ビジービーバー関数は $F[\omega_1^{\mathrm{CK}}](n)$ と近似できるものとします。

[5] Yedidia, A. and Aeronson, S. (2016) A relatively small Turing machine whose behavior is independent of set theory. *Complex Systems* 25(4): 297–327. doi:10.25088/ComplexSystems.25.4.297

8.2 神託機械

形式体系 T で m 文字以内の論理式で存在を証明できるすべての整数を計算して、最大の値を返すチューリングマシンを n 個の動作状態で作ることができるとします。そのようなチューリングマシンを実際に作るのは大変ですが、比較的小さな n で大きな m に対して計算ができると期待されます。つまり、形式体系 T で m 文字以内で「存在を証明」できるような数は、計算可能関数の範疇となります。「計算不可能」な関数は、その関数の全域性をいかなる再帰的な理論によっても「証明不可能」ということを意味します。

「計算も全域性の証明もできない全域関数は定義できない」と考えるのであれば、ビジービーバー関数は「定義されていない関数」ということになります。その場合には、本書は前章までしか意味を持たないことになります。本章は、「計算も全域性の証明もできないビジービーバー関数のような全域関数が定義される」ということを信じた人のみ意味を持ちます。これは、どちらが正しいかという問題ではなく、どちらの考え方を採用するか（どちらを信じるか）という問題だろうと思います。

さて、もしあなたがビジービーバー関数が定義できると信じたのであれば、その数が「計算」や「証明」によってではなく神託 (oracle) によってポンと出てくる、と信じたことと同等です。そのような神託を組み込んだ機械が神託機械 (oracle machine) です。神託機械は、計算ができない問題を決定して値を返す「神託」を持っている機械です。

そこで、神託機械を持つ **2** 次チューリングマシン (second order Turing machine) を考えます。これは、通常のチューリングマシンがいつ停止するかを決めることができるようなブラックボックス（神託）を持つ神託機械です。そのような神託を持つ n 状態 2 記号の 2 次チューリングマシンが出力できる最大の数を $\Sigma_2(n)$ として、これが **2** 次ビジービーバー関数 (second order busy beaver function) です。

2 次チューリングマシンは通常のチューリングマシン（1 次のチューリングマシン）の停止性問題を解くことができますが、自分自身がいつ停止するか、という停止性問題を解くことができません。よって、2 次ビジービー

バー関数 $\Sigma_2(n)$ は神託機械に対しても計算不可能な関数となります。

このようにして、一般に x 次のビジービーバー関数 $\Sigma_x(n)$ は $x+1$ 次のチューリングマシンによって計算できますが、x 次以下のチューリングマシンでは計算不可能です。

ビジービーバー関数を急増加関数で $F[\omega_1^{\mathrm{CK}}](n)$ と近似したように、2 次ビジービーバー関数を $F[\omega_2^{\mathrm{CK}}](n)$ で近似できると考えます。ここで ω_2^{CK} は ω_1^{CK} から再帰的に定義できない最小の順序数です。同様に x 次のチューリングマシンを $F[\omega_x^{\mathrm{CK}}](n)$ で近似できると考えます。実際の計算手順を考えて近似したというよりは、そのように考えると便利だからそのような近似の仕方を採用する、ということです。

再帰理論と計算可能性理論では基本的なチューリング次数 (Turing degree) という概念がありますが、その解説は本書の著者には荷が重すぎます。「次数 $0'$ くらいのほとんど全ての次数を支配する関数」[6] といったような面白そうな関数もあります。

8.3　クサイ関数

2011 年国際数学オリンピックの銅メダリストであるアダム・ガウチャー (Adam P. Goucher) が、2013 年にケンブリッジ大学トリニティ・カレッジの学生をしていたときに定義したクサイ関数 (Xi function)[7] について説明します。

まず、次のような SKI 表現を定義します。

- $I(x) = x$
- $K(x, y) = x$
- $S(x, y, z) = x(z, y(z))$

[6] Dobrinen, N. L. and Simpson, S. G. (2004) Almost everywhere domination. *Journal of Symbolic Logic* 69: 914–922.

[7] 巨大数研究 Wiki - クサイ関数 http://ja.googology.wikia.com/wiki/クサイ関数

たとえば $S(K,S,K(I,S))$ を計算してみます。S 式で $x=K, y=S, z=K(I,S)$ を代入すると、$K(K(I,S),S(K(I,S)))$ となります。次に K 式で $x=K(I,S), y=S(K(I,S))$ を代入すると

$$K(K(I,S),S(K(I,S)))=K(I,S)$$

となります。さらに、$K(I,S)=I$ となります。このように

$$S(K,S,K(I,S))$$

という SKI 表現は最終的に I になりましたが、無限に記述が増加するようなものもあれば、計算が終わるものもあります。n 個の文字から始める SKI 表現の中で、最大の有限の文字を返すものが返した文字数をクサイ関数 $\Xi(n)$ とします。SKI 表現はチューリングマシンと同じ強度で、$\Xi(n)$ 関数は計算不可能になります。SKI 計算そのものはチューリングマシンと同じ程度の強さですが、神託記号 (oracle combinator) $\Omega(x,y,z)$ を導入することで強くなります。$\Omega(x,y,z)$ は、x の計算結果が I になるときには y を、そうでなければ z を返します。

8.4 ふぃっしゅ数バージョン 4

神託機械を使って、2002 年にふぃっしゅ数バージョン 4 [8] を定義しました。

【定義】ふぃっしゅ数バージョン 4

1. 関数 f から関数 g への写像 $s'(1)$ を以下で定める。

   関数 f を神託として持つチューリングの神託機械を考え、このマシンによるビジービーバー関数を g とする。すなわち「チューリングマシン＋関数 f」のマシン n ステートでセット可能な 1 の数の最大を $g(n)$ とする。

2. $s'(n)(n>1)$ および $ss'(n)(n>0)$ を、F_3 と同様に定める。すな

[8] http://ja.googology.wikia.com/wiki/ふぃっしゅ数バージョン 4

わち、

$$s'(n)f := g; g(x) = [s'(n-1)^x]f(x)(n > 1)$$
$$ss'(1)f := g; g(x) = s'(x)f(x)$$
$$ss'(n)f := g; g(x) = [ss'(n-1)^x]f(x)(n > 1)$$

3. ふぃっしゅ関数 $F_4(x)$ を以下のように定める。

$$F_4(x) := ss'(2)^{63}f; f(x) = x + 1$$

4. ふぃっしゅ数 $F_4 := F_4^{63}(3)$ とする。

ふぃっしゅ数バージョン 3 の定義では、$s(1)$ はある関数から原始再帰で他の関数へと変換するような写像でした。それに対してふぃっしゅ数バージョン 4 の $s'(1)$ は、ある関数 f からその関数を神託として持つような神託機械を想定して、その神託機械によるビジービーバー関数へと変換するような写像です。

ここで $F_4(x)$ は「ビジービーバー関数を元にして再帰的定義を繰り返す」という関数ではありません。まず初期の関数である $f(x) = x + 1$ に対して $s'(1)$ を施すと、これは神託がないビジービーバー関数そのものなので、$s'(1)f(x)$ がビジービーバー関数に相当します。そしてこの $s'(1)f(x)$ に対して再度 $s'(1)$ を施した $s'(1)^2f(x)$ は、「ビジービーバー関数を神託として持つ神託機械」すなわち 2 次チューリングマシンによって計算できない 2 次ビジービーバー関数 $\Sigma_2(x)$ となります。同様に $s'(1)$ 変換を続けると、次のようになります。

$$\begin{aligned}
s'(1)f(x) &= \Sigma_1(x) \approx F[\omega_1^{\mathrm{CK}}](x)\\
s'(1)^2f(x) &= \Sigma_2(x) \approx F[\omega_2^{\mathrm{CK}}](x)\\
s'(1)^3f(x) &= \Sigma_3(x) \approx F[\omega_3^{\mathrm{CK}}](x)\\
s'(1)^nf(x) &= \Sigma_n(x) \approx F[\omega_n^{\mathrm{CK}}](x)\\
s'(1)^xf(x) &= \Sigma_x(x) \approx F[\omega_\omega^{\mathrm{CK}}](x)
\end{aligned}$$

$s'(n)$ 変換は、$s(n)$ 変換と同じように $s'(n-1)$ 変換を対角化する操作なの

で、次のようになります。

$$\begin{aligned} s'(2)f(x) &= s'(1)^x f(x) \approx F[\omega^{\mathrm{CK}}_{\omega}](x) \\ s'(1)s'(2)f(x) &\approx F[\omega^{\mathrm{CK}}_{\omega+1}](x) \\ s'(2)^2 f(x) &\approx F[\omega^{\mathrm{CK}}_{\omega\times 2}](x) \\ s'(3)f(x) &\approx F[\omega^{\mathrm{CK}}_{\omega^2}](x) \\ s'(4)f(x) &\approx F[\omega^{\mathrm{CK}}_{\omega^3}](x) \\ s'(n)f(x) &\approx F[\omega^{\mathrm{CK}}_{\omega^{n-1}}](x) \\ s'(x)f(x) &\approx F[\omega^{\mathrm{CK}}_{\omega^{\omega}}](x) \end{aligned}$$

ここから先は、ふぃっしゅ数バージョン 3 と同じような構造で

$$\begin{aligned} F_4(x) &\approx F[\omega^{\mathrm{CK}}_{(\omega^{\omega+1})63}](x) \\ F_4 &\approx F[\omega^{\mathrm{CK}}_{(\omega^{\omega+1})63}+1](63) \end{aligned}$$

となります。

このように、ふぃっしゅ数バージョン 4 はチューリングマシンに神託を組み込むことで強化する、という操作を「関数から関数への変換」として強い関数を定義する手法です。「関数を強める」という再帰的な定義をするのではなく「関数を強める変換」を定義するというふぃっしゅ数の考え方から、このような定義が自然とでてきました。

この定義をしたときにはまだふぃっしゅ数バージョン 5 以降ができていなかったのでバージョン 3 を元にした定義となりましたが、ふぃっしゅ数バージョン 5 の $m(2)$ とバージョン 6 の $m(0,2)$ をそれぞれ $s'(1)$ に置き換えることで、それぞれ $F[\omega^{\mathrm{CK}}_{\epsilon_0}+1](63)$ と $F[\omega^{\mathrm{CK}}_{\zeta_0}+1](63)$ 相当の巨大数となります。ふぃっしゅ数バージョン 4 でここまで大きな数を作ったのですから、バージョン 5 以降もそのように大きくしていくのが自然であると考えられますが、なぜまた計算可能関数に戻してしまったのでしょうか。それは当時の巨大数探索スレッドでは「計算不可能関数を使って巨大数を作るのは邪道」という意見が支配的だったためです。結果として、ふぃっしゅ数バージョン 5 を分かりやすい形で定義できたのは良かったと思います。

8.5　ラヨ数

2007 年 1 月 26 日に、MIT（マサチューセッツ工科大学）で巨大数決闘 (big number duel) [9] というイベントが開催されました。これは MIT の哲学者アグスティン・ラヨ (Agustin Rayo) [10] とプリンストン大学の哲学者アダム・エルガ (Adam Elga) [11] の間で、どちらがより大きな有限の数を定義するか、という勝負で、学生に計算理論、順序数、高階言語のような数学と哲学の境界領域に興味を持たせようとして開催されたものです。この勝負はラヨが勝利しました。 この巨大数決闘で勝利したラヨが定義した巨大数がラヨ数[12] (Rayo's number) で、以下のように定義されます。

【定義】ラヨ数

（簡単な定義）一階集合論の言葉でグーゴル個以内の記号で表現できるいかなる有限の数よりも大きな最小の非負整数をラヨ数とする。

（正確な定義）$[\phi]$ と $[\psi]$ を ゲーデル数化された論理式として、s と t を変数設定 (variable assignment) とする。$\mathrm{Sat}([\phi], s)$ を二階述語論理で次のように定義する。

$$
\begin{aligned}
\forall R \quad \{ & \\
& \{\forall [\psi], t : R([\psi], t) \leftrightarrow \\
& (([\psi] = \text{"}x_i \in x_j\text{"} \wedge t(x_1) \in t(x_j)) \\
& \vee([\psi] = \text{"}x_i = x_j\text{"} \wedge t(x_1) = t(x_j)) \\
& \vee([\psi] = \text{"}(\neg\theta)\text{"} \wedge \neg R([\theta], t)) \\
& \vee([\psi] = \text{"}([\theta] \wedge \xi)\text{"} \wedge R([\theta], t) \wedge R([\xi], t)) \\
& \vee([\psi] = \text{"}\exists x_i(\theta)\text{"} \wedge \exists t' \quad : R([\theta], t' \quad)) \\
& (\text{ここで t}' \quad \text{は t の } x_i \text{ を変化させたもの})
\end{aligned}
$$

[9] Big Number Duel http://web.mit.edu/arayo/www/bignums.html
[10] Agustin Rayo http://web.mit.edu/arayo/www/
[11] Adam Elga http://www.princeton.edu/~adame/
[12] ラヨ数 http://ja.googology.wikia.com/wiki/ラヨ数

$$)\} \Rightarrow R([\phi], s)$$
$$\}$$

n 個以下の記号を持ち、自由変数が 1 つ x_1 だけであり、そして次のすべての性質を満たすような式 $\phi(x_1)$ が存在するときに、自然数 m を「n 個の記号でラヨ命名可能である」とする。

(1) $\mathrm{Sat}([\phi(x_1)], s)$ を満たす変数設定 s で $x_1 = m$ が成り立つものが存在する。

(2) $\mathrm{Sat}([\phi(x_1)], t)$ を満たす全ての変数設定 t に対して、$x_1 = m$ が成立する。

$\mathrm{Rayo}(n)$ は n 個の記号でラヨ命名可能ないかなる有限の数よりも大きい最小の非負整数である。$\mathrm{Rayo}(10^{100})$ をラヨ数とする。

「グーゴル」のところを正の整数 n とすれば、急速に増加するラヨ関数 (Rayo function) $\mathrm{Rayo}(n)$ を定義できます。

ラヨ数の正確な定義は二階述語論理 (p.141) の式を使っていて難しいので、説明を書きます。無限に連続するオブジェクト (object) の並びを定義することを変数設定とします。たとえば $(3, 2, 6, 1/2, 4, \pi, \mathrm{Canada}, \omega, 65, \cdots)$ のようなもので、Canada が入っているように、オブジェクトは数である必要はありません。なんでも入ります。次に変数設定に対して適用する論理式を考えます。この論理式は

1. "$a \in b$" a 番目のオブジェクトは b 番目のオブジェクトの要素である
2. "$a = b$" a 番目のオブジェクトは b 番目のオブジェクトと等しい
3. "$(\neg e)$" 式 e の否定
4. "$(e \wedge f)$" 式 e と式 f の論理積 (and)
5. "$\exists a(e)$" 式 e が真となるように a 番目のオブジェクトを変えることができる

以上 5 つにより構成されます。この論理式を記述する言語を、ラヨは一階集合論 (first order set theory) の言語と呼んでいるため、略して **FOST** と呼びます。さて、ある数 m がある式 ϕ によって命名可能であるとは、式 ϕ

を満たすすべての変数設定に対して、最初のオブジェクトが必ず m となること、そしてそのような変数設定が少なくとも 1 つは可能であることを言います。ラヨ関数 Rayo(n) は、n 文字以内の式で命名可能ないかなる有限の正の整数よりも大きな最小の非負整数です。

集合論では、自然数を $0 = \{\}$（空集合）、$1 = \{\{\}\}$、のように考えます。そこで、0 を定義するには $(\neg\exists 2(2 \in 1))$ と定義します。これは「2 つ目の変数が 1 つ目の変数の要素であるように、2 つ目の変数を設定することはできない」ことを意味するので「1 つ目の変数には要素はない」となり、1 つ目の変数は空集合=0 となります。この方法で 1 を定義すると $(((\neg\exists 3(3 \in 2)) \wedge 2 \in 1) \wedge (\neg\exists 3((3 \in 1 \wedge (\neg 3 = 2)))))$ となります。1 を定義するために 38 文字も使ってしまいました。もし、38 文字以内で 1 よりも大きな数を定義する方法がなければ、$\text{Rayo}(38) = 2$ ということになります。これまでの急速増加関数と比べると、あまりにも増加速度が小さいように見えます。しかし文字数が増えれば、再帰的に大きな数を表現できるようになり、爆発的に増加します。

ラヨ数は哲学の教授が考案したものであることから定義には問題がないものと多くの人が考えて、Wikipedia には英語版、日本語版だけでなく 8 ヶ国語版のページができています。榎宮祐「ノーゲーム・ノーライフ」9 巻ではラヨ関数が使われています。しかし、数学者の間では、ラヨ数の定義は不十分なのではないかという意見があります。たとえば p 進大好き bot は以下のような問題点を指摘しています[13]。

1. ラヨ数の定義には「二階述語論理に使われる公理系」と、ラヨ数の定義において式 $\phi(x_1)$ がある性質を「満たすような」自然数の集合を定義するために必要な「一階集合論の公理系」が指定されていない。

2. 1 については、いずれも一階述語論理の ZFC であると解釈可能かもしれない。しかし、一階述語論理の ZFC を使って再定式化したとしても、なお以下の点から定義が不十分である。

3. ラヨ数の定義では、式 $\phi(x_1)$ がある性質を「満たすような」自然数の集合を定めている。ところがこれがどのような「モデル」で満たすと

[13] Googology Wiki: User blog:P 進大好き bot - https://bit.ly/2Q0QhZt

いう意味なのかが定められていないので、定義文になっていない。

4. 3 のモデルがなんらかの集合論のモデルであるとすると、「自然数を構成するための理論の公理系」と「モデルに課される公理系」がいずれも ZFC であるとすれば、不完全性定理からモデルを明示する方法がなくなってしまう。

5. 3 のモデルが定義可能クラスであれば定義可能であるが、その場合は証明可能性でしか真偽が定義できないので、ビジービーバー関数よりもはるかに強いような関数が定義されるわけではない。

6. プラトニズム宇宙 (Platonist universe) を導入したからといって、3 から 5 の問題が即座に解決されるわけではない。

7. 4 の 2 つの公理系において、後者に前者よりも「強い公理」を明示的に採用することで、ラヨ数改良版のような巨大数を定義することは可能である。

本書のここから先はラヨ数を拡張あるいは発展させることでより大きな巨大数を定義しようというものであり、ラヨ数の定義ができていないのであればここから先の巨大数の定義もあやしくなります。特にふぃっしゅ数バージョン 7 の定義はラヨ数の定義に依存しています。

8.6 ふぃっしゅ数バージョン 7

チューリングマシンに神託を組み込んでもラヨ関数にはかなわないとなれば、ラヨ関数をさらに強めるために FOST に神託を組み入れたらどうなるのでしょうか。チューリングマシンに神託を組み込んでビジービーバー関数よりも強いふぃっしゅ数バージョン 4 を作ることができたように、FOST に神託を組み込めばラヨ関数よりも強くなるのではないでしょうか。そのような発想から、2013 年 10 月 19 日に本書『巨大数論』初版の PDF ファイルを公開したときに、FOST に神託を組み入れたふぃっしゅ数バージョン **7** [14] を以下のように定義しました。

[14] http://ja.googology.wikia.com/wiki/ふぃっしゅ数バージョン 7

【定義】ふぃっしゅ数バージョン **7**

関数 f に対して「$f(a) = b$; a 番目のオブジェクトと b 番目のオブジェクトに対して $f(a) = b$ が成り立つ」という神託式 (oracle formula) を FOST の使用可能な式に加えた神託入り FOST (OFOST) のラヨ関数 g を考え、f から g へ変換する写像を RR とする。

ふぃっしゅ数バージョン 6 の定義で $m(0,2) = RR$ とした関数を $F_7(x)$ とする。ふぃっしゅ数バージョン 7、すなわち F_7 を次のように定義する。

$$F_7 := F_7^{63}(10^{100})$$

つまり、OFOST は

1. “$a \in b$” a 番目のオブジェクトは b 番目のオブジェクトの要素である
2. “$a = b$” a 番目のオブジェクトは b 番目のオブジェクトと等しい
3. “$(\neg e)$” 式 e の否定
4. “$(e \wedge f)$” 式 e と式 f の論理積 (and)
5. “$\exists a(e)$” 式 e が真となるように a 番目のオブジェクトを変えることができる
6. “$f(a) = b$” a 番目のオブジェクトと b 番目のオブジェクトに対して $f(a) = b$ が成り立つ

という式によって構成されます。ここで次のようなラヨ階層を考えます。

【定義】ラヨ関数のハーディ関数的拡張

順序数 α に対するラヨ階層を次のように定義する。

$$\begin{aligned} R[0](n) &= n \\ R[\alpha+1](n) &= RR(R[\alpha])(n) \\ R[\alpha](n) &= R[\alpha[n]](n) \quad (\alpha[n] \text{ は極限順序数 } \alpha \text{ の基本列の } n \text{ 番目}) \end{aligned}$$

まずは $R[1](n)$ について考えます。FOST に $R[0](n) = n$ という関数 f を

加えたもので、これは神託を必要とせずに決定できる式です。FOST の強さには影響を与えないため、そのラヨ関数 g は通常のラヨ関数と同じ強さになり、$R[1](n)$ はラヨ関数と等しくなります。

次に $R[2](n)$ は $R[1](n)$ という関数 f を神託式として持つ強化版 FOST のラヨ関数です。この強化版 FOST では、$f(a) = b$ という式で $\mathrm{Rayo}(a)$ を評価できてしまいます。したがって、ラヨ関数およびラヨ関数よりも増加速度が小さい関数（計算可能関数、ビジービーバー関数、クサイ関数）を使って定義されるいかなる関数よりも増加速度が大きいということになります。さらに、$\mathrm{Rayo}^{\mathrm{Rayo}(n)}(n)$ といったようなラヨ関数を使って再帰的に定義できるような関数も、ラヨ関数が組み入れられた強化版 FOST で式を構成することができるため、その強化版 FOST におけるラヨ関数に相当する $R[2](n)$ は、ラヨ関数を使って再帰的に定義できるいかなる関数も支配する関数となります。

そして $R[3](n)$ はこの $R[2](n)$ を神託式として組み込んだ「強化版をさらに強化した FOST」のラヨ関数で、$R[2](n)$ から再帰的に定義できるいかなる関数も支配する関数となります。

このように、n 次チューリングマシンを考えることができたのと同じような構造で、n 次 FOST と n 次ラヨ関数を考えることができます。

ふぃっしゅ数バージョン 7 はふぃっしゅ数バージョン 6 の定義で $m(0,2) = RR$ としたものなので、

$$\begin{aligned} m(0,2)m(0,1)(x) &= RR[m(0,1)](x) \approx R[1](x) \\ m(0,2)^2m(0,1)(x) &\approx RR[R[1]](x) = R[2](x) \\ m(0,2)^3m(0,1)(x) &\approx RR[R[2]](x) = R[3](x) \\ m(0,3)m(0,2)m(0,1)(x) &\approx R[\omega](x) \end{aligned}$$

のようになります。まず $m(0,2)m(0,1)(x)$ は $m(0,1)$ という再帰関数を写像 RR によってラヨ関数程度の大きさに変えます。これが $R[1](x)$ に相当します。そこから先は、ラヨ階層の 2 番目の定義式から $RR[R[1]](x) = R[2](x)$ のように n 次ラヨ関数の n が 1 ずつ増えていきます。さらに $m(0,3)m(0,2)m(0,1)(x)$ はその操作を対角化することで、$R[\omega](x)$ の強さとなります。

このように、ふぃっしゅ数バージョン 6 における急増加関数の順序数と、ふぃっしゅ数バージョン 7 におけるラヨ階層の順序数が対応します。ふぃっ

しゅ数バージョン 6 では、順序数 $\zeta_0 = \phi(2,0)$ にまで到達したので、ふぃっしゅ数バージョン 7 の大きさは $R[\zeta_0]^{63}(10^{100})$ 程度となります。

ふぃっしゅ数バージョン 7 を Googology Wiki に紹介したところ、当初はこの定義はうまくいかないとされました[15] 。その理由は「神託式を組み込むことで FOST を強めることはできない、なぜならば神託式として組み込む $f(a) = g$ という関数は FOST で書けるので、字数の短縮にしかなっていない」というものです。それに対して著者は次のように反論しました。「FOST でラヨ関数そのものを記述することはできない。もし記述することができたら、ラヨ数そのものをグーゴル以下の文字数で記述できてしまい、矛盾する」。この議論から、そもそも FOST の定義そのものが矛盾なのではないか、という議論も出ましたが、ラヨ数の定義では 1 つの数を指定しない式は除外されるので矛盾が生じるような式は除外される、という解釈をしました。いずれにしても、FOST 自身のラヨ関数を有効に FOST の中に組み込むことはできない、ということになります。

Googology Wiki では、FOST を強くする方法として神託式 $f(a) = b$ を使うというのはあまりスマートではないので、もっと良い方法もあるのではないか、というような話が進行しました。たとえばある式が真であるという判定をする真理述語 (truth predicate) を入れれば、ラヨ関数を FOST で記述してその式がある値になる、という式の真偽を判定することができるだろう、というような話になります。ただしタルスキの定理 (Tarski's undefinability theorem) によって真偽を判定するような式を定義することはできない、という問題が生じます。実際に、後に出てくるリトルビッゲドンという巨大数は、真理述語を使っています。

いずれにしても、このふぃっしゅ数バージョン 7 を提起したことで、それまでは「ラヨ数よりも本質的に大きな巨大数を考えるのは無理である」という Googology Wiki における雰囲気が崩れて、ラヨ数を超える巨大数を作ろう、という挑戦がされることとなりました。ふぃっしゅ数バージョン 7 は、ビッグフットから先の巨大数が生まれるきっかけとなったと言えます。

[15] http://googology.wikia.com/wiki/User_blog:Kyodaisuu/English_description_of_Fish_numbers

8.7 ビッグフット

Googology Wiki ユーザーの LittlePeng9 またの名を Wojowu （プロフィールによると居住地はポーランド）が、2014 年に考案したビッグフット (BIG FOOT) [16] [17] という巨大数について説明します。

【定義】ビッグフット

ウードル理論 FOOT によって n 文字以内で一意に定義される最大の自然数を FOOT(n) とする。ビッグフットは $\mathrm{FOOT}^{10}(10^{100})$ である。

このビッグフットは、ふぃっしゅ数バージョン 7 よりも大きい数です。ビッグフットの定義を見ると、ラヨ数の定義とよく似ていますが、ラヨ数では FOST を使っていたところが、Wojowu が考案したウードル理論 (oodle theory) に変わっています。ウードル理論は、first-order oodle theory とも呼んで、**FOOT** と略します。一次でもなんでもないのですが、FOOT という略語を作りたかっただけとのことです。

ここで BIG FOOT という名前はサイビアンが提案したものが採用されました。ビッグフット (bigfoot) はアメリカとカナダのロッキー山脈一帯で目撃される未確認動物です。サイビアンが考案した巨大数の名前には、神 (god)、怪物 (monster)、巨人 (giant)、古代エジプトの神トート (Thoth) 、宇宙生物クトゥルフ (Cthulhu) などが多く使われています。FOOT からビッグフットを連想するのは、サイビアンならではです。

それでは、FOOT について Wojowu のオリジナルの説明と同じように説明をします。FOST の議論領域であるフォン・ノイマン宇宙 V の集まりをクラスとして、クラスに対する変数を導入することで二階集合論 (SOST; second order set theory) を作ることができます。そしてクラスの上にさらに「スーパークラス」を加えることでさらに高次の理論を作り、というように高めていくことはできるでしょうか。そのような発想で Wojowu が考えたのが FOOT です。FOOT の言語は可算無限の変数とウードル結合子（非論理記号）$\in$ と $=$、そしていくつかの標準的な論理記号、たとえば $\exists, \wedge, \neg$ を持っていま

[16] http://ja.googology.wikia.com/wiki/ビッグフット
[17] First-order oodle theory http://snappizz.com/foot

す。そして、ある変数設定における論理式 $x \in y, x = y, \phi \wedge \psi, \neg\phi, \exists x : \phi(x)$ の定義を、標準的な方法で定めています。

ウードルを整礎とするために、ウードルの階数 (rank) を次のように定義します。

- A が空のウードルであれば、A の階数は自分自身である。
- A が階数 B の要素を含み、A のすべての要素の階数が B であるか B の要素であるときに、A の階数は $B \cup \{B\}$ である。
- A が上記のような要素を持たないときには、A の階数はすべての要素の階数の和集合である。

$A \in B$ のときに階数 A が階数 B よりも「小さい」とします。そしてウードルの階数をウードル順序数 (oodinal) とします。$\alpha \cup \{\alpha\} = \alpha + 1$ と表記し、そのようなウードル順序数を後続ウードル順序数、後続でないウードル順序数を極限ウードル順序数とします。すべてのウードルは必ず階数を持ちます。「階数が α よりも小さいすべてのウードル」を V_α とすると、ウードル階層 (oodle hierarchy) を次のように定義できます。

- $V_\varnothing = \varnothing$
- $V_{\alpha+1} = \mathcal{P}(V_\alpha)$
- $V_\alpha = \bigcup_{\beta<\alpha} V_\beta$

$V = \bigcup V_\alpha$ をウードル宇宙 (oodleverse) とします。ウードル宇宙はすべてのウードルを含みますが、ウードル宇宙自身はウードルではありません。

ここで、FOOT 言語から定義できないほど大きなウードル順序数 Ord をある方法によって定義します（定義は省略します)。そして Ord よりも階数が小さいウードルが「集合」で、Ord の要素が「順序数」であると定義します。この Ord そのものは、FOOT 言語から定義することはできません。そこで、FOOT に Ord という定数記号を加えることで FOOT を拡張します。すると、Ord 記号がない FOOT 言語で扱われている集合は Ord よりも階数が小さいウードル、すなわち V_{Ord} 内の集合となります。そして

そのような集合の集まりは $V_{\mathrm{Ord}+1}$ 内のウードルであることから、集合論における真理述語を使うことができます。そのことから FOST におけるラヨ関数を定義できて、二階集合論 (SOST) と同じようなことを記述できます（このことは、後述するように 2016 年末に否定されます)。同様の拡張を $\mathrm{Ord}+\omega$, $\mathrm{Ord2} = \mathrm{Ord} + \mathrm{Ord}$, Ord^2, $\mathrm{Ord}^{\mathrm{Ord}}$, $\Gamma_{\mathrm{Ord}}+1$, $\omega^{\mathrm{CK}}_{\mathrm{Ord}+1}$, ... と続けることができます。

次に、Ord という記号が入った FOOT で定義可能ないかなる順序数よりも大きな最小のウードル順序数を Ord_2 とします。同様に Ord_3, Ord_4, ... と記号を追加します。次に角かっこ [ ] を導入して、変数 x がウードル α の値をとるときに、$[x]$ はウードル順序数 Ord_α を意味するものとします。そして Ord_α を「$\beta < \alpha$ を満たす β に対して定数記号 Ord_β を使うことができるような、Ord_α よりも階数が低いパラメータを含む FOOT 言語で定義できるいかなる順序数よりも大きい最小のウードル順序数」であると定義します。これが FOOT です。そして、この FOOT からビッグフットが定義されます。

FOOT の強さについては、2016 年末に再評価がされました。まず、12 月 25 日に Googology Wiki ユーザーの Emlightened が「ビッグフットはふぃっしゅ数バージョン 7 よりも小さい」というブログを書きました。それを受けて、同日にビッグフット考案者の Wojowu が「FOOT は私が考えていたほど大きくなかった」というブログを書きました。そのブログによれば、FOOT は FOST に真理述語を加えた言語と同程度の強さであり、SOST ほどの強さはない（すなわち、上記の説明の中で SOST との比較については誤りであった)、とのことです。一方で FOOT でふぃっしゅ数バージョン 7 の RR を記述できるため、ビッグフットはふぃっしゅ数バージョン 7 よりは大きい、としています。

8.8 サスクワッチ

ビッグフットが「それほどの大きさではない」ことを指摘した Emlightened が、2017 年 1 月 5 日に Googology Wiki のブログでビッグフットよりも大

きいリトルビッゲドン[18] (Little Bigeddon) という巨大数を定義しました。Emlightened は ϵmli という名前で署名しているので、ここではエミリと呼びます。プロフィールによれば、エミリは女性です。まず集合論の言語に階数変数と真理述語 T を導入した言語 $\mathcal{L} = \{\in, T\}$ を定義しました。その定義は本書では割愛するので、脚注の出典を参照してください。ここで真理述語は階数変数によって階層化されています。そして、リトルビッゲドンを次のように定義しました。

【定義】リトルビッゲドン

リトルビッゲドンは、以下の条件を満たす最大の k である。
言語 $\mathcal{L} = \{\in, T\}$ で $\exists! a(\varphi(a)) \wedge \varphi(k)$ すなわち「$\varphi(a)$ が真であるような a がただ 1 つ存在し、$\varphi(k)$ が真である」が真となるような、階数が $12 \uparrow\uparrow 12$ 以下の 1 変数式 φ が存在する。

ここで φ の文字数を制限しなくても、階数が $12 \uparrow\uparrow 12$ 以下で自由変数が 1 個の式は、等価な式を 1 つと数えると有限個となるように、階数が定義されています。

エミリはさらに、2017 年 3 月 27 日にサスクワッチ[19] (Sasquatch) またの名をビッグビッゲドン (Big Bigeddon) を定義しました。サスクワッチは未確認動物ビッグフットの別名です。等号が定義されている言語 $(\in, \bar{\in}, <)$ の 2 項述語 $\in, \bar{\in}, <$ と、単項関数 F と R を定義して（定義は出典を参照）、ビッグフットのように「宇宙を拡張する」ような能力を持たせました。そして、サスクワッチを以下のように定義しました。

【定義】サスクワッチ

言語 $\{\bar{\in}, Q\}$ (ここで $Q(a,b) \leftrightarrow R(a) = b$) で $\exists! a(\phi(a)) \wedge \phi(k)$ を満たす、階数が $12 \uparrow\uparrow 12$ 以下の 1 変数式 ϕ が存在するような最大の k をサスクワッチ（ビッグビッゲドン）とする。

これが現在最強の巨大数と言えるでしょう。

[18] http://ja.googology.wikia.com/wiki/リトルビッゲドン
[19] http://ja.googology.wikia.com/wiki/サスクワッチ

おわりに

ここでは、本書の著者、つまり「私」と巨大数の関わりについて記します。本書を読んでいる途中の息抜きトピックをここにまとめておくという趣旨です。

私は子供の頃から大きな数に対する関心を持っていて、本文中にも記したように、小学生の時には無量大数までの数の単位を覚えて喜んでいました。とはいえ、無量大数よりも大きな数については、グーゴルについて知っていた程度でした。大学では数学を専門的に学んだわけではないので、本書に書いたような再帰理論や順序数、数学基礎論、数理論理学などについては、何も知識がない状態でした。

その私が、巨大数について探求しようとしたきっかけは、2 ちゃんねる掲示板というインターネットの掲示板でのことです。2 ちゃんねる掲示板では、話題のジャンルごとに掲示板群「板」が設定され、板の中で話題ごとに立てられる「スレッド」において、会話がなされます。数学の話題を扱う「数学板」の中で、2002 年 6 月 17 日に「一番でかい数出した奴が優勝」というスレッドが立ちました。このスレッドでグラハム数について言及されていたことで、グラハム数の存在を知って、その大きさを想像して驚きました。そこで、グラハム数を超える数をグラハム数とは別の仕組みで定義して、そのスレッドに書き込みました。掲示板の一時的なお遊びだと考えていたので、「ふぃっしゅっしゅ」というふざけた名前で、その数の名前を「ふぃっしゅ数」としました。当時は、巨大数を生み出す関数としては矢印表記とアッカーマン関数だけしか知識がなかったので、その知識を元に作成したのがふぃっしゅ数バージョン 1 です。とにかく複雑にして大きな数を作ろうと考えたので、今から振り返ると無駄に難解な定義となっています。

当初はとても軽いノリで考えていましたが、その後「ふぃっしゅ数はどうやって計算するのか」といったことで、スレッドが盛り上がりました。難解だったことが、むしろそういった盛り上がりを誘発した可能性があります。特に、695 さん（のちに巨大数漫画『寿司　虚空編』を描くことになる小林銅蟲さん）が計算過程を丁寧に検証し、さらにスレッドにはチェーン表記が紹介され、ふぃっしゅ数とチェーン表記を比較するとどうなるか、といったようなことで、スレッドが盛り上がりました。また、チェーン表記を回転させるバード数（旧バード数）のサイトが見つけられ、ふぃっしゅ数とバード数ではどちらが大きいか、というようなことでさらにスレッドが盛り上がりました。そして、この「巨大数探索スレッド」で議論を重ねながら、ふぃっしゅ数の新しいバージョンを考えていきました。2003 年に考案した「ふぃっしゅ数バージョン 5」は自信作となりました。巨大数探索スレッドでは、順序数やハーディ階層、再帰理論など様々な理論が議論されましたが、当時の私にはまだ十分に理解できていないものがほとんどでした。なお、巨大数探索スレッドは本書を執筆中の今も継続中で、2017 年 1 月 20 日には「巨大数探索スレッド 12」が開始しています。

私の巨大数に関する関心が 2003 年以降で再度高まったのは、2007 年です。巨大数探索スレッドでは、たろうさんが多変数アッカーマン関数の定義、急増加関数による評価、ふぃっしゅ数バージョン 5 の急増加関数による評価、その他もろもろを書き込みました。すぐには理解できませんでしたが、掲示板でたろうさんに色々と聞きながら、ようやく理解ができました。そして、11 月 3 日に、東京都の文京区シビックセンターで小林銅蟲さん主催による巨大数勉強会が開かれました（小林銅蟲さんは「もやしっ子」の名前で書き込みをされていましたが、このときに漫画「ねぎ姉さん」の作者であると明かされました）。その勉強会には私も参加し、たろうさんによる順序数の解説や、ポチさんによるヒドラとふぃっしゅ数バージョン 5 の解説があり、私の中で理解が一気に進みました。ちなみに、ポチさんはそのときに「ブーフホルツのヒドラというものもある」と紹介をされましたが、その意味を私が理解するのは何年も後になります。

巨大数探索スレッドでは、このように面白い議論がされていましたが、はじめて巨大数に興味を持った人がスレッドを読んで理解をするのは大変な

ので、その内容をまとめておきたいと思い、2007 年 9 月 24 日に『巨大数論(まだ書きかけ)』という PDF ファイルを作成してアップしました。2007 年 10 月 27 日には、ふぃっしゅ数バージョン 6 の定義と計算を PDF ファイルに加えました。当時の書きかけ巨大数論では、グラハム数とふぃっしゅ数バージョン 1 から話が始まり、巨大数探索スレッドで議論がされていた話題を中心にまとめていました。なお、本書のホームページに過去の版がアーカイブされています。その書きかけバージョンは、2009 年 3 月 2 日に一度微修正されたまま、放置されていました。

数年間放置していた巨大数論の執筆を再開しようと思ったきっかけは、2013 年に小林銅蟲さんの巨大数漫画『寿司　虚空編』が描かれているのを知ったときです。そうです、たしかに 2007 年の巨大数勉強会では「ぜひ、巨大数の漫画を描いてください」と言ったような気はしますが、まさか本当に連載が開始して話題になるとはびっくりです。話の内容も、第 1 話がグラハム数で第 2 話がふぃっしゅ数と、当時の書きかけバージョン『巨大数論』と同じです。これは、巨大数論をきちんと書かなければ、と思い、10 月 19 日に初版を発行しました。まずは、いきなりグラハム数の話をしても普通は大きさがぴんと来ないので、グラハム数に到達するまでの話を詳しく記述しました。そして、グッドスタイン数列やヒドラゲームの説明をしました。

その執筆をしているときに、巨大数に関する情報をネットで見てまわっていたところ、Googology Wiki が設立されていることを知り、そこに書かれているクサイ関数やラヨ数についてとりあえずわかった範囲で書きました。そのときに、ラヨ数に神託を組み込むことを考えて、ふぃっしゅ数バージョン 7 を定義しました。

Googology Wiki ではアカウント Kyodaisuu を取得し、英語で簡単にふぃっしゅ数の説明をしました。すると、海外のグーゴロジストたちの関心はふぃっしゅ数バージョン 7 がラヨ数を超えるしくみに集まりました。その議論がきっかけとなり、やがてビッグフットという巨大数が考案され、さらにリトルビッゲドンへと発展するときにもふぃっしゅ数バージョン 7 との比較が良い議論のきっかけとなりました。2013 年末には、Googology Wiki の日本語版「巨大数研究 Wiki」が開設され、そこで情報を整理しな

がら、少しずつ海外における巨大数論の発展の様子を理解してきました。時々、小林銅蟲さん主催の巨大数勉強会に参加しています。

そして、『寿司　虚空編』がきっかけとなって、2014 年以降、関西すうがく徒のつどい、ニコニコ超会議、サイエンス・アゴラ、ロマンティック数学ナイトなど、巨大数論に関して発表される機会が増えて、巨大数論に対する関心が高まりました。詳しくは、巨大数研究 Wiki の Googol ニュースを参照してください。

巨大数論の第 2 版を発行しようと思ったのは、初版では主に日本の巨大数論、すなわち巨大数探索スレッドで議論されてきた内容が中心であったのに対して、海外における巨大数論の体系が見えてきたことで、それを形にしておきたいと思ったためです。また、巨大数探索スレッドに書き込まれた原始数列やペア数列をはじめとするバシク行列システムについても、記述しておきたいと思いました。このようにして本書は、英語圏における巨大数論の流れだけではなく、日本における巨大数論の流れ、その中でも特殊かつ複雑な私が考案した「ふぃっしゅ数」の話が組み込まれているため、まとまりがないものとなっています。内容が理解できてくれば、それぞれの関連性が見えてきます。

このように、私がふぃっしゅ数を考案して巨大数を考え始めてから 15 年かけて少しずつ理解してきた内容や考えてきたことを、その都度書きたいように書きためてきたものなので、いきなり本書の内容をすべて理解するのは難しいかもしれません。特に、後半の内容は専門的に高度な内容が含まれていて、私自身にとっても難しいです。すぐに読んでわからないところは、またしばらくして読み直すと、わかるときがあるかもしれません。式を目で追うだけでなく、紙と鉛筆を使って自分の手でいろいろと計算をしてみることで、わかったという人も多いです。

なにか新しいことが「わかる」ということは、とても楽しいものです。私自身が、巨大数論と関わる中で何度も「なるほど」と思うときがあったように、本書を読んだみなさまが、何か 1 つでも「なるほど」とか「これは面白い」と感じていただけることがあれば、本書を世に出した甲斐があったと考えます。

巨大数年表

日付	出来事	ページ
287 - 212 BC 頃	アルキメデスが『砂粒を数えるもの』で $10^{8 \cdot 10^{16}}$ まで定義	p.28
1〜7 世紀	華厳経に不可説不可説転の記載	p.31
1299 年	朱世傑『算学啓蒙』で極以上の命数法が初めて登場	p.15
1484 年	シュケがヨーロッパ系言語ではじめて「〜リオン」という接尾語で数の名称を体系的かつ非常に大きな桁まで命名	p.18
1631 年	吉田光由の『塵劫記』で無量大数までの数の体系ができる	p.13
1904 年	ハーディ階層の定義	p.126
1908 年	ヴェブレン関数の定義	p.178
1920 年	グーゴルの命名	p.19
1928 年	アッカーマン関数の発表	p.65
1933 年	第 1 スキューズ数が論文に登場	p.39
1938 年	エディントンが宇宙に存在する全陽子の数が 136×2^{256} 個であると議論	p.18
1944 年	グッドスタイン数列の定義	p.154
1947 年	グッドスタインがテトレーション、ペンテーション、ヘキセーションの命名	p.49
1950 年	シャノン数の計算	p.21

日付	事項	ページ
1950 年	スタインハウスの多角形表記の発表（初出不明）	p.72
1953 年	グジェゴルチクとリトルウッドが、それぞれ急増加関数の元となる関数を定義	p.131
1955 年	第 2 スキューズ数が論文に登場	p.39
1957 年	竹内外史の順序数図形	p.186
1962 年	ラドのビジービーバー関数	p.239
1971 年	グラハムが、今では小グラハム数と呼ばれる数を論文で使用	p. 75
1975 年	フリードマンが逆数学を提唱	p.194
1976 年	クヌースが矢印表記を考案	p.108
1977 年	ガードナーが現在グラハム数として知られる数を紹介	p. 75
1980 年	グラハム数がギネス世界記録で「数学の証明で使われた最大の数」として掲載	p.75
1982 年	カービーらのヒドラゲーム	p.156
1987 年	ブーフホルツのヒドラゲーム	p.216
1994 年	リンデの確率過程的インフレーション宇宙のポアンカレ時間が $10^{10^{10^{10^{10^{1.1}}}}}$ 年であると計算	p.40
1995 年	コンウェイのチェーン表記	p.76
1996 年	ムナフォが巨大数のホームページを作成	p.11
2000 年 6 月 1 日	フリードマンの超越整数	p.229
2001 年 12 月	巨大数ベイクオフ大会でローダー数のプログラムが優勝	p.226
2002 年	バウアーズの配列表記と拡張配列表記	p.108

日付	出来事	ページ
2002 年 6 月 29 日	ふぃっしゅ数バージョン 1 の定義（2002 年にバージョン 4 まで）	p.87
2003 年 3 月 10 日	ベクレミシェフの虫	p.160
2003 年 3 月 13 日	小林銅蟲が巨大数研究室を開設	p.44
2003 年 4 月 5 日	ふぃっしゅ数バージョン 5 の定義	p.151
2006 年	バードが配列表記を考案し、5 変数配列表記がチェーン表記を超えると証明	p.114
2006 年 4 月 8 日	フリードマンがサブキュービックグラフ数を定義	p.215
2007 年	バウアーズが BEAF を開発	p.170
2007 年 1 月 26 日	ラヨ数の定義	p.248
2007 年 9 月 24 日	『巨大数論』の書きかけバージョン	p.260
2007 年 10 月 17 日	多変数アッカーマン関数の定義	p.81
2007 年 10 月 27 日	ふぃっしゅ数バージョン 6 の定義	p.201
2008 年 3 月	バウアーズがミーミーミーロッカプーワ・ウンパを定義	p.221
2008 年 12 月 5 日	Googology Wiki の開設	p.44
2008 年 12 月 9 日	サイビアンが巨大数の Web 書籍を公開	p.34
2009 年 9 月 2 日	フリードマンが有限約束ゲームを定義	p.232
2010 年 1 月 1 日	フリードマンが欲張りクリーク列を定義	p.234
2011 年 5 月 9 日	拡張チェーン表記の定義	p.86
2012 年 3 月 16 日	タラノフスキーの C 表記	p.200
2012 年 11 月 25 日	Aeton が巨大数動画の制作開始	p.77
2013 年 1 月 6 日	ガウチャーがクサイ関数を定義	p.244
2013 年 6 月 5 日	Wythagoras がドル関数を定義	p.149

日付	出来事	ページ
2013 年 9 月 11 日	小林銅蟲『寿司 虚空編』が裏サンデーの U-2 リーグで連載開始	p.44
2013 年 10 月 19 日	『巨大数論』初版でふぃっしゅ数バージョン 7 の定義	p.251
2013 年 11 月 10 日	Hyp cos の R 関数シリーズ	p.224
2013 年 12 月 5 日	巨大数研究 Wiki の開設	p.44
2014 年 8 月 14 日	バシク行列の元となる原始数列とペア数列のプログラムの発表	p.222
2014 年 10 月 30 日	Wojowu がビッグフットを定義	p.255
2015 年 4 月 13 日	『寿司 虚空編』の連載が pixiv コミックで再開	p.44
2015 年 4 月 26 日	ニコニコ超会議で巨大数の発表	p.262
2015 年 7 月 9 日	Hyp cos が強配列表記を定義	p.224
2016 年 9 月 6 日	鈴木真治『巨大数』の出版	p.44
2017 年 1 月 4 日	フィッシュ『巨大数入門』の出版	p.6
2017 年 1 月 5 日	Emlightened がリトルビッゲドンを定義	p.257
2017 年 3 月 27 日	Emlightened がサスクワッチ（ビッグビッゲドン）を定義	p.258
2017 年 6 月 29 日	フィッシュ『巨大数論』2 版（本書）出版	
2017 年 8 月 10 日	小林銅蟲『寿司 虚空編』単行本出版	p.44
2018 年 3 月 15 日	第 2 回東方巨大数開催	
2018 年 3 月 20 日	POD 個人出版アワードで本書が審査員特別賞（窓の杜賞）を受賞	p.6

http://gyafun.jp/ln/
電子書籍（PDF）はこちら

巨大数論　第 2 版

2013 年 10 月 19 日　初　版第 1 刷発行
2017 年 6 月 29 日　第 2 版第 1 刷発行
2018 年 10 月 13 日　第 2 版第 2 刷発行

フィッシュ 著
http://gyafun.jp/ln/

株式会社インプレス R&D
著者向け POD 出版サービス
https://open.nextpublishing.jp/author/

ISBN 978-4802093194