連続と計算
—総人とボトム—

立木 秀樹
人間・環境学研究科

我々は小さい頃から様々な計算の仕方を習ってきた。簡単な計算は日常の一部となっているし、コンピュータという複雑な計算を高速に実行するマシンに囲まれた生活をあたりまえのように行っている。ところで「計算」とは、いったい何だろうか。

1 デジタルな計算

アナログな計算も存在するが、我々が通常考えるのは、デジタルな計算のことである。そして、デジタルな計算とは、文字列による表現と、表現文字列上の機械的な操作のことと言っていいだろう。例えば、自然数の筆算による掛け算を考えよう。まず、数を 0 から 9 までの 10 種類の文字からなる文字列として表現する。数の 10 進表現は通常は意識することもないが、123 という数と”123”という 3 文字の文字列は別物であることに注意されたい。そして、2 つの数を表現する文字列を上下に書いて、横線を引っ張って、あの小学校の時に習った機械的な文字列操作で、掛け算の結果の数をあらわす文字列を構成する。これは、掛け算という意味を知らなくてもできる、全く機械的な操作である。機械的な操作だからこそ、コンピュータという意味を考えることのできない機械でもできてしまうのである。他の対象の上の他の操作の計算でも同じである。文字列に表現することと、文字列を操作する具体的な手順として演算を表記すること、それが計算の本質である。

では、文字列に表現できないものの上の計算はどうか？文字列に表現できないものとしては、人間の心とか、人間そのものとか、いろいろと考えられる。ここでは、これらの興味深いがそもそも計算とは馴染みそうにない対象ではなく、より具体的でみなさんよくご存知の実数について考えてみよう。実数が有限の文字列で表現できないことは、証明はともかくとして、直感的に理解できることだろう。π の値は 10 進で展開しようとしても、3.141592653589... と無限に続いて決して終わることはない。文字列で表現できなければ、上に書いた様な意味の計算はできない。コンピュータは実数に対しても計算を行っている様に見えるが、それは、実数の代わりに有限の文字列で書ける近似値を扱っているだけである。近似値だから誤差はつきものだし、誤差の評価を間違えるととんでもない計算結果をうのみにしたりする。そもそも、近似値による計算では、実数という連続な構造を持った対象の上で計算を行っているとはいえない。実数上で、近似ではない計算は考えられるだろうか。

2 実数上の計算

実数は、有限文字列では書けないが、無限の文字列としてなら表現することができる。無限の文字列に対しても、左から入力を読みながら左から出力を書くという手順で、機械的な文字列操作が可能である。例えば、3 で割るという手順はそのように実現可能であり、π を表現する無限文字列に対して行うと、1.04719755119... と暗算でも求めることができる。結果の無限文字列全体を有限時間内に

求めることは不可能だが、時間さえかければどこまででも計算ができるという意味で、これは、実数上の 3 で割るという演算の計算手続きと言える。この「無限列による表現と無限列を無限列に変換する手順」を計算と考えれば、実数の計算も可能な様に見える。では、3 をかけるという演算はどうか。π の表現に対してなら、暗算ででも、9.42477796076... と 3 倍にした無限文字列を計算することができよう。しかし、その手順は、常に適用可能とは言えない。たまたま入力が 3.33333... と続いていたとしよう。すると、3 をかけるという手順は、10.00000...（あるいは 9.9999）という結果を出さなくてはならない。しかし、入力無限列のどこまで見ても、最初を 1 にしていいか、9 にしていいか決められず、永遠に 1 文字も出力できないことになる。すなわち、3 をかけるという計算は、10 進展開という無限列による表現に対して計算不可能ということになる。この計算不可能性は、無限列と実数の構造（数学では位相といいます）の違いによる。無限列はあくまでも 0 と 1 の世界であり離散的であるが、実数は「連続」なのである。デジタルな計算は離散的な対象にしか定義できないというのは、一見、常識に合致するように見える。では、連続な実数上の計算は、どうすれば可能になるのだろうか。筆者の提案している方法について述べてみたい。

3 実数のグレイコード展開と⊥（ボトム）

自然数の 2 進コードについては、皆さんご存じであろう。コンピュータの中で自然数を 0、1 の列として表現するのに用いられている方法である。自然数を 0、1 の列として表現する方法は、この他にもいろいろ考えられる。2 進反転グレイコードはその中の一つである（図 1）。通常の 2 進コードでは、n 桁で表現しきれなくなり桁溢れを起こす時に、n+1 桁目を 1 にしてそれまでのコードを繰り返すのに対し、このコードではそれまでのコードを逆の順番に繰り返す。グレイコードの特徴として、数が 1 増えるときに、必ず 1 文字しか変化しないことがあげられる。2 進の時に 63 から 64 に変化するのに 011111 から 100000 に変わるが、そういった、急激な変化がないのである。

数	2 進コード	グレイコード
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

Figure 1: 自然数の 2 進コードとグレイコード

このグレイコードでもって実数を展開することを考える。図 2、図 3 は、[0、1] 区間に対して通常の 2 進展開と、グレイコードによる展開を比較したものである。線を引いてある部分は、対応する桁の値が 1、線のない部分は 0 であるとを意味している。（斜めの線は，綺麗に見せるための補助線。）さて、3/4 に対して 2 進展開は 110000... と 101111... と 2 つ存在していた。グレイコードでも同様で、111000... と 101000... の 2 つが存在する。しかし、これらは 1 文字（この場合は 2 文字目）しか違わない。他の所でも同様である。2 進でもグレイコードでも分母が 2 の冪である有理数に対して 2 つ展開が存在するが、それらは，1 文字しか違わず、その後の文字列は必ず 1000... である。展開の文字列は、数の値を指し示すためのものである。しかし、この

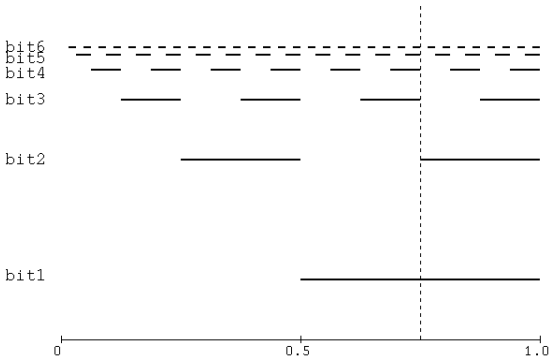

Figure 2: [0, 1] の 2 進展開

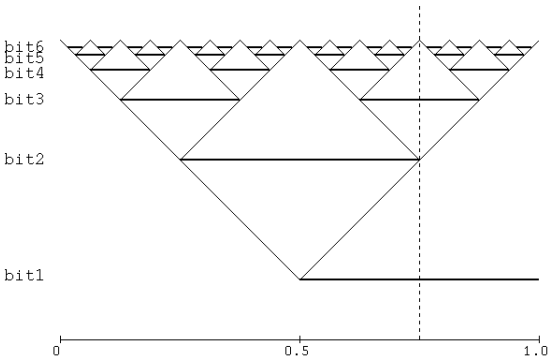

Figure 3: [0, 1] のグレイコード展開

2 文字目は、0 であっても 1 であっても結果の数は同じであり、この値が 3/4 であるということに貢献していない。よって、この部分は、0 か 1 に無理やり決めずに「0 か 1 か分からない」という状態に残しておく方が自然であろう。この、「0 か 1 か分からない」という状況のことを、⊥（ボトム）と書くことにする。そして、3/4 に対する展開は、1 ⊥ 0000... と定義しよう。他の所も同様にする。言い換えれば、図 3 の中で、線の端点は 0 と 1 が変化する所であるが、そこでの値を 0 とか 1 とか決めずに、⊥にするのである。

これにより、[0、1] 区間に含まれる実数全体を、0、1 の無限列だが、その中に⊥が高々1 つ含まれてもいいという文字列（1 ⊥-列と呼ぶことにします）として、一意に表現することができる。実は、この表現は、実数全体の 1 ⊥-列全体の集合への位相的埋め込みとなっている。これは、直感的には、両者の構造が同じで、実数を左から右に大きくしていった時に、対応する 1 ⊥-列が 011111... から 10000... に変わる様な急激な変化を起こさずになめらかに変化することといえる。

4　1 文字とばし

1 ⊥-列全体の上の計算の仕組みを考えよう。数の近似が時間と共によくなっていく状況、すなわち、数 x に対して、$a < x < b$ となる a、b が計算によって分かるとし、a、b が時間の経過とともに限りなく近づく状況を考え、それを元にして、x を表す 1 ⊥文字列を出力することを考えよう。もし、ある時点で $1/2 < x$ と分かったら、1 文字目に 1 を書くことができる。同様に、$x < 1/2$ と分かったら 0 を書くことができる。問題は、x=1/2 の時にはどちらも起きないことである。すなわち、1 文字目に 1 も 0 も書くことができない。これは、3 をかけるという演算が 10 進でうまく表現できなかった時と同じ問題である。しかし、x=1/2 なら、$1/4 < x < 3/4$ ということが、いつかは分かるはずである。その時、グレイコードなら、1 文字目を飛ばして 2 文字目に 1 を書くことができる。その後、もし $1/4 < x < 1/2$ または $1/2 < x < 3/4$ と分かったら、飛ばしていた 1 文字目に 0 または 1 を書くことができる。そして、$3/8 < x < 5/8$ と分かったら、3 文字目に 0 を書くことができる。そのようにして、x=1/2 の時には、1 文字目を飛ばしたまま、2 文字目以降に 1000... と書くことができる。よって、最初は全ての文字が⊥から始まるとすれば、x のグレイコードの値を出力したことになる。このようにして、前から 0 か 1 か決めていくのではなく、1 文字飛ばせるという、次の入出力の場所の自由度を与えることにより、1 ⊥-列を構成することができる（図 5 参照）。この 1 ⊥-列での表現とその上の計算の仕組みで、3 をかける演算も計算可能となる。

詳細は述べないが、空間（可分距離空間）

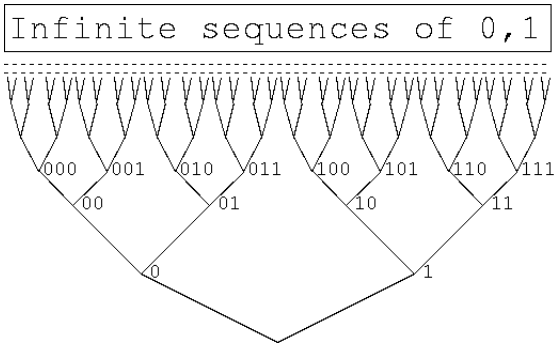

Figure 4: [0, 1] の無限文字列の近似構造

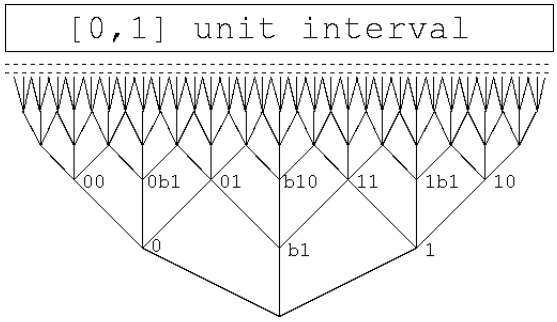

Figure 5: [0, 1] のグレイコード表現のもつ近似構造（b は⊥）

を表現するのに必要な⊥の個数が空間の次元と一致することが証明できる。次元というのは、直線が 1 次元で、平面が 2 次元で、空間は 3 次元でというあの次元である。次元は空間内での運動の自由度と考えられるが、それが文字列を埋める順番の自由度と対応していることは面白い。

5　総人と⊥（ボトム）

さて、ここの総合人間学部は、理系と文系という垣根を廃した学部である。垣根を廃すというのは連続につなげることかというのは議論が必要だが、少なくとも、「私は理系（文系）の研究をするのだ」という最初の 0、1 を決めないで、⊥に残したまま、みなさんは大学生になることができた。これから埋めることもできるし、ひょっとしたら、いつまでも埋めないでいることも可能かもしれない。総合人間学部では、この他にも、いくつもの⊥が用意されていると思う。学系という緩い単位に属することもそうだし、その学系を変える自由がかなり高いこともそうだ。ただ、自由度が高いことは、何も勉強しないで終わってしまう危険性が高いことだと意識してほしい。理系にも文系にも属さずに、その前の時点で（すなわち、どちらの勉強もせずに）止ってしまう人も少なからず存在する。1 文字目の 0、1 は決めなくても、その先の文字列を出力していかなくてはいけないのに、それをしないのだ。境界領域の研究をしたいのなら、両方の分野の勉強が必要となるのに、そういう認識が薄いのだ。

私自身、専門は数学なの?計算機科学なの?と聞かれても、どちらとも決めていないという、まさに⊥の状態である。しかし、この⊥というのは、「どちらでもない」のではない。両方ともが専門なのである（でなければならないと思っている。）計算の途中での⊥は、0 でも 1 でもないということではなく、0 になる可能性も 1 になる可能性もまだあるということを意味していた。1/2 の 1 桁目が⊥というのは，無限の極限までどちらの可能性も残して突き進んだということだ。どっちつかずのまま、両方とも勉強していないのでどちらにもなれないという状況は、⊥ではない。

境界領域に面白い問題が多いというのは確かだと思う。しかし、学生に許される自由度が高くて⊥の個数が多いと、それだけどれも埋められないで先に進めない学生も多くなると思う。自主性が求められる皆さんは大変だが、こころしてかかってほしい。

(京都大学総合人間学部図書館「バベルの図書館」第 8 巻 第 1 号 （通巻 14 号）)