実数の表現とグレイコード

立 木　秀 樹

1. タイプ 2 マシン

自然数 $1, 2, 3, \ldots$ 上の関数の計算可能性の定義には同値なものが幾つかあるが、その中に、チューリングマシンという仮想的な計算機を用いるものがある。チューリングマシンにも様々な同値な定式化があるが、ここでは、次のような n 入力 1 出力のものを考えることにしよう[*1)]。

n 本の入力テープ（テープは全て右に無限に伸びており、文字を書くことができる）と 1 本の出力テープ、それに、$k(k \geq 0)$ 本のワーキングテープがあって、それぞれのテープには、ヘッドが 1 つづつある。テープの文字集合はテープごとに違っていてもいいが、ここでは単純のため、全ての必要な文字を含んだ集合 Σ であるとする。Σ の文字の他に、空白を示す特別な文字、ブランク文字がテープには置けるものとする。マシンには、各テープのヘッド位置の文字を元にして次の動作を決める動作ルールが与えられている。動作としては、幾つかのヘッドの位置に文字を書いて幾つかのヘッドを左か右に 1 文字分移動するものと、Halt というマシンの動作を終了させるものがある。ここでは、ルールにもう少し制限を加えて、入出力テープのヘッドは右にしか移動することができないことにする。さらに、出力テープに関しては、文字を書く動作と右に移動する動作は一体として行なうこととして、各枠に対する書き込みは 1 度だけで、左から右に飛ばすことなくブランク以外の文字を出力する様に制限する。また、入力テープには書き込みを許さないことにする。

チューリングマシン M は、n 本の入力テープに入力文字列が与えられ（余白はブランク文字）、出力テープとワーキングテープはブランク文字で埋められ、各テープのヘッドは一番右にセットされた状態からスタートする。そして、ルールの適用を繰り返し、Halt を実行した時の出力テープの状態を M の出力と定義する。

自然数を文字列として表現する方法は、10 進法、2 進法などあるが、どれか固定しておくことにする。そして、自然数上の n 引数の関数 f に対し、$x_i(i = 1, \ldots, n)$ を表現する文字列を入力とした時 $f(x_1, \ldots, x_n)$ を表現する文字列を出力する様なチューリングマシン M が存在する時、 M が f を計算すると言い、f は計算可能（あるいは再帰的）な関数という。

このチューリングマシンの考え方は、再帰的関数論やアルゴリズムの理論の基本となっている。しかし、チューリングがこのマシンを考えた目的は、それだけではなく、実数の上に計算可能の概念を導入するためであったことは、あまり知られていないようだ[2)]。上の定義は、無限の長さの入出力を持つ時にも利用できる。そ

*1) プログラミングの経験のある人は、チューリングマシンの代わりに、C 言語の putchar(), getchar() みたいな 1 文字づつの入出力関数を用いて作られたプログラムを考えてもよい。

の際、Halt は必要ない。ブランク文字も、入力テープには不必要だ。無限に長い入力テープに、実数 $x_i (i = 1, \ldots, n)$ の表現（例えば、10 進や 2 進展開）をおいて、マシンを走らせると、出力テープに実数 $f(x_1, \ldots, x_n)$ の表現を無限に出力し続ける、そういうマシン M の存在によって、計算可能な実数上の関数 f を定義し、特に、$n = 0$ の時、すなわち、入力なしで実数の表現を出力し続けるマシンが存在する時、その実数を計算可能な実数と定義しようという訳である。このようなマシンは、タイプ 2 マシンと呼ばれている。チューリングマシンは停止しない時は意味をなさない。それに対し、タイプ 2 マシンは停止しないで、しかも無限に出力し続けることに意味がある。

この定義は、例えば、3.14159... と無限に数字の列をプリントし続けるプログラムが存在するからπは計算可能だという具合に、自然で分かりやすい。入出力は、実数以外のものの表現でもよい。有限／無限の文字列で表現できるものなら、何に対してでも、この方法で計算可能の概念を定義できる。入出力テープごとに何の表現かを選べるので、例えば、実数と自然数をもらって実数を返す関数だとか、実数上の連続関数をもらって実数上の連続関数を返す操作の計算可能性も考えることができる。しかし、本稿では、実数の表現に話を限ることにする。

このタイプ 2 マシンを用いた計算可能性は、表現として通常の 10 進や 2 進展開を用いたのでは、残念ながら、本特集の他の記事における定義とは異なる、不自然なものとなってしまう。

例えば、表現として 10 進展開を用いると、引数を 3 倍するという 1 引数の関数は、計算可能でなくなってしまう。なぜなら、この関数を表現するマシン M が存在したとしよう。そして、入力テープには、0.333333... と無限に続く入力が与えられたとしよう。この入力に対し、このマシンは、1.00000... または、0.99999... のどちらかを出力しないといけない。しかも、マシンに電源を入れてから有限の時間内に、1 桁目に 0 か 1 を出力しなくてはならない。しかし、それは不可能である。当然、それまでに読み込める入力の文字列は有限である。その時点で、マシンはその先の入力が 3333... とずっと続くことを予見できない。3332... となる可能性も 3334... となる可能性も考えて出力を決めなくてはならない。よって、どこまで入力を読んでも、結果が $0 \leq x \leq 1$ の範囲にあることも $1 \leq x \leq 2$ の範囲にあることも決定できないので、1 桁目を出力できないのである。10 進ではなく 2 進展開を用いても、同じような現象は起きる。

これは、タイプ 2 マシンの考え方が悪いのではなく、10 進展開、2 進展開という実数の表現を用いたのが不適切だっただけで、実際、他の表現を用いると、通常の計算可能と同値な定義を得ることができる。そして、このマシンを介した計算可能性の概念は、Type 2 theory of effectivity という、計算可能解析の一つのアプローチの基礎となっている [3]。

本稿の前半は、3) に沿って、表現の選び方と計算可能性との関係について解説する。後半は、1) に基づいて、グレイコードを用いた、より直接的な表現について解説する。

2. 無限列の計算可能性

実数について考える前に、無限文字列の計算可能性について考察しよう。

Σ を有限の記号の集合、Σ^* を Σ の記号の有限列の集合、Σ^ω を Σ の記号の無限列の集合とする。a, b を Σ の元に、α, β を Σ^* の元に、p, q, r を Σ^ω の元に用いることにする。無限文字列のことを、単に列と言うことにする。

n 入力のタイプ 2 マシン M が存在したとする。1 章で述べたように、M は、$(\Sigma^\omega)^n$ から Σ^ω への関数 f_M を定義している。ある入力に対してはマシンが止まらないこともあるので、f_M は、ある入力に対しては値が定まらない。そのような関数は、一般に部分関数と呼ばれている。このように、あるタイプ 2 マシン M によって、f_M と書ける様な列上の部分関数のことを、計

算可能な関数という。特に、0 引数の計算可能な関数のことを、計算可能な列という。

列の先頭から始まる有限部分 ($\in \Sigma^*$) のことのことを、接頭辞と呼ぶことにする。今、$\alpha \in \Sigma^*$ と $A \subset \Sigma^*$ に対して、α を接頭辞とする列全体の集合を $\alpha\Sigma^\omega (\subset \Sigma^\omega)$, A の要素を接頭辞とする列全体の集合を $A\Sigma^\omega (\subset \Sigma^\omega)$ と書くことにする。Σ^ω 上には、$A \subset \Sigma^*$ に対して、$A\Sigma^\omega$ という形をした集合を開集合とする位相（カントール位相）が入る。位相とは “近い” という概念を抽象化したものだが、この集合の意味を考えてみると、ある列が属しているという判断が接頭辞だけから分かる、つまり、ある列を前から観察してった時、その列がその集合に入っているなら、そのことが必ず有限時間内に判断できる集合ということになる[*2)]。もっとも、この判断が、実際に計算できるものとは限らない。計算ができるためには、A が計算によって入っているということが分かる集合、すなわち、帰納的に加算な集合である必要がある。その時、$A\Sigma^\omega$ を (無限列の上の) 帰納的に加算な集合と呼ぶ。

では、カントール位相に関して、関数が連続であるとはどういうことか？それは、$A\Sigma^\omega$ という結果を返す様な入力が開集合をなすということである。すなわち、α から始まる文字列を返すということが、先頭から有限な部分だけで決められる、そういった関数である。

実は、連続性は、計算可能関数のひとつの特徴である。f が計算可能とし、それを計算するタイプ 2 マシンを M とおく。M が α という出力を行なったなら、その瞬間の 入力テープのヘッド位置までの文字列を β とおくと、その先がどんな文字列でも α を出力をするはずである。よって、計算可能な関数は連続である。

*2) 入っていないということは、有限時間では判断できない。例えば、$A = \{0.1, 0.01, 0.001, \ldots\}$ とすると、$p \in A\Sigma^\omega$ ということは、前から見ていって 1 に出会った時点で分かる。しかし、0.0000... が A に入っていないことは、列全体を見ないと分からない。

3. 実数の表現と、表現に相対な計算可能性

定義 1 文字集合 Σ の上での実数の表現 (representation) とは、Σ^ω から 実数全体の集合 R への部分関数で、全射であるもののことである。

全射であることは、全ての実数に対して、表現が存在することを意味しており、関数でなく部分関数であるのは、列の中には、実数として無意味なものが含まれていてもいいことを意味している。δ が表現の時、実数 x に対し、$\delta(p) = x$ となる p のことを、x の δ 表現という。

実数表現 1 2進展開 δ_{bin} とは、列 $a_n \ldots a_0.a_{-1}a_{-2}\ldots$ $(n \geq 0,\ a_i = 0, 1)$ に対して $\Sigma_{i \leq n}\{a_i \cdot 2^i\}$ を返す関数である。

実数表現 2 10進展開 δ_{dec} も同様に定義される。

2 進展開で、$0.111\ldots$ も $1.000\ldots$ も 1 という同じ実数を意味している。このように、一つの実数に対して表現が複数あっても構わない。後で述べるように、このような表現の冗長性は、計算を考える上で重要である。

さて、表現を通して、列の上の計算可能の概念を実数の上に移すことができる。

定義 2 実数上の n 引数の関数 f に対し、列上の計算可能関数 f_M で、$x_i (i = 1, \ldots, n)$ の δ 表現に対し $f(x_1, \ldots, x_n)$ の δ 表現を返すものが存在する時、f は δ-計算可能という。

上で述べた様に、実数の δ 表現は一意とは限らない。よって、f を δ-計算可能にするタイプ 2 マシンは、x のどんな δ 表現に対しても、$f(x)$ の δ 表現のどれかを返す必要がある。

4. 符合つき 2 進展開

1 章で見たように、δ_{bin} や δ_{dec} は実数の計算可能性を考えるのに不適切であった。その理由は、例えば、求める実数が 1 であり、$1/2 < x < 3/2,\ 3/4 < x < 5/4,\ \ldots$ という具合に、計算によって x の存在範囲が狭まっていく時、

それを表現できなかったからである。この問題は、表現の持つ冗長性を上げることにより解決できる。

実数表現 3 符合つき 2 進展開 ρ とは、-1 を $\overline{1}$ と表記することにして、列 $a_n \ldots a_0.a_{-1}a_{-2}\ldots (n \geq 0, a_i = \overline{1}, 0, 1)$ に対して $\Sigma_{i \leq n}\{a_i \cdot 2^i\}$ を返す関数のことである。

この表現において、$\overline{1}$ を用いなければ 2 進展開と同じである。$\overline{1}$ を使えることにより、例えば、$3/4 = 1/2 + 1/4 = 1 - 1/4 = 1 - 1/2 + 1/4$ なので、$3/4$ に、0.110...、$1.0\overline{1}0$...、$1.\overline{1}10$... という複数の表現が与えられることになる。

有限列 α を接頭辞とする δ-表現を持つ実数の集合を $\delta(\alpha)$ と略記することにする。すると、$\rho(0.1)$ は、最大 $\rho(0.111\ldots)$ で最小 $\rho(0.1\overline{1}\overline{1}\ldots)$ なので、$0 \leq x \leq 1$ という閉区間 ($[0,1]$ と書く) になる。同様に、$\rho(1.0)$ は $[1/2, 3/2]$、$\rho(1.1)$ は $[1,2]$、$\rho(1.\overline{1})$ は $\rho(0.1)$ と等しく $[0,1]$ という区間になる。このように "1." で始まる小数 1 桁までの接頭辞の意味する区間が $1/2$ づつだぶっている。よって、求める実数がそのどれかに含まれることが分かれば、小数 1 桁目を出力できる。これにより、1 章の $\times 3$ の様な、δ_{bin} や δ_{dec} で計算不可能であった関数も ρ-計算可能となる。

5. 計算可能な関数

さて、二つの表現 ρ と δ_{bin} は、異なる計算可能の概念を実数関数の上に導いた。このように、表現ごとにその表現での計算可能の概念が定義できる。しかし、これでは実数の上の計算可能性を定義したことにはならない。表現に依存しない計算可能の概念を考えるために、まず、表現の間の関係について考えよう。

まず、ρ と δ_{bin} による表現を比べてみよう。上に見た様に、δ_{bin} 表現は ρ 表現でもある。そういう意味で、実数の δ_{bin} 表現から ρ 表現を構成することができる。逆はどうだろうか。ρ 表現から δ_{bin} 表現を計算により構成することはできるだろうか。もっと正確に言うと、1 入力のタイプ 2 マシンで、ρ 表現を δ_{bin} 表現に変換するものは存在するだろうか。それは、次のように不可能であることが分かる。δ_{bin} 表現の 1 の位のビットを見ると、その値が $x \geq 1$ か $x \leq 1$ のどちらか片方の情報が得られる。しかし、ρ-表現では、有限の接頭辞からその情報が得られるとは限らない。実際、$1.000\ldots$ という ρ-表現に対し、それは不可能である。つまり、表現の有限部分から分かる実数の情報として、δ_{bin} 表現は、ρ-表現以上のものを含んでいることになる。ρ-表現から δ_{bin} 表現への変換を行なうには、その情報を ρ-表現から構成する必要があり、それはタイプ 2 マシンでは不可能である。

このように、表現の間の変換可能性は、接頭辞から計算的に分かる情報の量の大小を表していることになる。

定義 3 δ と δ' を実数の 2 つの表現とする。δ 表現から δ' 表現への計算による変換が可能な時、すなわち、δ の定義域に属する任意の y に対し、$\delta(y) = \delta'(f(y))$ となる列上の計算可能関数 f が存在する時、δ が δ' に簡約可能 (reducible) であると言い、$\delta \leq \delta'$ と書く。$\delta \leq \delta'$ かつ $\delta' \leq \delta$ の時、δ と δ' は同値な表現とよび、$\delta \equiv \delta'$ と書く。

命題 1 $\delta_{dec} \leq \rho$, $\rho \not\leq \delta_{dec}$, $\delta_{bin} \leq \rho$, $\rho \not\leq \delta_{bin}$.

上での説明でも分かる様に、$\delta \leq \delta'$ の時、δ-計算可能実数は δ'-計算可能実数である。しかし、関数についてはこのことは言えない。例えば、符合つき 2 進展開の先頭に、有理数なら 1、無理数なら 0 をつけた表現 δ_R を考えると、$\delta_R \leq \rho$ であるが、実数をもらって有理数なら 1、無理数なら 0 を返す関数は δ_R-計算可能であるが、ρ-計算可能ではない。しかし、同値な表現は同じ計算可能関数の概念を導く。

あと 2 つ、よく用いられる表現を紹介しよう。有理数は整数のペアとして表現できる。整数は 2 進表記で文字列で表現できる。さらに、文字列のペアや文字列の列も、区切り記号を導入し

て文字列として表現することができる。よって、表現は無限文字列だが、有理数の列や、有理数のペアの列も、この様な記号化を通して、実数の表現として用いることができる。下の 2 つの表現は、このような記号化を用いて定義されている。

実数表現 4　区間の縮小列表現 ρ_I とは、$a_i < x < b_i$ となる有理数のペアの列で、a_i と b_i がそれぞれ同じ実数 x に収束するものを x の表現としたもの。

実数表現 5　実効的に収束するコーシー列表現 ρ_C とは、有理数のコーシー列で実効的に x に収束するもの (八杉の記事を参照) を x の表現としたもの。

これらの表現で p が x を表現しているとすると、p の途中から先もまた x を表現している。そういう意味で、前者 3 つよりも冗長性が高いが、その分、扱いやすい。つまり、こういったものを入出力するタイプ 2 マシンとして、様々なアルゴリズムを記述しやすい。

命題 2　$\rho \equiv \rho_I \equiv \rho_C$.

さて、これら 3 つの表現は、δ_{bin} より強いだけではなく、この種の表現の中で、もっとも強いものと言える。任意の有理数 u, v に対して $\{p \mid u < \delta(p) < v\}$ が開集合である様な表現の集合を K_t とする。また、この集合が帰納的に加算である様な表現の集合を K_c とする。$\delta \in K_t$ とは、開区間に含まれることが、接頭辞から分かるということを意味している。K_c は、K_t より強く ($K_c \subset K_t$)、$u < \delta(p) < v$ が成り立つならそのことをいつかは見つけ出すプログラムの存在を意味している。表現にとって接頭辞から分かる情報の量が重要だというという話をしたが、K_t や K_c は、近似による計算を成り立たせるのに必要なだけの情報を表現が持っていることを意味している。ρ, ρ_I, ρ_C は、K_c に属しているが、それだけではなく、k_t や K_c に属している表現の中でもっとも強いもの、言い替えれば、そういう情報しか持っていないものだということを示している。

定理 3　表現 δ に関して以下のことは互いに同値である。

1. $\delta \in K_t$.
2. δ は連続関数.
3. $\delta \leq_t \rho$.

また、以下の 2 つは同値である。

1. $\delta \in K_c$.
2. $\delta \leq \rho$.

ここで、$\leq_t$ というのは、定義 3 において f が計算可能というのを連続に置き換えて得られる関係である。ρ-計算可能（ρ_I-計算可能、ρ_C-計算可能でも同じ）な関数のクラスは、本特集の鷲原および森の記事で定義されている Pour-El, Richards 流の計算可能性と一致することが知られている。よって、ρ-計算可能な実関数のことを、単に計算可能な実関数と呼ぶことにする。

定理 4　計算可能な実関数は全て連続である。

ρ と同値な表現は全て計算可能の概念を導くが、そのような表現は、次のように、冗長性を持っていることが知られている。

定理 5　ρ と同値な表現は単射ではない。

6. 実数の列への埋め込み

実数の計算可能性の理論は、タイプ 2 マシンによる列上の計算可能性の定義と、表現による実数の計算可能性への結び付けの 2 段構えとなっている。しかし、この 1 段目と 2 段目で扱っている対象は、かなり性質が異なっている。カントール位相が示す様に、列上の構造は離散的あり、先頭が “0” の世界と先頭が “1” の世界は結び付いていない。一方で、実数は連続な構造を持っている。この両者を結び付けるために、表現を冗長にして糊しろを作って、列同士を貼りつける必要があった。このことより、1 段目

での計算が、全て 2 段目の計算を導出する訳ではなく、貼り合わせた所で同じ値をとるものだけが、実数の計算を意味していた。

実数の計算は、もっと直接的に定義できないだろうか。実数の上の計算可能性をそのまま反映した構造を無限列の上に構成できないだろうか。具体的には、表現を単射にして、実数を列の中に埋め込んでやり、それに適した新たな計算方法を列の上に考えることができないだろうか。

そういう動機で考えられた、グレイコードに基づく実数の表現について説明しよう。以下では、簡単のために、実数全体ではなく、0 と 1 の間の開区間 $I = (0, 1)$ の上で考えることにする。

7. グレイコード

グレイコード（Gray Code）は、2 進法とは異なる “0” と “1” による自然数の表現である。表 1 の様に、通常の 2 進コードでは、2^n から $2^{n+1} - 1$ のコードは、最上位ビットを立てて、0 から $2^n - 1$ までのコードを繰り返すことにより作られる。それに対して、グレイコードでは、最上位ビットを立てて、0 から $2^n - 1$ までのコードの順番をひっくり返すことにより作られる。このコードは、n から n+1 に移行した時に、1 ビットしか値が変化しないという特徴を持つ。例えば、通常の 2 進コードでは、01111111 の次は 10000000 と、全てのビットが変化する。ところが、グレイコードでは、これに対応して 010000000 の次が 110000000 となり、1 ビットだけしか変化しない。この特徴により、グレイコードは、回路の素子数の最小化や画像圧縮といった分野で応用されている。再帰的なプログラムの代表的例題であるハノイの塔の解とも関係がある (n ビット目の変化を n 番目の板の移動と見る) し、本特集で森氏の書かれているウォルシュ解析とも関係がある。

2 進コードからグレイコードへの変換は容易である。2 進コードを 1 ビット右にずらしたものと、ビットごとの排他的論理和をとってやればよい。逆変換も、もちろん簡単にかける。

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

表 1　自然数の 2 進コードとグレイコード

8. 実数のグレイコード表現

グレイコードは自然数に対するものだが、実数にも容易に拡張できる。上記の計算方法は無限列に対しても適用できるので、実数の 2 進展開の列に対してこの手続きを実行してやればよい。I の実数のグレイコード展開のそれぞれのビットの値を図示したら、図 8 の様になる。このように、全ての 2 進有理数 ($n/2^m$ という形の数) において対称性を持った、フラクタルな構造をしている。興味深いことに、この展開は、カオスの理論に現れる、テント関数

$$f(x) = \begin{cases} 2x & (0 \le x \le 1/2) \\ 2(1-x) & (1/2 \le x \le 1) \end{cases}$$

による記号力学系の旅程という概念とも一致している。

2 進展開は、2 進有理数において 2 つ存在した。グレイコード展開も、それに対応して 2 つ存在することになる。例えば、$3/4$ の 2 進展開は、0.110000... と 0.101111... である。それにに対応して、グレイコード展開も、0.111000.... と 0.101000... の 2 つが存在する。しかし、これらは 1 ビットしか違わない。一般に、2 進有理数の 2 つのグレイコード展開は 1 ビットしか違

図 1　実数のグレイコード展開

わず、その異なるビットの後は、必ず 10000... となる。

このように、実数の展開は、数を表現するための文字列なのに、$3/4$ に対しては、2 ビット目の値はその数が $3/4$ だということを表現するのに貢献していない。よって、$3/4$ の表現は、$0.1\perp 10000...$ とするのが自然だと考えられる。$\perp$ は、不定あるいは未定義を表している。こうすることにより、各実数に対して、それを表現する列を一意にすることができる。よって、以下のように実数のグレイコード表現を定義する。

定義 4　実数 x $(0<x<1)$ のグレイコード表現 $G(x)$ とは、$\{\perp, 0, 1\}$ という記号集合の上の列 $a_0a_1\ldots$ であって、以下のように a_i が構成されたものである。$m\times 2^{-i}-2^{-(i+1)}<x<m\times 2^{-i}+2^{-(i+1)}$ が、奇数 m に対して成り立つときには $a_i=1$、偶数 m に対して成り立つときには $a_i=0$、どちらも成り立たないときには $a_i=\perp$.

表現は、通常、列から実数への関数として定義されるが、この表現は単射なので、逆向きの関数 G によって定義することにする。この表現は、$\perp$ という “歯抜け” が存在するかもしれないが、基本的には “0” と “1” の列である。この表現を入出力するマシンを考えることにより、実数の上の計算を定義することを考えよう。

9. NM2-マシン

定理 5 から、単射であるグレイコード表現は、通常のタイプ 2 マシンとの組み合わせでは通常の計算可能と同じ計算可能性の概念を導けないことが分かる。例えば、計算によって、$a_1<x<b_1$, $a_2<x<b_2$, ... と実数の存在する範囲が狭まっていく計算過程を考えよう。$3/4$ を出力する計算を考えると、有限時間では、マシンはその値が $3/4$ ということを認識できないため、$\perp$ を 2 ビット目に出力することはできず、ここで出力がとまってしまう。しかし、確かに 3 ビット目は確定しないが、もし、計算によって、$5/8<x<7/8$ だということが分かったなら、その時点で 4 ビット目は 1 と確定する。よって、もし、マシンは前から順に出力するという制限を緩めたなら、3 ビット目をとばして 4 ビット目を 1 と出力することができる。その後、$5/8<x<3/4$ と分かれば飛ばしていた 3 ビット目に 1 を、$3/4<x<7/8$ と分かれば 3 ビット目に 0 を、そして、$11/16<x<13/16$ と分かれば 5 ビット目に 0 を出力するという具合に計算を進めることができる。出力値が $3/4$ の時には、最後まで 3 ビット目を埋めず、4 ビット目から先に $1000\ldots$ と出力することになる。よって、このマシンでは、$\perp$ はその桁の値が出力されてないことを意味し、ブランク文字が $\perp$ を表現していると考えればよいことが分かる。以下では、ブランク文字を $\perp$ と同一視する。

入力の方はどうか。入力テープに情報が与えられる順番は、上記の出力過程と同じものを想定するのが自然である。すなわち、最初は $\perp$ で埋まっていて、1 箇所だけ飛ばすことを許しながら、文字が書き込まれてゆく。後に上書きされるかもしれないので、マシンは、ヘッド位置の文字が $\perp$ であることに応じた動作はできない。しかし、それでは、例えば $3/4$ を入力しようとすると、3 文字目の入力で止まってしまう。それを防ぐために、2 つヘッド（メインヘッド

と先読みヘッド）を持つマシンを考える。先読みヘッドから文字入力があれば、メインヘッドはそのままで、先読みヘッドだけを前に進める、メインヘッドから入力があれば、メインヘッドを先読みヘッドの位置に、先読みヘッドは 1 文字前に進める。そういう動作でグレイコードを読み込むことができる。

動作ルールはどうか。単純のため、入力テープは 1 本としよう。すると、マシンの動作ルールは、2 つの入力ヘッドのどちらかの入力の値（先読みヘッドの入力の値はそれぞれの状況で一意に定まるので、パターンとしては 3 つ）と、ワーキングテープのヘッド位置の文字によって、次の動作を決める形となる。すると、2 つのヘッドの位置に共に入力がある場合には、可能な動作が 2 つ存在し、入力テープが埋められる順番になどに依存して、どちらかが選ばれることになる。よって、マシンの動作は非決定的となる。こういうマシンを NM2-マシンと呼ぶことにする。通常の非決定性と違うのは、どれかのパスが値を返すのではなく、全てのパスが同じ値を返さなくてはならないことである。

以上より、NM2-マシン が関数 f を計算するというのを、以下のように定義する。

定義 5　I^n から I への部分関数 f がグレイコード計算可能であるとは、n 入力の NM2-マシン M で、$G(x_1),\ldots,G(x_n)$ を入力すると、$G(f(x_1,\ldots,x_k))$ を出力するものが存在することである。

定理 6　実数上の関数のグレイコード計算可能と計算可能とは同値な概念である。

NM2-マシンの非決定性は、タイプ 2 マシンによる計算の定義の、ある数の全ての表現に対して同じ結果を出力するというのをマシンの中にシフトしただけの様にも思える。実際、符合付き 2 進展開を入出力するタイプ 2 マシンと、対応する NM2-マシンは、容易に変換可能である。しかし、グレイコードにより実数が文字列に埋め込まれていると考えると、NM2-マシンは実数に対してより直接的に計算の概念を定義しているとも考えられ、この非決定性が、実数の計算の一つの性質を表しているとも考えられる。

10. 位相的性質

NM2-マシンの出力の集合 $P \subset \{0,1,\perp\}^\omega$ の上に位相を考えよう。カントール位相と同様に、マシンによる有限の観察に対応した位相を考える。すなわち、NM2-マシンの有限時間での出力の集合を $Q \subset \{0,1,\perp\}^*$ とすると、$d \in Q$ に対し、d を観察した後に現れる可能性のある出力の集合を開基とする位相を考え、これを NM2-マシンが定める位相と定義する。

この位相は、次の様に、$\{0,1,\perp\}^\omega$ の部分集合の位相として自然である。グレイコードは $\{0,1,\perp\}$ を文字集合としているが、$\perp$ は 0 か 1 に書き換えられることを考えると、$\perp \leq 0$, $\perp \leq 1$ という順序が自然に考えられる。順序集合には、スコット位相と呼ばれる位相がはいる。この場合には、$\{\{\},\{0\},\{1\},\{0,1,\perp\}\}$ が開集合となる。そして、その積位相を $\{0,1,\perp\}^\omega$ 上に考え、その部分位相を P 上に考える。すると、その位相は、NM2-マシンが定める位相と一致する。

さて、P という（列からなる）位相空間ができた。実は、G は I を P に 1 対 1 に写像するだけでなく、同相写像となることが分かる。このことを用いると、定理 4（計算可能関数の連続性）は、直接導くことができる。

1) Hideki Tsuiki. Gray Code Representation of Exact Real Numbers. In *Proc. of 3rd Workshop on Computability and Complexity in Analysis,* (1998). http://www.i.h.kyoto-u.ac.jp/~tsuiki/cca/gray.ps
2) Alan M. Turing. On computable numbers, with an application to the Entscheidungsproblem, in *Proc. of the London Mathematical Society* (2) **42**, 230–265 (1936), and (2) **43**, 544–546 (1937).
3) K. Weihrauch. A simple introduction to computable analysis. *Technical Report 171-7/1995, Fern Universiat,* (1995).

（ついき・ひでき，京都大学、総合人間学部）