アルゴリズム概論 II

野　崎　昭　弘†

前回，問題の分類と，テューリング機械によるアルゴリズムの定式化，およびアルゴリズムの限界を示すひとつの定理を紹介した．今回は，残された問題：

アルゴリズムの評価，
正当性の表現，
停止性，
等価性

について，基本的な事柄をとりあげて解説してゆくことにしたい．

6. アルゴリズムの評価

よいアルゴリズムとは何であろうか．評価のしかたは，場合によって異なるであろうが，次のようなパラメータが（適当な重みをかけた上で）関係するであろう．

（a） わかりやすさ・保守のしやすさ，
（b） 準備すべきデータの量，フォーマットの簡単さ，
（c） 使用する機器の簡単さ，
（d） 消費時間が短いこと，
（e） 消費する記憶容量が小さくてすむこと．

これらのうち，(a),(b)は実用上重要であろうが，定量的に論ずることは困難である．また(c)も，それぞれの場所ごとに，『現有の機器で処理できるか，どうか』といういわば1ビットのパラメータなので，一般的には論じがたい．ただ，記憶容量の問題だけは，ある程度定量的に議論できるので，別にわけた．消費時間の評価は，かなり一般的で，しかも定量的な議論ができるので，昔からいろいろと研究されているところである．ここでも(d)を中心とし，(e)をつけ加えるという形で，述べることにしよう．

消費時間を測る尺度としては，計算機によって異なるサイクル・タイムの差が響かないように，適当に基本操作をえらび，そ（れら）の基本操作が『何回必要であるか』という形で評価を行なうのがふつうである．ただ基本操作として何をえらぶかは，問題により，また場合によって著しく異なる．

昔から知られているパズルを例にとるなら，次のような問題がある．

『1個だけ重さが異なる，n個の球がある．天秤にかけることによって，重さの異なる球をみつけだすためには，何回天秤を使えば充分か？最悪の場合の天秤使用回数が最小になるような，アルゴリズムを示せ．ただし，天秤の皿には，1回に1個以上何個のせてもよいものとする．』

たとえば，$n=6$ の場合には，次のようなアルゴリズムがある．

アルゴリズム A．まず勝手な2個甲，乙をえらんで，天秤で比較する．

場合 1．それらが等しかった場合――それは“標準の”（多数派）球であり，重さが異る球（以下これを**求める球**と呼ぶ）は残り4個の中にある．したがって，残り4個をひとつずつ，標準の球（最初にえらんだ2個のうち，任意の一方）と比較してゆけば，かならず求める球がわかる．

場合 2．それらが一致しなかった場合――それらのどちらかが求める球で，残りの4個が標準の球である．したがって，それらの一方を（たとえば甲），標準の球と比較すれば，一致しなかったときはその球（甲）が，一致したときには他方（乙）が求める球である．

このアルゴリズムの場合，運がよければ2回の比較で，最悪の場合でも5回の比較で求める球がわかる．また平均比較回数は，次のように計算される．

場合1が起こる確率 $p=\frac{{}_5C_2}{{}_6C_2}=\frac{2}{3}$,

平均比較回数

$$=1+\left(p\times\frac{1+2+3+4}{4}+(1-p)\times 1\right)$$

$$=3.$$

アルゴリズム B．まず天秤の両側に2個ずつ，計4個をのせて比較する．

† 東京大学理学部情報科学研究施設

場合 1. それらが等しかった場合——残りの2個甲，乙の一方（たとえば甲）と，最初にえらばれた4個のうちの任意のひとつ（これは標準の球）と比較する．一致すれば乙が，一致しなければ甲が，求める球である．

場合 2. それらが一致しなかった場合——天秤の両側から1個ずつを除いて比較してみる．残りが一致すれば，求める球は今除いた2個のうちのどちらかであるから，その一方と標準の球とを比較すれば，どちらかわかる．また，残りが等しくなければ，そのどちらかが求める球だから，やはりその一方と標準の球とを比較して，どちらかあてられる．

このアルゴリズムによれば，運がよければ2回，最悪の場合でも3回の比較で求める球がわかる．平均比較回数は，次のように計算される．

$$\frac{1}{3}\times 2+\frac{2}{3}\times 3=2\frac{2}{3}.$$

こうして，Bの方が（少なくとも時間に関して）よいアルゴリズムであることがわかった．最悪の場合でも2回の比較で求める球をあてるようなアルゴリズムは存在しない（証明を要するが）から，アルゴリズムBはひとつの正解である．

これは，天秤による比較を基本とする，アルゴリズムの評価である．ここで，

最悪の場合のステップ数

と

平均のステップ数

というふたつの概念があらわれたことに注意しておこう．例として引用した問題では，前者について問うているが，実用上は後者の方が重要であることが多い．ただ平均値が算出しにくい場合や，両者が接近していると考えられる場合には，前者による評価が行なわれる．この例題についていえば，平均値まで考えるとアルゴリズムBは最適でなく，もっとよい方法がある．

与えられたデータを小さい順に並べよという，いわゆる分類（sorting）の問題についても，同じような理論がある[11]．この場合には，基本操作として

2数の比較

いいかえれば，両側の皿に1個ずつしかのせられない天秤による比較をえらぶことが多い．また主記憶装置の中で，ランダムに並べられたデータを小さい順に『並べかえる』ことが要求される場合には，

2数の交換（入れかえ）

をも基本操作に含めて，評価を行なう．必要な比較回数は，データの回数を n として，最悪の場合を $M(n)$，平均を $m(n)$ とすると，組合わせ論的考察から

$$M(n)\geqq\log_2(n!)$$

が，情報理論的考察から

$$M(n)\geqq m(n)\geqq\log_2(n!)$$

がわかる[11]．なお

$$\log_2(n!)\fallingdotseq(\log_2 n-1.443)\left(n+\frac{1}{2}\right)$$
$$\fallingdotseq n\log_2 n$$

である．

$n\log_2 n$ というオーダーは，併合法（sort by merge）によっても，Floyd の Treesort によっても達成される[10]．しかし，$M(n)$（あるいは $m(n)$）の正確な値は未だわずかしか知られていない．これまでに知られているのは，

(1°)　$n\leqq 11$ のとき，$M(n)=\lceil\log_2(n!)\rceil$,

(2°)　$n=20,21$ のときも，同様，

(3°)　$n=12$ のとき，$M(n)=\lceil\log_2(n!)\rceil+1$

だけである（(3°) は M. Wells によってたしかめられた[13),18)]）．ここで「x」は x 以上の最小の整数を意味する．これらの理想値は Ford-Johnson のトーナメント法[11]で達成される．

一般の数値計算の場合には，乗除算と加減算（あるいは，後者を無視して乗除算のみ）を基本操作と考え，それらの実行回数を評価することが多い．たとえば，2次式

$$ax^2+bx+c$$

を計算するのにも，

$$a\times x\times x+b\times x+c$$

を計算すると3回の乗算が必要であるが，

$$(a\times x+b)\times x+c \qquad (1)$$

になおして計算すれば2回ですむ．さらに，

$$a=1,\quad b=3,\quad c=-5$$

などと具体的に与えられている場合には，

$$(x+1.5)^2-7.25 \qquad (2)$$

を計算することにすれば，乗算は1回ですむ．（この式を何回も使用する場合には，1.5 や 7.25 を求める手間は全体として無視できるであろう．）

一般の n 次多項式に対しても，この種の工夫ができる．係数が不定のばあい，あるいは1回かぎりの計算では，（1）の形で計算するのが最適である（V. A. Pan[15]）．また，同じ係数について計算をくり返す場合には，係数についてあらかじめ（1回だけ）計算することを許すと，乗算が高々

$$\left\lfloor \frac{1}{2}(n+4) \right\rfloor$$

回ですむように，式を変形できる[15]．ここで $\lfloor x \rfloor$ は x 以下の最大の整数を意味する．これは基本関数を計算するサブルーチンなどに適しているので，推奨されてよい方法と思われる．

線型計算についても，同じようなアルゴリズム上の工夫が可能である．たとえば n 次正方行列の積を求めるという，基本的な問題を考えてみよう．積のひとつの成分 c_{ij} を求めるたびに，

$$c_{ij}=a_{i1}b_{1j}+a_{i2}b_{2j}+\cdots+a_{in}b_{nj}$$

という形の，“n 次ベクトルの内積”を計算すると，そのたびに n 回の乗算が必要である（証明はなかなかむずかしい[19]）．全部で n^2 個の成分を求めなければならないので，n^3 回の乗算が要る．

驚くべきことは，この n^3 回という数字が最小ではなく，アルゴリズムを工夫することによってさらに小さくできる，ということである．これは奇妙なことのようであるが，n^2 回の成分を求める計算が全く独立でなく，互いに関連しあっているので，ひとつだけとりだして求めようとすると n 回の乗算が必要なのであるが，全体としては n^3 回より少なくできるのである．たとえば Winograd の方法によれば約 $1/2n^3$ 回[20]，Strassen の方法によれば，$n^{2.807\cdots}$ $(2.807\cdots=\log_2 7)$ のオーダーの比較で積を求めることができる[16]．乗算の回数をこれより少なくするアルゴリズムは未だ知られていないが，Strassen の方法が最適であるという証明もまだできていない[12]．

なお，ここで注意すべきことは，『基本操作の回数』という尺度が，実際の計算時間にどれくらい反映するか，ということである．簡単なプログラムならば，基本操作の回数は実際の計算時間（入出力は除く）の主要項を占めるであろうが，アルゴリズムが複雑で，面倒な添字（番地）計算を伴う場合には，オーバーヘッドの部分が無視できず，思ったほどの効果が挙げられない場合がある．たとえば Ford-Johnson のトーナメント法は，比較回数に関するかぎり今まで知られているものの中で最良の方法であるが，データのならべかえ（あるいはポインタ操作）のオーバーヘッドがかなり出ると思われるので，Floyd の Treesort に比べて実際の能率はむしろ悪いであろう．また，行列の積を Winograd 法で計算したときの所要時間については，日科技研の矢島敬二氏等の詳しい実験がある[20]が，それほどのスピード・アップはできず，むしろプログラムを書く言語（フォートランか，アセンブラ語か）の方が時間に大きく影響することが知られている．Strassen の方法については，まだ充分な検討がなされていないようであるが，

1°) 小さなサイズ（$n\leqq 2^5$ ぐらい）では，かえって損である，

2°) 大きなサイズでは，記憶容量を余計に使うので，大型機でないと計算できなくなる

という実験報告がある（東大大学院生，佐藤中君）．オーバーヘッドを小さくすることには，プログラミング上の技巧も大きく関係するので，速断はできないが，今後の検討が待たれるところである．

アルゴリズムの工夫によって，計算時間が眼に見えて短縮されたケースももちろんたくさんある．中でも有名なのは高速フーリエ変換（FFT）であるが，これについては本誌第14巻8号に解説がのっているから，そちらをご参照いただきたい[17]．この方面では他にも細かい結果がたくさんあり[9),14]，また未解決の問題もごろごろしている．

さて，このような評価とは一見かけはなれた方法として，チューリング機械による時間評価の方法がある．これは，基本操作をずっと大づかみにとり，いわば『主記憶装置内で計算できること』を1ステップとみなす方法である．それでも問題が大きくなると，主記憶装置におさまりきれなくなり，磁気テープその他とのデータ交換が必要になる．この外部補助記憶装置に頼らざるをえない部分の複雑さを評価するには，チューリング機械による時間評価が適当である．

話を具体的にするために，n 台の磁気テープ装置をもつ計算機システムを考えよう．簡単のために，CPU はこれらのテープ装置に同時に，独立に読み書きができるものとする．

このシステムは，近似的に，次のようなチューリング機械と同等である．すなわち，CPU と等価な本体と，ます目に仕切られた（ここがもとのシステムと一致しない）n 本のテープをもつ機械 M である．M の本体は，その内部状態 q と，各テープのます目に書かれた情報

$$x_1, x_2, \cdots, x_n$$

とに応じて，状態を q' に変え，テープの情報を

$$y_1, y_2, \cdots, y_n$$

に書きかえて，各テープを独立に，右または左に1ます分動かすか，あるいは動かさずにおくか，または全体としての動作を停止するかをえらぶ．（この選択と

q' および $y_1, \cdots, y_n$ は，q および $x_1, \cdots, x_n$ から一意的に決定されると考える.）これを基本の1ステップとして，きめられた初期状態から，停止するまで動作を続ける.

ここで再び，記号列 α に記号列 $\varphi(\alpha)$ を対応させる関数 φ を考えよう．上記のテューリング機械 M が，この関数 φ を計算するとは，第1のテープに α を書き，他のテープは全部空白として，ヘッドを α の左端において出発させると，やがて（有限時間内に，必ず）停止して，どれかのテープに記号列 β を残すことをいう．（β の位置は，わかるようになっていなければならない.）このとき，停止するまでの基本ステップの回数を M にする φ の**時間計算量**といい，$T_M(\alpha)$ であらわす．また，使用されたます目の個数を**テープ計算量**といい，$L_M(\alpha)$ であらわす．これらの関数の値が，α の長さ n に対して，どれくらいのオーダーになるかが興味の対象で，たとえば

$$T_M(\alpha) \leqq cn$$

が示されるなら，磁気テープの操作に要する時間はせいぜい n に比例する程度でおさえられる，ということができる.

この評価は，本体 (CPU) だけでできる計算は全部1ステップの中に押しこめてしまっているので，きわめて荒っぽい評価である．本体の機能が何倍かになれば，時間計算量は何分の一かになってしまうであろう．（実際，このことはある簡単な条件のもとで証明でき，**加速定理**と呼ばれている.）しかし，本体をどのように拡張・改良しても，計算量のオーダー（n^2 か，n^3 か，$n\log n$ か，等々）は下げられないことがある．そこで，下げられる（なるべく）ギリギリのオーダーが，調べられるわけである.

よく知られた結果の一例として，計算機プログラムの文法チェックの仕事量をいくつか挙げておこう．それはもちろん，そのプログラムが書かれた言語 L の性質によって変わるのであるが，問題をもう少し正確に述べれば次のようになる.

プログラムを記号列と考え（カードの切れめも記号と考える），α であらわす．すると，プログラミング言語 L をひとつきめたとき，

$$\varphi(\alpha)=\begin{cases}1 \cdots \alpha \text{ は L の中で文法的に正しい,}\\ 0 \cdots \alpha \text{ は L の中で誤り}\end{cases}$$

によって関数 φ が定義できる．そしてこの φ を計算するテューリング機械の，時間計算量やテープ計算量の上界を問題にするのである．（機械を下手に作れば上界はいくらでも大きくなってしまうので，ここでは上手に作ったときの，いわば“最小”上界を考えるのである.）すると，L の性質に応じて，表1に示すような上界が得られている[7].

表 1

言語 L の性質	時間計算量	テープ計算量
正規言語	n	0
決定性文脈自由型言語	n	n
線型言語	n^2	n
文脈自由型言語	n^3	$(\log n)^2$

テープ計算量を測るときは，データ α を書く部分を除き，そこは read only で書きかえができないと仮定する.

なお表中，文脈自由型言語の場合の時間計算量の上界は，理論的に“最小”かどうか不明である．また，テープ計算量を $(\log n)^2$ まで下げようとすると，今まで知られているテューリング機械では，時間計算量が n^3 を越えてしまうので，これらの上界を同時に実現できるかどうかはわかっていない.

最後に，計算量のもっと質的な分類として，どのような手続きで関数が構成できるか，に着目した分類を紹介しておこう．簡単のために，非負整数

$$0, 1, 2, 3, \cdots$$

に非負整数を対応させる，多変数関数（数論的関数）について述べる．（記号列に記号列を対応させる関数も，記号列を何進法かの整数表示として読めば，数論的関数に翻訳できる.）まず，基本的な関数として，

$$C_0(x)=5,$$
$$N(x)=x+1,$$
$$P_i^n(x_1, \cdots, x_n)=x_i$$

を考える．そして，これらの関数から，次の手続きを有限回くり返して得られる関数を，**原始帰納的関数**と呼ぶ.

（1） 合成：n 変数関数 f，m 変数関数 $g_1, \cdots, g_n$ から，関数 h を次のように定義すること.

$$h(x_1, \cdots, x_m)=f(g_1(x_1, \cdots, x_m), \cdots, g_n(x_1, \cdots, x_m))$$

（2） 帰納：$n-1$ 変数関数 f，$n+1$ 変数関数 g から，n 変数関数 h を次のように定義すること.

$$h(x_1, \cdots, x_{n-1}, 0)=f(x_1, \cdots, x_{n-1}) \qquad (2.1)$$
$$h(x_1, \cdots, x_{n-1}, x+1) \qquad (2.2)$$
$$=g(x_1, \cdots, x_{n-1}, x, h(x_1, \cdots, x_{n-1}, x))$$

ただし $n=1$ のときは，f として定数を考える.

〔例〕 $G(x, y, z)=z+1$ は，基本関数 N および P_3^3 から合成できるから，原始帰納的である．また，

$$\begin{cases} h(x,0)=P_1^1(x), \\ h(x,y+1)=G(x,y,h(x,y)) \end{cases}$$

とおくと，h は P_1^1 と G から帰納によって構成できるので，これも原始帰納的である．実は

$$h(x,y)=x+y$$

なので，『加算が原始帰納的』とわかったことになる．

合成・帰納の他に，次の操作もゆるして定義できる関数を**帰納的関数**という．

（3） 最小化：$n+1$ 変数関数 f が，次の条件をみたすとする．任意の $x_1,\cdots,x_n$ に対して，適当に y をえらべば，

$$f(x_1,\cdots,x_n,y)=0.$$

このような f から，関数 h を次のように定義することを，最小化という．

$$h(x_1,\cdots,x_n)=\text{“}f(x_1,\cdots,x_n,y)=0 \text{ となるような } y \text{ の最小値”}$$

帰納的ではあるが原始帰納的でない関数の例としては，次の方程式で定義される Ackerman 関数 A が有名である[6]．

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

帰納的関数は，必ず適当なテューリング機械で（ごく自然な意味で）計算されるし，また逆も成り立つ．これは前の一般論に含めるべきであったが，テューリングの機械の一般性を信じる根拠のひとつである．

7. 正当性・停止性・等価性

最初に，事柄をはっきりさせるために，図1のような流れ図を考えてみよう．これは，何をするプログラムであろうか？

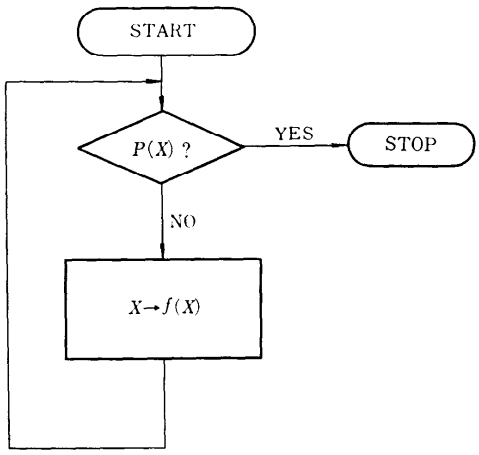

図 1

このプログラムの機能をあらわすには，大ざっぱにいって二つの方法がある．

（1） プログラムが含む変数

$$X=(x_1,\cdots,x_n)$$

に対する関数として表現する．たとえば，アルゴル風の記法を借りれば，

$$F(X)=\textbf{if } P(X) \quad \textbf{then } X \quad \textbf{else } F(f(X)) \quad \textbf{end}$$

（2） 変数 X の出発点での値と，停止した時点での値の因果関係を，論理記号であらわす．たとえば，

出発点で：$A(X)=\text{“}x_1=x_2=n\text{”}$

終点で：$B(X)=\text{“}x_2=1+2+\cdots+n\text{”}$

第一の表現をえらんだ場合には，正当性の問題とは，プログラムが果たして指定された関数 F_0 をほんとうに計算してくれるかをたしかめる，ということになるであろう．第二の表現をえらんだ場合には，それは，出発点で $A(X)$ が成り立つと仮定したとき，終点でかならず $B(X)$ が成り立つかどうかをたしかめる，ということになるであろう．いくつかの例を挙げるなら，

（i） Floyd が行なったプログラムの意味づけは，(2)の立場によっている[21]．

（ii） McCarthy が行なったコンパイラの正当性の証明は，source program が表現している関数と，object program が表現している関数との同値性を示すこと，すなわち(1)の立場である[22]．

（iii） Yanov らが考察したプログラムの等価性は，具体的な意味づけを極力除いた形で，(1)の立場をとっているとみることができる[26]．

（iv） 伊藤貴康らが構成したプログラムの代数的理論は，プログラムを（帰納的関数の定義のように）構成的に定義し，その構成手続きについての一意性定理によって，正当性や同値性を証明するものである[22]．したがってこれも(1)の立場をとるとみることができる．

これらのおのおのについて，さらには Scott によるプログラム束の理論の紹介にまで詳しく立ち入ることは，筆者の任でない．それらはふつういうアルゴリズムの理論とは独立の，大きな分野である．ここではその分野——プログラミング理論の紹介にはなっていないことをお断りした上で，まず最もわかりやすいと思われるいくつかの基本概念を紹介し，またこの方面で

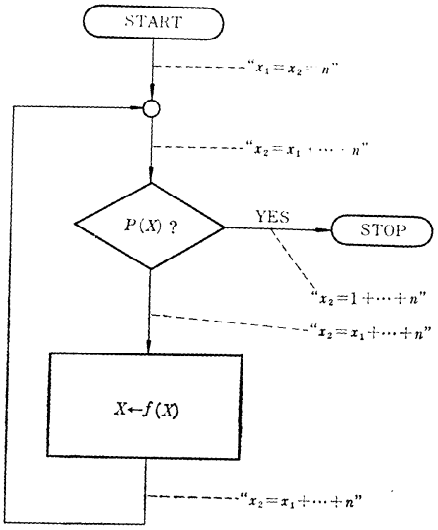

図 2

のアルゴリズム的問題，すなわち『同値性の証明の複雑さ』についての触れておくことにしたい．

正当性について　Floyd は，プログラムの意味づけを，『プログラマにコメントを書かせる』ことによって与えることにした．たとえば図2のように，各矢印に，そこで変数値がみたすべき条件を，結びつけるのである．

各条件は，変数値の組 X についての命題と考えられるので，以下次のように書くことにする．

$A(X)=$"$x_1=x_2=n$",

$B(X)=$"$x_2=x_1+\cdots+n$",

$C(X)=$"$x_2=1+2+\cdots+n$",

$D(X)=E(X)=B(X)$.

次に，これらのコメントが正しいかどうかの吟味にとりかかる．それにはまず，分岐条件 P とオペレーション f がわかっていないといけないので，

$P(X)=$"$x_1=0$",

$f(X)=(x_1-1, x_2+(x_1-1))$

とおくことにする．f は，いいかえれば，

$x_1\leftarrow x_1-1$,　　$x_2\leftarrow x_2+x_1$

ということである．すると，コメント $A,\cdots,E$ が正しいかどうかは，次の諸条件が成り立つかどうかできまる．（これらを**検証条件**という．）

（1） オペレーション f の前後で（図3（a））：

$R(X)\rightarrow Q(f(X))$

（2） 分岐 Φ の前後で（図3（b））：

$$\begin{cases} R(X)\wedge\Phi(X)\rightarrow Q_1(X), \\ R(X)\wedge(\neg\Phi(X))\rightarrow Q_2(X) \end{cases}$$

（3） 分流点の前後で（図3（c））：

$R_1(X)\vee R_2(X)\vee\cdots\vee R_t(X)\rightarrow Q(X)$.

図2については，これら全部が成り立つから，出発点の条件 $A(X)$ さえ正しければ，どのような計算の流れが起こっても，通過する矢印のすべてに対して，それに結びつけられた命題が成り立ってくれる．それゆえ，終点に達したときには，命題 $C(X)$ が成り立つ．

これで，このプログラムは，条件 $A(X)$ のもとで，停止したときに $C(X)$ を成り立たせる（すなわち，x_2 に $1+2+\cdots+n$ を求める）プログラムであること

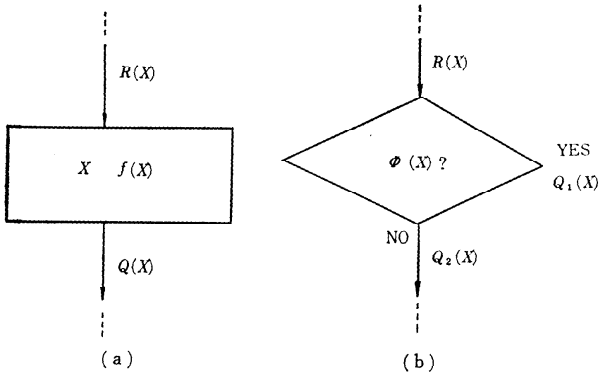

図 3

がわかった.

ここで,『停止したときに』という条件は重要である.このプログラムが停止するということはまだ証明されていないし,また実は停止しないかもしれないのである.このような条件つきの正当性を,弱い意味の正当性という.

停止性について 図2の例では,Aに条件

"nは非負整数である"

をつけ加えなければ,プログラムは停止してくれない.また,この条件をつけ加えたとしても,停止することの証明は簡単ではない.

すぐ思いつかれる方法は,ループをまわるたびに,x_1の値が減少することに注目することであろう.そうすれば,$x_1=n$についての帰納法によって,停止性が証明される[23].

もう少し形式的にやるには,図2の代りに,関数表現を利用するとよい.再掲すると,

$$F(X)=\textbf{if } P(X) \quad \textbf{then } X \quad \textbf{else } F(f(X)) \quad \textbf{end}$$

そこで,

"$F(X)$の計算が停止する"

という命題を

$t(X)$

であらわしてみると,右辺の計算が停止すればよいのだから,

$$[P(X)\vee t(f(X))]\rightarrow t(X)$$

が得られる.これは,さっきやった数学的帰納法を,はっきり示唆してくれる式である.

このような式から,停止性の証明を完成するには,次の事実が便利である.まず前提として,考えるべき変数値 X の集合 $\boldsymbol{X}$ に,ある順序 $\prec$ を定義しておく.たとえば,

$$\boldsymbol{X}=\{(m,n)\mid \ m,n \text{ は非負整数}\},$$
$$(m,n)\prec(m',n')$$
$$\Leftrightarrow[m<m' \text{ であるかまたは,}\ (m=m' \text{ かつ } n<n')]$$

すると,この順序$\prec$は,いわゆる降鎖律をみたす.すなわち,Xの列

$$X_1\succ X_2\succ X_3\succ\cdots$$

は,必ず有限の長さで切れる.

さて,関数 F の定義から導いた式

$$P(X)\vee t(f(X))\rightarrow t(X)$$

について,もし

$$P(X)\vee(f(X)\prec X) \qquad (*)$$

が示せるなら,降鎖列についての一般論から,必ず$t(X)$が真になることがいえるのである.式(*)の証明はこの例では容易で,たちまち証明が完成するが,(*)のような式を,形式的に導くことができるのがこの方法の強みである.(ただしいつでも成功するとは限らない.)

Ackermann関数の停止性にこの方法を応用すると,次のようになる.($\boldsymbol{X},\prec$ は前のとおりとする.)

$$\begin{cases} x=0\rightarrow t(x,y), \\ t(x,1)\rightarrow t(x+1,0), \\ t(x,A(x+1,y))\wedge t(x+1,y)\rightarrow t(x+1,y+1). \end{cases}$$

ここから,証明すべき条件は,

$$\begin{cases} (x,1)\prec(x+1,0), \\ (x+1,y)\prec(x+1,y+1), \\ (x,A(x+1,y))\prec(x+1,y+1). \end{cases}$$

これらは自明である.

等価性について 再び図1を考えよう.このプログラムでありうる計算の流れ,たとえば3回ループをまわってから停止することは,その流れにあらわれる命題や関数をならべてあらわすことができる.

$$\neg P, f, \neg P, f, \neg P, f, P$$

ありうる計算の流れの全体は,このような記号列の集合としてあらわすことができる.

すなわち:

$$L_A=\{((\neg P)f)^nP\mid n\geqq 0\}.$$

ただし β^n は,記号列 β を n 回ならべたものをあらわす.

プログラム A,B が与えられたとき,このような集合 L_A, L_B を考えると,

$$L_A=L_B$$

は,プログラムが同じ関数を計算するための十分条件になっている.もう少しくわしくいえば,この等式は,『命題に副作用(すなわち変数 X の値の変更)がありうるとして,関数・命題の具体的な定義を与えたとき,それがどんな定義であっても,A と B とは同じ(部分)関数を計算する』ための必要十分条件になっている.

命題の副作用を禁止した場合にも,L_A, L_B の定義を少し修正することによって,同じようなことがいえる.また,A,B が recursive call を含む場合にも,やはり"ありうる計算の流れ"から記号列の集合が定義できて,同じ議論ができる.

ここで,

$$L_A=L_B \qquad (**)$$

の決定問題を考えると，次のことがいえる．

（1） recursive call をゆるす一般のプログラムについては，命題の副作用をゆるすなら等価性(＊＊)は決定可能，そうでない場合は未解決の難問である．

（2） recursive call をゆるす場合でも，exit の直前に必ずある（命題でない）オペレーションが実行されるようなプログラムどうしなら，命題の副作用の有無にかかわらず，(＊＊)は決定可能である．

（3） 少なくとも一方が recursive call を含まないプログラムであれば，命題の副作用の有無にかかわらず，(＊＊)は決定可能である．

決定可能な場合については，決定のための手間の評価を示すことができる．今のところ決定可能性の証明には，ある長さ以下の記号列を全部調べればよい，という形で行なわれるので，その『長さ』が決定の手間の尺度に使えるであろう．そこでその長さを評価してみると，次のようになる．

（1），（2）で決定可能の場合，

$cN^{(c+1)(c+3)}$

（2）で，entry の直前には必ずある特定のオペレーション ENT，exit の直前には必ずある特定のオペレーション EXT が実行されるとした場合，

cNK

両方が recursive call を含まない場合，

cN

ただし

N＝プログラム A，B の節点の総数，

K＝節点から終点への最短経路の最大値，

$c=\begin{cases}\text{命題が副作用をもつとき} & 1,\\ \text{命題が副作用をもたないとき} & 2^q.\end{cases}$

（q は使用される異なる命題の総数）である[25]．

これは非常に粗い結果であるが，このあたりのことは，まだほとんど調べられていないといってよさそうである．

あとがき

アルゴリズムについては，他にも述べるべきテーマがいろいろある．理論に興味をもつ方は，アルゴリズムの有限性を越えたところ（Degrees of Unsolvability）の分類に関心をもつかもしれないし，応用上の問題としては，グラフの平面性の判定（Tarjan）や，最短経路を求めるアルゴリズム（野下）など，最近の話題にことを欠かない．しかしそれらを網羅することは，筆者の能力も，ページ数の制限をも越えることで，とうていできなかった．ここに述べたことは，少し理論に偏りすぎたかもしれないが，基礎的な『問題』だけはひととおりふれたつもりである．読者の方々に，少しでも参考になるところがあれば，誠に幸いである．

参考文献†

A．一般向きの解説

1) トラハチェンブロート：電子計算機理論の基礎，河野繁雄訳，東京図書（1960）．

2) 駒宮安男：メタコンピュータ，河出書房新社（1970）．

3) 数理科学，特集=アルゴリズム，Vol. 8, No. 9，ダイヤモンド社（1970）．

4) 情報処理ハンドブック，1篇=基礎理論，情報処理学会編，オーム社（1972）．

B．専門向き，基礎的教科書・参考書

5) デーヴィス：計算の理論，赤・渡辺訳，岩波書店（1958）．

6) 相沢輝昭：計算理論の基礎，総合図書（1970）．

7) ホップクロフト，ウルマン：言語理論とオートマトン，木村・野崎訳，サイエンス社（1969）．

8) D. E. Knuth: The Art of the Computer Programming, Addison-Wesley, Vol. 1 (1968), Vol. 2 (1969), Vol. 3 (1972).

C．アルゴリズムの評価について

9) M. Blum, et al.: Computing Medians in linear minimax time, Proc. of the Conference on Combinatorial Algorithms (1972).

10) R. W. Floyd: TREESORT 3, CACM, Vol. 7, pp. 701 (1964).

11) L. R. Ford and S. M. Johnson: A Tournament Problem, American Mathematical Monthly, Vol. 66, pp. 387～389 (1959).

12) J. E. Hopcroft and E. H. Kerr: Some techniques for proving certain simple programs optimal, Proc. of IEEE 10-th symposium on Switching and Automata Theory, pp. 36～45 (1969).

13) D. E. Knuth and E. B. Kaehler: An Enperiment in Optimal Sorting, Information Processing Letters, North-Holland, pp. 173～176 (1972).

14) A. Nozaki: Two Entropies of Generalized Shorting Problem, Journal of Computer & System Sciences to appear.

15) V. Y. Pan: Methods of Computing Values of Polynomials, Russian Mathematical Surveys, Vol. 21, pp. 105～136 (1966).

16) V. Strassen: Gaussian Elimination is not Optimal, Numerische Mathematik, Vol. 13,

† A，B には本文中で引用しなかったものを含む．

No. 5, pp. 354～356 (1969).
17) 高橋秀俊：高速フーリエ変換 (FFT) について，情報処理，Vol. 14, No. 8, pp. 616～622 (1973).
18) M. B. Wells: Applications of a Language for Combinatorial Computing, Proc. of IFIP Congress '65, Vol. 2, pp. 497 (1965).
19) S. Winograd: On the Number of Multiplication Required to Compute Certain Functions, Proc. of NAS, Vol. 58, pp. 1840～1842 (1967).
20) 矢島敬二，恒川純吉：行列の積に関するプログラム技法の比較 TOSBAC REPORT, No. 5, pp. 42～46 (1970).

D. 正当性・停止性・同値性

一般的な解説書としては，近く伊藤貴康氏の著書，また五十嵐滋氏の訳による Manna の著書が出版されるはずである．

21) R. W. Floyd: Assigning Meanings to Programs, Proc. of Symposia in Applied Math., Vol. 19, pp. 19～32 (1967).
22) 伊藤貴康：プログラム理論とその応用，電気学会雑誌，Vol. 91, No. 8, pp. 29～39 (1971).
23) Y. Kanayama: Algebraic Properties of Programs, Proc. of 1st UJCC, pp. 380～383 (1972).
24) J. McCarthy, et al.: Correctness of a Compiler for Arithmetic Expressions, Proc. of Symposia in Applied Math., Vol. 19, pp. 33～41 (1967).
25) A. Nozaki: Relativised Strong Equivalence and its Computational Complexity, 京都大学数理解析研究所講究録 (近刊).
26) Y. I. Yanov: The logical schemes of algorithms, Problems of Cybernetics, No. 1, pp 82～140 (1960).

(昭和48年10月13日受付)