グルシュコフ氏講演：「アルゴリズム代数とマイクロプログラム制御計算機設計」の紹介*

鳥　井　　寛**

グルシュコフ氏が参画した計算機設計の歴史は下記の如くである．

第1時期 (1958-1961) は ПРОМІН'ь (Promin') 型計算機で産業用で互換性もないプロトタイプである．

第2時期 (1962-1965) は Мир-1 (Mir-1) 型計算機で現在数百台使用されている科学技術用数値計算専用機である．

第3時期 (1966-1969) は Мир-2 (Mir-2) 型計算機で，会話形式で動作し，コンパイルは簡単で翻訳した目的プログラムを再入力しない科学技術専用機である．

Mir-1 はプログラム中の精度 (Precision) を4桁から5桁に変更してもプログラムの他の部分には変更は及ばない．また Σ, Π, e, π, ∞, $\int_a^b$, 線形演算，! の記号が使用可能である．

例えば Pr. 10 $\sum_{n=1}^{\infty}\frac{1}{n!}$, Print A where $A=\sum_{n=1}^{\infty}\frac{1}{n!}$ $\frac{1}{n!}<10^{-k}$ で精度の変更は Pr. 10→Pr. 15 で行われる．

Mir-2 は解析言語が使用できる．Mir-1 の言語の拡張になっている．その特徴は

1) 数値計算を分数の形でもできる．

例 $\frac{2}{3}+\frac{1}{6}=\frac{5}{6}$.

2) 主メモリを主ストーレジにしている．

3) マイクロ命令は解析言語を用いて行われる．例として sh, $\int$, $\frac{d}{dx}$ の使用ができる．データリストもまた使用可能である．代数表現式の変形も行う．例えば，$(a^2-b^2)/(a+b)\rightarrow a-b$ を実行する．これらは ROM に恒等式の集合（約300式）をセットしていて，これらを呼び出して行われるのである．恒等式のなかには左辺から右辺と一方向のみに変換されるものと左辺と右辺との両方向に変換されるものがあり，それらによりマイクロプログラムの分岐数が増える．例として $\int e^{ax}\cos bx\,dx=\ldots$ と不定積分ができるし，$\sin\frac{\pi}{2}=1$, $a^0=1$ も数値計算によらないで直接に代入をする．また例として $7\times a\times e^x\times 8=56\times a\times e^x$ とか $7\times a\times e^x\times 8\times a^3=56\times a^4\times e^x$ とかの式計算をする．積分の問題の 85% は計算式で実行できた．難かしい問題はディスプレイ上の式のうち下線を引いて指令で会話モードで式変換を行うことができる．

Mir-2 の記憶容量は 16 kB+4 kB のメモリで他に1M ビット中80万ビットは ROM である．

式変換の内容は次の如くマイクロプログラムが4段階になっていることによって達成された．

マイクロプログラムの4層間は上位が下位に対してそれぞれマイクロプログラムになっている．積分，微分はいくつかのマクロ命令のマイクロプログラムで，基本関数は基本マイクロ演算のマイクロプログラムで表現される（図1）．

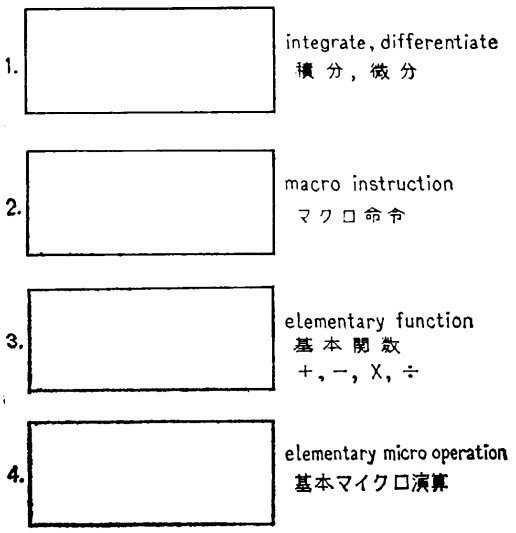

図 1

* Algebra of Algorithms and Microprogrammed Control Computer Design by Виктор Михайдович Гдушков (Viktor Mikhailovich Glushkov: President, Institute of Cybernetics, Kiev), December 8, 1971.

** (株) 日立製作所コンピュータ第一事業部

例 $A=\sqrt{e^{ax}+\sin y}$ で $y=\frac{\pi}{2}$, $a=2$ とおくと

$A=\sqrt{e^{2x}+1}$ と変形を行う.

プログラムの小部分は式の記載の著しい差があり約200 ないし 1000 ステップ位になる.

compiler, translation の選択は次のことによる. 第2世代では Compiler は低速度であったが, Mir-2 では

1) プログラムの最適化によって高速度になり, かつ式変換が可能となった.

2) モードの転換はプログラムの変換を必要とする.

例として $y=\sqrt{a^2+b^2}$ は $y=\sqrt{z}$, $z=u+v$, $u=a^2$, $v=b^2$ に分解してそれらをマイクロプログラムにする. さらに $\cos X=1-\frac{X^2}{2}+\ldots$ を命令で使用できるようにセットする.

1マイクロサイクルに作用するマイクロ命令をうまく設定することが計算機設計の1つのポイントである.

interpretation をうまく設計思想に持込むためには二つの源がある.

1) 経験によるもの. 特に数値数学は経験的なことによって発展される.

2) 実現化の複雑性の考慮. 変換を行うときにどのようなマイクロ演算が実現化されやすいかによって定まる.

マイクロ演算としてはシフト, カウンター, +1, −1 等々をとっている.

要するにマイクロプログラム集合は形式命令 (formal instruction) と 10進, 2進 (decimal, binary) 演算命令とによってアルゴリズム的に完全 (algorithmically complete) であよるうに定義 (definition) を設定することにある.

logical expression (論理式) を下記の如くとる.

主記憶マトリックスを両側 (both side) に無限であると考える. 行はレジスタ (register) とし, 列をレジスタの桁にとる. 2進レジスターで考えるのでは桁はビット (bit) を対応させることになる. それを図で表

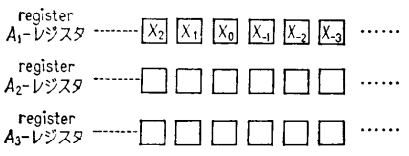

図 2

現すれば第2図の如くなる.

図の A_1, A_2, A_3 等々はレジスター名であり, X_2, X_1, X_0, X_{-1}, X_{-2}, X_{-3} 等々はレジスターの 2^2, 2^1, 2^0, 2^{-1}, 2^{-2}, 2^{-3} 等に対応する桁を表示する.

レジスタの集合を基礎集合にとって各桁間の関数関係を設定したり, 定義する. 例えば1レジスタ上の X_0 は X_0 のある近傍の桁によって次の時点の X_0' が決まる. ($'$ は次の時点を表わす.)

$$X_0'=f(X_0,\ X_1,\ X_{-1}).$$

これは次の X_0' の桁 (ビット) は現在の X_0, X_1, X_{-1} の桁 (ビット) の状態によって決まることを表わしている. X_0 桁 (ビット) の形式で, 他の X_i' の桁 (ビット) は

$$X_i'=f(X_{1+i},\ X_{1+i},\ X_{-1+i})$$

で表わす. この表現を2次元から3次元に拡張したり, 定週期性を導入したりして, 一般的に種々の関数を表現できる. 実際に計算機設計でそれぞれの関数を組込むためには現代の技術程度によって設計の複雑性が決まり, そのために最も簡単なマイクロ命令の集合をとることになる. 例えば

右シフトは, $X_i'=X_{i+1}$ で表わされ, 実際に簡単なマイクロ演算子である.

反転は, $X_i'=\neg X_i$ で表わされ, 比較的簡単な演算子である. (¬は否定記号を表わす) 等々.

このような基本マイクロ演算集合は如何なる他の演算をも完全に代数的表現 (completely algebraic representation) をしうるように選択をする.

そのためにマイクロプログラムは基本マイクロ演算子列で表わされることになる. その手順を以下に述べる. (この手法がアルゴリズム代数である.)

マイクロプログラム形式化のために制御オートマトンA (処理装置をこのように抽象化する) と演算オートマトンB (主記憶装置をこのように抽象化する) との相互作用の型式でとられる. 処理装置Aの入力は主記憶装置の状態であり (状態はビット配置内容を意味している), 出力は演算である. (演算は記憶装置の状

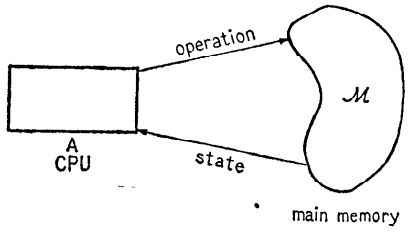

図 3

態を変える.) 主記憶装置の状態集合 $\mathcal{M}$ (すべてのビット配置内容を抽象化したもの) の入力は処理装置Aからの演算であり，出力は状態である.

基本演算 (elementary operation) は $\mathcal{P}: \mathcal{M}\to\mathcal{M}$ として考えられる. 詳言すれば演算 $\mathcal{P}$ は状態集合 $\mathcal{M}$ から $\mathcal{M}$ への写像である. 例えば1レジスタの状態集合だけで考えれば $+1$ という演算は $x\in\mathcal{M}$ に $x+1\in\mathcal{M}$ を対応させる写像である.

論理条件とは記憶状態集合を2元集合 {真，偽} への写像をいう. 例えば1レジスタの内容が零かどうかは論理条件である. レジスタの内容が零であれば，その状態に真を，そうでなければ，偽を対応させる.

i) マイクロ命令 (micro instruction) の集合 $p_1, p_2, \ldots, p_m$ と論理条件 (logical condition) の集合 $\alpha_1, \alpha_2, \ldots, \alpha_k$ を自分達の技術 (実現化の複雑性等々) を考慮して貴方自身で選ぶ.

ii) 最も簡単なハードウェア条件からネットワークの記述をする.

この2条件からマイクロプログラムの基礎が決まる.

記憶状態をレジスタの集合と考えて，ベクトル表示

$$\begin{pmatrix}a_1\\a_2\\a_3\\\vdots\end{pmatrix}=m\in\mathcal{M}$$

として，演算 $\mathcal{P}$ を行えば $\mathcal{P}(m)=m'\in\mathcal{M}$

となると表示する. 具体的には2進ベクトル (binary vector) を信号 (signal) としてとる.

ネットワークは中央処理装置とROMをベクトルで表わしたもので表現する.

マイクロプログラムの集合は各マイクロプログラムの演算の出現の確率より中央処理装置内にレジスターとの転送 (transfer) を考慮して固定化 (fix) する.

例えば，$\sqrt{\ }$, $\int$ 等々を固定化する. (図4)

レジスターの最終結果を3レジスタの場合には $\begin{pmatrix}X\\Y\\Z\end{pmatrix}$

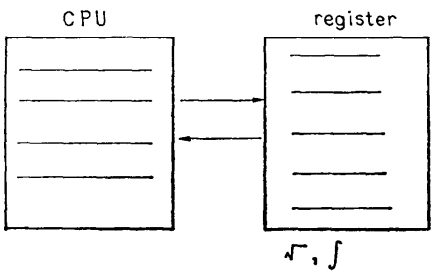

図 4

で表わす. ハードウェアでは $\alpha: \mathcal{M}\to(0, 1)$ は論理条件である.

それらを計算機援用設計(Computer-Aided design)，マイクロプログラムの圧縮(compression of microprogram)を1ステップごとの改良(improve step by step)とオートマトンの簡約化 (minimization of automata) を用いて自動的に処理をする (automatic procedure). 問題の集合 (set of problem) を最終の仕事 (final task) と考えて設計する.

$\mathcal{P}: \mathcal{M}\to\mathcal{M}$ として例えば2レジスタ上の乗算は

$$\begin{matrix}a\text{ レジスタ}:\\ b\text{ レジスタ}:\end{matrix}\begin{pmatrix}a\\b\end{pmatrix}\to\begin{pmatrix}a\times b\\b\end{pmatrix}$$

で表わされて実現できる.

例として e^x も制約された技術 (restricted technology) で実現できるものに入る.

目的は形式変換が容易なものを使用する.

$\beta: \mathcal{M}\to(0, 1)$ を他の論理条件とする.

そうして以下の如き演算を定義する.

$\mathcal{P}$ と Q が二演算子ならば，$\mathcal{P}\cdot Q$ は積 (multiplication) または合成 (superposition) というものが定義される.

例 $\mathcal{P}: x\to e^x$ で $Q: x\to x^2$ のとき

$\mathcal{P}\cdot Q=(e^x)^2=e^{2x}$ および $Q\cdot\mathcal{P}=e^{x^2}$ である.

論理条件 α と演算 $\mathcal{P}$ と Q について選言 (disjunction) が $\alpha(m)=1$ のとき演算 $\mathcal{P}$ を行い，$\alpha(m)=0$ のとき演算 Q を行う演算として定義されて $(\mathcal{P}\underset{\alpha}{\vee}Q)$ と書く. (図5)

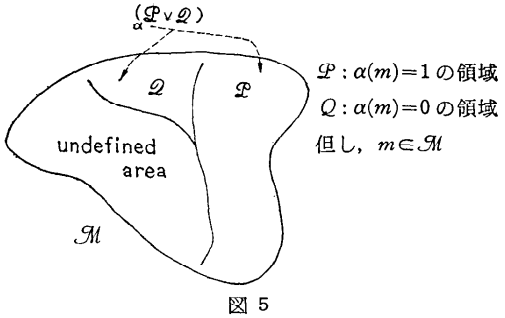

図 5

(補足すれば状態 m が論理条件 α を充たすとき ($\alpha(m)=1$)，選言記号の左側の演算を行い，充たさないとき ($\alpha(m)=0$)，右側の演算を行う. また α はすべての元 $m\in\mathcal{M}$ については必ずしも定義されていない.)

さらに論理条件 α と演算 $\mathcal{P}$ について反復 (iteration) が次の如く定義される新演算である.

$m\in\mathcal{M}$ について，$\alpha(m)$, $\alpha(\mathcal{P}(m))$, $\alpha(\mathcal{P}^2(m)), \ldots$

が定義されていて，そのうちで始めて α が成立する元を対応させる演算である．

以上の3演算が演算集合に定義されるのに対して，論理条件については従来通りの論理演算 $\alpha\vee\beta$，$\alpha\vee\beta$ と $\bar{\alpha}$（選言，連言，否定）が定義されているが，新たに条件 α と演算との乗法演算は次の如く定義される．演算 $\mathcal{P}$ を実行して後に，条件 α が成立するかどうかの検証をする論理条件と同値な新条件 β を生成する演算である．これを記号 $\mathcal{P}\alpha=\beta$ で表わす．論理条件は集合 $\mathcal{M}$ で部分的に定義されていることを注意しておく．

例えば $\alpha=\alpha_i\vee\underset{\alpha_2}{\{p_1\}}\alpha_3$ は論理条件であり，$\mathcal{P}=\underset{\beta_2\alpha_1}{(\mathcal{P}_2}\mathcal{P}_1\vee\underset{\alpha_2}{\{\mathcal{P}_3\}})$ は演算である．

このように基本演算については演算子代数が，論理条件については条件代数が成立すれば，一般の代数と同様に高等学校程度の式の変形と同様なことが行われる．

$\mathcal{P}\cdot Q$ と $\underset{\alpha}{(\mathcal{P}\vee Q)}$ と $\underset{\alpha}{\{\mathcal{P}\}}$ のみで表現されるものは正則 (regular) といわれるが，マイクロプログラムは変形を行えば正則なマイクロプログラムになる．

別の記法では $\overset{\downarrow}{\overline{\ p_1p_2\alpha\ }}\ \overset{\downarrow}{\overline{\ \beta_3\ }}$ と書いたものを上記の如く改良した．(jump on condition で yes ならば → の方向へ，no ならば ← の方向へ行くことを表わしていた．)

式の変形を行う場合には変換関係式を設定する必要がある．それには2種類ある．

第1は定義関係式である．例えば $a(b+c)=ab+ac$ の如き代数表現がこれに属する．一般代数でもアルゴリズム代数でも成立する恒等関係式である．

第2は与えられたマイクロプログラムについての基本演算と論理条件についてのみ成立する定義関係式である．これは具体的なハードウェアに定まる特殊関係式であって一般代数では成立しないものが多い．

例えば，2レジスタ a，b について s を第1レジスタの内容を第2レジスタの内容に加算する基本演算を l を第1レジスタの内容を左シフトする基本演算とすれば

$$s:\begin{pmatrix}a\\b\end{pmatrix}\to\begin{pmatrix}a\\a+b\end{pmatrix}\text{ で }l:\begin{pmatrix}a\\b\end{pmatrix}\to\begin{pmatrix}2a\\b\end{pmatrix}$$

と表示されて $s^2l=ls$ となり非可換関係式が成立つ．

$$\left[\begin{pmatrix}a\\b\end{pmatrix}\overset{s}{\to}\begin{pmatrix}a\\a+b\end{pmatrix}\overset{s}{\to}\begin{pmatrix}a\\2a+b\end{pmatrix}\overset{l}{\to}\begin{pmatrix}2a\\2a+b\end{pmatrix},\right.$$

$$\left.\begin{pmatrix}a\\b\end{pmatrix}\overset{l}{\to}\begin{pmatrix}2a\\b\end{pmatrix}\overset{s}{\to}\begin{pmatrix}2a\\2a+b\end{pmatrix}\right]$$

となるからである．この式を複雑度 (complexity) 5 の因子 (factor) を持つという．

これらの関係式のネットワークをとり簡単にする．ブール関数をアルゴリズム的に可解なものにする．

$p_1p_2p_3=p_6p_5$ という変換関係式では左辺から右辺に移る場合は因子の個数が3から2へと減少するが，これはマイクロプログラムを簡約化する方向に相当し，右辺から左辺に向う変換はマイクロプログラム実行の高速化に向う方向になる場合が多い．

その他に正則表現で成立する関係式も含まれる．

例えば

$$\underset{\lambda}{\{\mathcal{P}\}}=\underset{\mu}{(e\vee\mathcal{P})}\underset{\lambda}{\{\mathcal{P}^2\}}\qquad\text{ただし }\mu=\underset{\lambda\vee\lambda\mathcal{P}}{\{\mathcal{P}^2\}}\lambda$$

の如きもので，$\mathcal{P}$ は任意の演算で λ は任意の条件であり，e は恒等演算である．

例：3レジスタ上に於ける乗算は下記の如く記述できる．第1レジスタと第3レジスタに被乗数と乗数が置数され，乗算の結果として第2レジスタに積が置数されるとする．求める乗算のマイクロプログラムを

$$\mathcal{P}:\begin{pmatrix}a\\b\\c\end{pmatrix}\to\begin{pmatrix}0\\a\times c\\0\end{pmatrix}$$

とすれば，$\mathcal{P}$ は下記のマイクロ演算を使用して表現できる．ただし

$$O_1:\begin{pmatrix}a\\b\\c\end{pmatrix}\to\begin{pmatrix}0\\b\\c\end{pmatrix}\qquad 0_1\text{ は第1レジスタのクリア演算：}$$

$$s_{12}:\begin{pmatrix}a\\b\\c\end{pmatrix}\to\begin{pmatrix}a\\a+b\\c\end{pmatrix}\qquad s_{12}\text{ は第1レジスタの内容を第2レジスタの内容への加算；}$$

$$p_3^{-1}:\begin{pmatrix}a\\b\\c\end{pmatrix}\to\begin{pmatrix}a\\b\\c-1\end{pmatrix}\qquad p_3^{-1}\text{ は第3レジスタの内容から1を減算；}$$

$\alpha_3\equiv(c=0)$　α_3 は第3レジスタの内容は零に等しい論理条件；

$$\mathcal{P}=O_2\underset{\alpha_3}{\{s_{12}p_3^{-1}\}}O_1O_3$$

と表わされる．

この意味は第2レジスタの内容を破算して第1レジスタの内容を第2レジスタの内容に加算してその度ごとに乗数の内容（第3レジスタの内容）から1を減算をすることを何回続けて α_3 即ち第3レジスタの内容が零になった時に第1レジスタと第3レジスタの内容

を破算することを表わす.

$$\mathcal{P}=O_2\{s_{12}p_3^{-1}\}O_1O_3-\underset{\alpha_3}{O_2}\{s_{12}p_3^{-1}\}lO_1O_3=\ldots$$

($l_1O_1=O_1$ が成立する事を用いて変形する.)

(註. 以下変換が続く所で時間切れになり講演が終わる. グルシュコフの論文では最終的に

$$\mathcal{P}=\underset{\alpha_3\beta_3}{O_2}\{(e\vee s_{12}p_3^{-1})l_1\gamma_3\}O_1O_2$$

(但し β_3 は第3レジスターの 2^0 ビットが零である論理条件である.)

となることを導き, 反復の内部にある演算が最初の式に比べて半減し, 速度が2倍になったことを述べている. ただし恒等変換式をかなり挙げている. $\mathcal{P}$の式は講演時の内容を論文によって補った.)

質疑応答

問 後藤氏: 公理集合, 言語集合, 五十嵐氏の論文, LISP との関連如何.

答 グ氏: われわれの方が便利である. 最適変換をアルゴリズム的に実行できるように選んだ.

問 渋谷氏: Mir-2 は比較的小さな計算機なのになぜ数式処理の機能を備えているのか.

答 グ氏: 技術者の間で数式処理が便利とみなされているから.

あとがき

グルシュコフ氏はウクライナ連邦共和国科学アカデミーのサイバネティック研究所長である. ここはソ連邦有数の情報処理システムの研究機関であり, 最近京大の坂井教授と日科技連の矢島氏が訪問されて, ミール-2 <Мир-2> 型計算機を見学されミニコンでありながらインタープリターシステムをハードウェアとして内蔵して居りソフトのハード化として注目された. その設計の中心はマイクロプログラムの高度の応用とそれを技法的に解決したアルゴリズム代数 (旧名マイクロプログラム代数) である. それが本講演のテーマであったが筆者の実力不足でまとまりのない断片的要約になったことを御断り致します. なお念のため関係の論文, 図書名を参考のためあげておく.

参考文献

1) デイジタルオートマトンの合成, モスクワ, 1962, pp. 1-476.
2) 翻訳生成システムのある計算機, キエフ, 1970, pp. 1-259. (共著である.)
3) オートマトン理論とディジタル機械の構造設計問題 (雑誌キベルネテイカ, No.1, 1965, pp. 3-11, キエフ.)
4) オートマトン理論とマイクロプログラムの形式変換 (雑誌キベルネテイカ, No. 5, 1965, pp 1-9, キエフ.)

(昭和47年3月18日 受付)