技報

UNIVAC TECHNOLOGY REVIEW

1981年8月　第1号

論　説

3次元磁場解析への有限要素法の適用
——ベクトル・ポテンシャルの取扱い……渡部義維　89

Kolmogorov-Chaitinの計算論的情報量と
Lempel-Ziv万能データ圧縮算法……山田真市　74

論　文

VLSIで実現した汎用コンピュータの耐故障性……
R. M. Sedmak, H. L. Liebergot　1

OAのためのワークフロー・コントロール・モデル
……L. S. Baumann, R. D. Coop　63

報　告

MACRO：プログラム言語……S. R. Greenwood　16

高水準会話型デバッギング・システム AIDS……J. J. Hart　33

汎用構文解析プログラム GSA……T. N. Turba　42

TECHNOLOGY TREND……100

BOOKS……107

CALENDAR……109

MEMORANDUM……112

EDITORS' NOTE……114

日本ユニバック

柔軟な思考力・創造的な設計力を培うために　〈内容見本進呈〉

岩波講座 情報科学

全24巻

A5判・各巻平均230頁

編集委員　相磯秀夫　伊理正夫　高橋秀俊
長尾 真　森口繁一　米田信夫

マイコンや新しいプログラム言語，アルコリズムの開発など，最近の情報処理の理論と技術の進歩はめざましく，しかも広範囲にわたっている．本講座は，こうした情報科学・情報工学の現状を綿密に検討し，新しい学問体系の樹立とそれを基礎にした理想的な教育カリキュラムの編成をめざしたものである．基礎的な事項に重点をおき，考えの筋道をわかりやすくイメージ豊かに説きあかすとともに，実践的な例題やプログラム例を豊富に配し，情報処理技術の急速な進歩に対応する柔軟な思考力と創造的な設計力を培う．〈予約出版〉詳細は内容見本をご覧ください．

本講座の特色

- ▶情報科学・情報工学を固有の論理と方法をもつ構造として体系化し，編成した新しいカリキュラム．
- ▶基礎的な事項に重点を置き，考えの筋道をていねいに，わかりやすく解説する．
- ▶抽象的な概念や方法を，イメージ豊かに，十分に納得がいくよう説きあかす．
- ▶例題，算法，具体的なプログラム例，図・表などを豊富に配し，学生・技術者・研究者の実践的な手引となるようにした．
- ▶マイクロコンピュータについては，各所で必要かつ十分な解説を与える．
- ▶パターン認識や人工知能，第5世代計算機などの進んだテーマについての講義もバランス良く含めた．

本講座の構成

入門
1 情報科学の歩み
2 電子計算機への手引き
3 プログラムの読み方

表現
4 情報と符号の理論
5 情報ネットワークの理論
6 オートマトン・形式言語理論と計算論
7 論理と意味
8 情報の構造とデータ・ベース

処理
9 プログラム言語
10 基本的算法
11 データ管理算法
12 算法表現論

機構
13 順序機械
14 計算機の機能と構造
15 計算機アーキテクチャ
16 オペレーティング・システムの機能と構成

数理
17 離散数学
18 数値計算
19 最適化
20 信号処理とシステム制御

特論
21 パターン認識と図形処理
22 人工知能
23 数と式と文の処理
24 生体における情報処理

第一回配本 第3巻 プログラムの読み方 240頁 定価2300円 ——10月9日発売！

岩波書店
東京・千代田・一ツ橋
振替〈東京〉6-26240

論文 VLSI で実現した汎用コンピュータの耐故障性

Fault Tolerance of a General Purpose Computer Implemented by Very Large Scale Integration

R. M. Sedmak, H. L. Liebergot

要 約 VLSI による集積回路(IC)論理要素——すなわち何千ものゲート——によるコンピュータ・システムの構築の傾向は必然的な成り行きである。すでに，記憶素子内では，このレベルの回路がとり入れられている．汎用コンピュータの VLSI 化において，従来の高信頼性技術では問題点があり，新しいアプローチを研究しなければならない．

高水準な障害の検出，回復，障害の分離は，従来の VLSI を使用した半導体技術では実現が困難であった．本稿は，この困難を克服する VLSI 素子を用いた汎用コンピュータの設計上の研究努力の一端を述べている．基本的なアプローチは，効果的な障害検出を VLSI チップ内において行うという斬新な方式により，論理回路を設計し分割するものである．

このアプローチにより，きわめて高いレベルの障害検出を得ることができ，他の設計法を結びつけることにより，ほとんどの間欠的障害と多くの固定的障害に対して耐故障性を持たせることができる．故障の分離がきわめて正確に行われるので，障害の原因となっている交換すべき単位を決めるために，診断テストまたはその他の保守作業が不必要となる．

また，本稿では汎用コンピュータの開発に際して考慮しなければならない設計上のトレード・オフに関する理論についても研究している．

Abstract The construction of computer systems containing integrated circuit logic components with very large scale integration (VLSI), that is, many thousands of gates, is inevitable. Such levels of integration have already been achieved in memory components. There are significant problems in using some conventional fault-tolerant techniques in VLSI implementations for general purpose computers; consequently, modified approaches must be investigated.

This paper describes preliminary results of a research effort to design a general purpose computer with VLSI components which will achieve a level of fault detection, recovery, and failure isolation far exceeding non-VLSI implementations. The fundamental approach is to design and partition the logical elements in such a way that effective fault detection can be done by a novel method which places the detection responsibility inside the VLSI chip.

This approach results in an extremely high level of fault detection, and combined with certain other design techniques, also results in tolerance of most transient and many solid failures. Failure isolation is sufficiently exact that is unlikely that any diagnostic test or other maintenance action will be necessary to define the failing replaceable unit.

The rationale for the design tradeoffs which must be made in the development of a general purpose computer is also explored.

1. はじめに

本稿の背景に関係して，過去に重要な研究がいくつかある．まず，W. Carter ら[1]は1964年に，障害が間欠的に起こるためにシステムが長期にわたってサービスされない状態が発生すること，またこの問題を克服するには障害を速かに検出することが非常に重要で

あることを示した．W. Carter と P. Schneider[2] は，1968年に，自己検査回路の基本原理について論じた．1972年に M. A. Mehta ら[4]は，内部冗長性について提案したが，要求される冗長性が極端である(単一回路の2倍以上)ことと，この技法が単なる内部障害の診断だけを可能にしただけであったために採用されなかった．彼らは，出力検査回路を通して入力の誤りの診断を簡単にするツール (Ambiguity Resolver) を使って，よりよいアプローチを行うことができるであろうと示唆した．J. F. Arnold[5] は1972年に，高度な信頼性を得るために，機械内のすべての回路に対して，障害の検出と回復を行うことの重要性を示した．

N. Tanaka ら[7]は1977年に，汎用コンピュータ ACOS 800/900 のいくつかの部分における二重化について述べた．最後に FTC 7 で，W. Carter ら[8]が述べている次の結論は注目に値する．すなわち，IBM 360 の LSI 版を完全に検査するのに必要な回路は(普通の IBM 360 の障害検出と比較して) 6.5 パーセント増加するだけであること，マイクロ命令と単純なマクロ命令の再試行はチップを1個追加するだけで実行でき，この際，通常の IMB 360 の性能に比べ，はっきりした速度の縮退もないことである．

2. 背　　景

本稿は，信頼性の高い中・大規模性能のコンピュータを安い費用で実現するための研究の延長線上にある．すなわち，UNIVAC の新しいコンピュータの研究において考えられる半導体技術は ECL LSI であり，部分的に SSI と MSI を使用するものであったが，本稿での研究は，同じ設計のコンピュータを 5,000 ゲート/チップ以上の VLSI を用いて，価格と耐故障性を大幅に改善することに全力を注いだ．この際に現存の回路技術を使う想定であるが，必要なゲート密度と伝播の遅延が可能になるように，将来の VLSI 処理パラメタと結びつけて考えた．また，高密度が達成可能であり，歩留りと製造価格が許容範囲であることを前提とした．残された主要な問題は，耐故障性の高い汎用コンピュータの設計において，各チップ内に組み込まれた何千ものゲートをいかに効果的に使用することができるかということであった．

設計過程のほとんどの場合と同様であるが，次の3つの基本要因が考慮され，互いに平衡をとる必要がある．それらの基本要因とは，価格，性能および耐故障性である．従来，システム・エンジニアは，2つの要因を固定したままで，他の1つの要因を大幅に改良しようとしても，そのいずれかの要因が大幅に犠牲になることを経験してきた．しかし，VLSI の出現はこの状況を大きく変えたと思われる．設計に組み込まれるゲート数の増加は，製造価値の上昇に直結していた．VLSI の場合はこれと異なり，ゲートの追加による価格上昇の割合は非常に小さくなった．したがって，システム設計者が性能を一定にしたまま(あるいは性能を向上させてまで)回路素子の追加によって耐故障性を増そうとした場合，機械の製造価格の実質的増加は，従来の設計の価格と比較して，非常に小さいであろう．

2.1 耐故障性の外観

商業用汎用コンピュータに関して，耐故障性の設計に影響する市場パラメタは，次に示すとおりである．

1) TMR (triple modular redundancy) によって障害を閉じ込める方法は，次の場合には必要がない．すなわち，他の技術や手法によって，システムの停止時間の全使用時間に対する割合を小さくすることができるときである．

2) かりに中央処理装置群に障害がない (failure-free) ように設計されたとした場合で

も，機械的な周辺装置の障害に対してユーザから保守要求がなされる．したがって，とくに中央処理の設計において，保守のための緊急呼出しを回避するために“延期保守 (deferrable maintenance)”の機能を備えることが望ましい．しかし，自己修復機能を備える必要は必ずしもない．

3) 100パーセント近くの即時障害検出が必要である．障害を察知して，誤った出力が生成されないようにし，障害回復を助け，障害分離を行うために障害の検出がなされなければならない．

4) 一時的または間欠的障害から，完全な回復がなされなければならない．これは，そのような障害の分離は非常にむずかしいこと，および，回復が基本的に瞬時の冗長度(再試行)によって可能であり，したがって相対的に経済的であることの2つに起因する．

5) 保守時間の最大部分を占めるのは，通常，診断である．したがって操作員の介在なしに即時障害分離が可能であれば，故障時間を短縮することができる．

ここで提案する VLSI 機械の耐故障性は，上記パラメタを満たすように設計されている．

3. 中央処理装置群の一般構造

本稿で記述する処理装置の構造は，仮想記憶装置つきの中・大規模の UNIVAC 90 シリーズ中央処理装置を基にしており，マクロ命令の先取りを使うことなく，単一のマクロ流れの実行を行う．バイト形式およびワード形式データと同様に，可変長命令も許される．機械は，マイクロプログラム化されており，したがって，命令実行を加速するために採用されているランダム制御論理はほんのわずかである．この特性は，VLSI を採用したためではなく，基本的に，最初の版の設計の結果であり，性能上からわずかハードウェア化を増大したとしても経済的に十分見合うものとされる．

命令処理装置内では，シフト機構，母線，レジスタ・スタック等，資源の高度利用が行われている．さらに，マイクロ・サイクルの4個のクロック・フェーズの間に，多量のハードウェアが同時に効率的に実行される．入出力は，中央処理装置に組み込まれたバイト多重チャネル・インタフェースを通して処理される．入出力アクティビティは，かなりの量の能力が個々のチャネル内にあるにもかかわらず，割込み駆動型マイクロ・コードを通じて扱われる．

MOS 回路技術は VLSI に向いていると考えられているが，他の高速度高密度技術も使うことができると考えられる．耐故障性に関する主な改良で，それほど価格を上昇させずに大きな利益が得られるのは，ゲートの高密度のためである． 実装方法は VLSI 回路の動作に関連し，結合パラメタと多層プリント配線基板および背面パネルに特徴づけられる．

コンピュータ設計において，最も重要な耐故障性は，障害の検出であり，これは耐故障性全体の基礎である．検出可能範囲は IC チップの故障だけでなく，一時的な電源不安定やクロックの故障，および背面ボードのショートやホイルのオープンやケーブルのゆるみ等の機械的結合上の障害も含むことに注意しなければならない．こうして，この機械の基本設計基準は100パーセント近くの単一障害検出率である．これらの詳細を以下に示す．

4. 耐故障性の詳細

障害の検出，分離，および回復について，前述の設計目的に合わせるため，重要な技法がいくつも使われてきた．これらの方式を次に示し，順に説明する．

1) “相補論理” を使った内部冗長性
2) 誤り処理チップ
3) チップ間回路ネットワークの検査のための，誤り検出/修正コードと冗長性
4) 誤り検出回路の自己検査
5) チップ外部の検査

4.1 汎用 VLSI チップ

図1は，本論文で提案している設計の典型的な VLSI チップの一般構造である．論理の大部分は，“機能（論理）回路” と呼ばれる部分と，ほぼ同じ大きさの “重複相補論理 (duplicate complementary logic)” と呼ばれる部分からなる．これらの論理ブロックの出力は，各種中間結果と同様に，比較器1から n までに与えられる．出力データと制御情報は，コード（ECC またはパリティ）とともに出力ピンに送られる．

機能回路と重複回路は，入力ピンから入力データと制御情報を受け取る．この情報は入力コード検査器 (input code checkers) 1から m までが検査して，入力データが正しいかどうかを調べる．電源とグランド入力は冗長性があり，機能論理と重複論理のそれぞれに与えられる．冗長な電源入力は比較器により検査される．クロック入力は，もう1つの検査器と論理ブロックに与えられる．

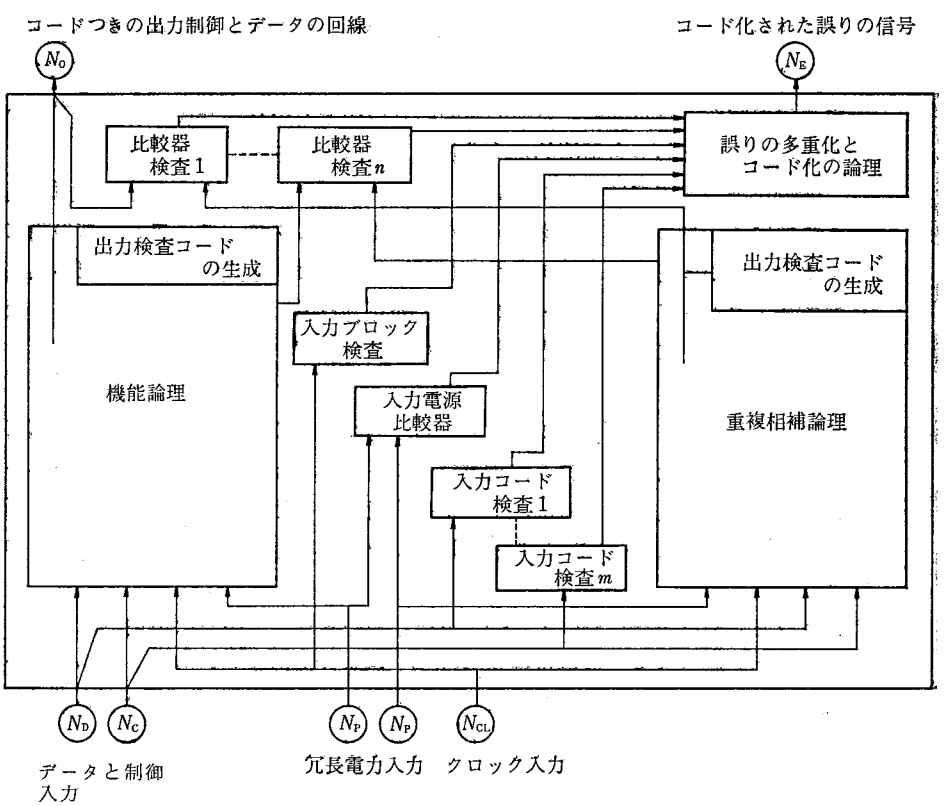


図 1 汎用 VLSI チップ
Fig. 1 Generalized VLSI chip

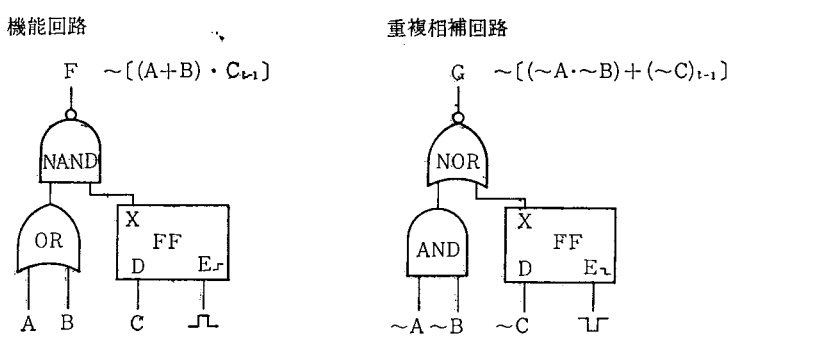
次に，検査器の出力はすべて，誤り符号化論理 (error multiplexing encoding logic) に送られ，他の出力ピンに出される．

4.2 障害検出と処理技法

4.2.1 相補論理を使った内部冗長性

図2に，VLSI チップ内で使用する相補重複の一例を示す．機能部分内のある1つの組合せ論理素子に対し，ゲートに入る信号 (A, B, C) とゲートから出る信号 (F) は，相補部分のそれらとは反対の極性を示すことが，この図から読み取れる．順序論理とともに，同じ物理的素子が使われる．ただし，制御信号と機能論理内に蓄えられるデータは，相補論理内のこれらの極性とは逆の極性を持つ．この技法は次の2つの特徴を持つ．第1に，同じマスクまたはセル型を内部的に2度，すなわち機能論理に1回，次に重複回路に1回適用することによって起きる問題を取り除くことができる．この問題とは，マスクまたはセルの障害が機能回路と重複回路の両方に現れて同じ障害状態がつくりだされ，比較器では検出できない障害(設計，過程，および磨耗)が起こりうることである．第2に，相補重複を持つ設計は，単なる重複に比べてはるかに橋絡型の障害は検出しやすい．これは，同じ論理関数を実行するのに必要な論理配線において同一極性で同じ関数を同時に実行するネットワークがはるかに少ないという事実，すなわち，橋絡したため検出されない誤りになったとしても，それ以降の回路で障害が必ず検出されることによる．

1から4までの4個の比較器を，4個のマシン・クロック・フェーズのそれぞれと結びつけた形で，各チップ内部に持つことが望ましいことが，論理回路の分割過程でわかってきた．各比較器は，それぞれのフェーズの間，安定している信号を検査する．この結論は次の事実から得られたものである．すなわち，内部論理の密度が高いために遅れてくるチ



入力			出力	
A_t	B_t	C_{t-1}	F_t	G_t
L	L	L	H	L
L	L	H	H	L
L	H	L	H	L
L	H	H	L	H
H	L	L	H	L
H	L	H	L	H
H	H	L	H	L
H	H	H	L	H

キー

記号	意味
～	NOT
•	論理積
＋	論理和
添字 t	時間 t
添字 $t-1$	時間 $t-1$
E↑	クロックの低から高への変移
E↓	クロックの高から低への変移

図 2　機能回路と重複相補回路の対比
Fig. 2　Example of functional versus duplicate complementary circuits

ップ出力が比較できるようになるまで待つのでなく，中間データまたは制御結果の比較が，しばしば必要になる．たとえば，あるマイクロ命令中に，VLSI チップに入れられたレジスタ・ファイルを書き込み，その後でしばらくマイクロ命令(またはマクロ命令)がその内容を読み出さない場合を仮定する．チップの最終出力でのみ比較を行う場合，レジスタを正しく書く際の障害はずっと後で読取りを行うときまで検出されない．これは，その間に再試行に必要なデータが放棄されている可能性があり，再試行手続きができないかもしれないからである．ただし，更新レジスタの比較等のように中間結果の比較を行う場合には，再試行が可能なくらい十分な早さで障害検出ができる．同じ理由により，チップの外部的重複とそれらの比較による方法も，VLSI の実現により実際的になり有効になることは明らかである．

4.2.2 誤り処理チップ (EHC)

中央処理装置内の誤り信号はすべて，誤り処理チップ EHC (error handling chip) といわれる VLSI チップ（この設計では誤り信号の個数が十分に低ければ1つで十分である）に送られる．この論理回路内で，障害の組合せ，分類，および分離が行われる．図3に示すように EHC は，図1で示した内部重複と障害検出を含む各種技法を使っている．ただし，この論理は精密なので，EHC はプロセッサおよびコンソール・クロックを受け取り(コンソールは独立のインテリジェント・マイクロ・プロセッサを備えている)，どちらかの障害を内部で検出し，正常な方のクロックに自動的に切り換える．したがって，CPU ク

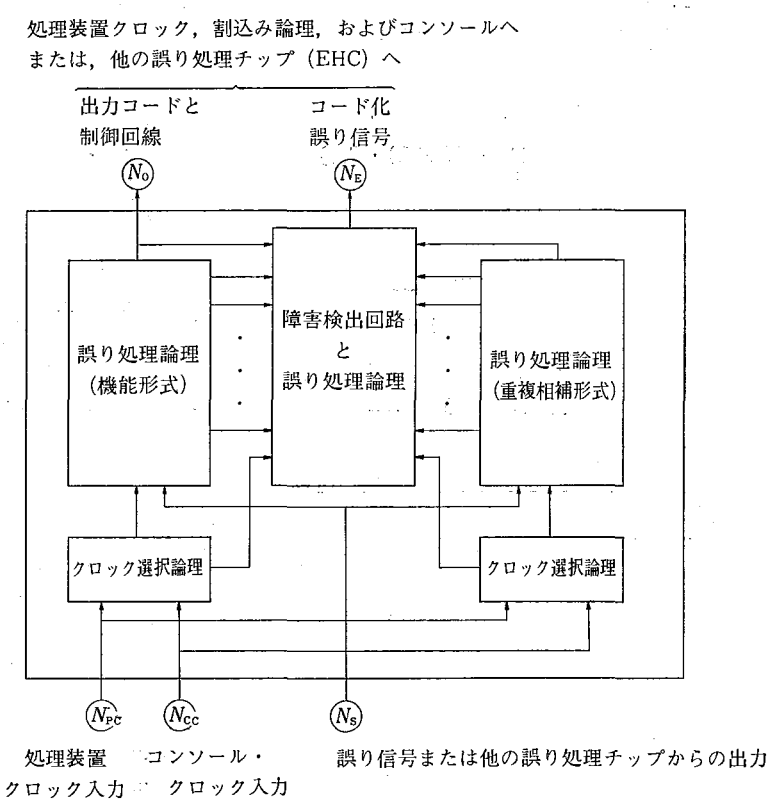


図 3 誤り処理チップ (EHC)
Eig. 3 Error handling chip

ロックの障害が起きても，正しく障害の状態が伝えられる．

複数個の EHC を使う場合，木構造による EHC の結合が使われる．各種 VLSI チップからのコード化された誤り信号はすべて，EHC の第1レベルに入る．木の各レベルで前処理が行われ，情報を圧縮して(冗長部の消去，不適切なデータの除去等)，入力誤りコードをビット数のより少ない1つのコードに変換する．各レベルでの前処理の後，マイクロ・コードの誤り割込みを実行するために，EHC の木の最終出力は，マイクロ・プログラム制御ストア・アドレスの指定に使用される．これらの出力はマスタ・クロック生成と独立コンソールにも送られ，それぞれ順序制御と，最終的障害処理，およびその後の誤り状態の表示に使われる．EHC 内の内部検査方式によって，これらのチップ内の障害もコンソールに送られて表示される．EHC は頻繁には使用されないので誤り状態に反応する論理回路を含み，障害注入 (fault injection) 能力を備えている必要があろう．

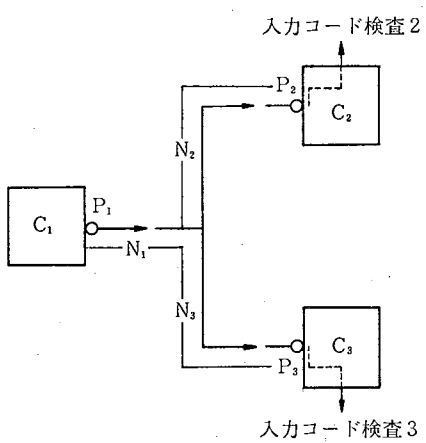
4.2.3 チップ間ネットの検査のための誤り検出/修正コードおよび冗長性

ここで，また図1を参照しよう．チップ間のすべてのデータと制御回線に対して誤り検査コードが付加され，単一または特殊信号線（クリア回線等）の場合には冗長性が使われる．利用される基本コードは，単純パリティか，またはタイミングとピンが許す場合には ECC コードである．

この研究で扱っているアプローチの重要な特徴は，各 VLSI チップ内に2個以上の障害検出回路があり，多くの場合，同じネット内に数個のチップがあって同じ出力を監視していることである．膨大な量の障害情報が得られるため，EHC での前処理およびコンソール論理内の最終処理においてかなりの障害の分離が実現できる．数個の明確な誤り検査回路から出力を監視することによる障害分離の簡単な例を，図4に図示した．この図表は，2個の入力コード検査の組合せによって，障害の位置が特定できることを示す．すなわち，2個のネット分岐のうちの1つとその受取りチップ，またはネットの分岐前の部分とその関連する出力ピンの3つのケースに分離することができる．

4.2.4 自己検査の検査器

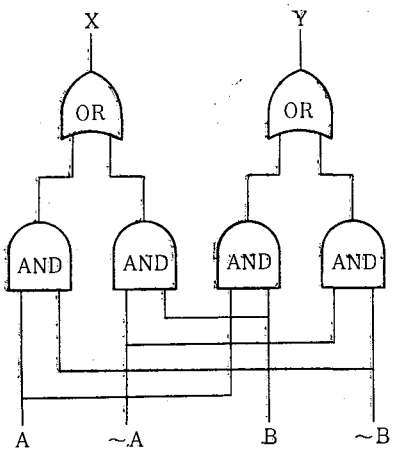
コード検出器と比較器とは，自己検査設計アプローチを使ってつくられている．したがって，各検出回路は障害の危険がなく，単一障害に対して自己検査ができるので，検査の



入力コード検査2？	入力コード検査3？	単一障害位置
NO	NO	なし
NO	YES	N_3, P_3, C_3
YES	NO	N_2, P_2, C_2
YES	YES	N_1, P_1

注 1) 'C'：チップ．'N'：ネット（信号のセット）．'p'：ピン（ピンのセット）．
注 2) C_1 内で，チップ内障害は検出されなかったと仮定する．

図 4 複数分岐ネットにおける単一障害の分離
Fig. 4 Single failure isolation in multiple branch net



出力		意　味
X	Y	
0	0	比較の誤りまたは検査器の障害
0	1	誤りでない（比較）
1	0	誤りでない（比較）
1	1	比較の誤りまたは検査器の障害

注 1) A, Bは機能論理からの2回線，～A, ～Bは重複相補論理からの2回線.
注 2) Aは～Aと，Bは～Bと対応する.

図 5　自己検査比較器：基本的組立ブロック回路
回路が2ビットの比較器として使われる場合
Fig. 5　Self-checking comparator : Possible basic building block circuit

ために特別な診断手続きを設ける必要はない．図5は，自己検査比較器の構成に使うことのできる回路の一例（D. A. Anderson[3] に記述）を示している．この設計は1つのアプローチのみを示しているにすぎない．現在，検査器内部および外部の単一障害を区別することのできる新しい回路も研究中であり，よりよい分離能力を持つものが今後できるであろう．図5の回路は，2ビットの比較器として使われる場合，出力が同じときには検査器に障害があることを示し，出力が異なるときには障害がないことを示している．この回路の優れた点は，障害のない入力が変化するにつれて(1, 0)から(0, 1)へと変わり，検査器中での障害の検出を促進する点である．検査器のない論理はすべて冗長であり，検査器論理はすべて自己検査的である．したがって内部チップでは，すべての単一障害と多くの多重障害が完全に検出される．

図1に示すように，各種誤り検査器はすべて，EHC への送信のための信号を多重化し，コード化する論理回路に送られる．この論理回路もコード化された誤り信号と同じく，自己検査的である．コード化アプローチの目的は，誤り信号に必要な出力ピンの個数を減らすことである．

4.2.5 “チップの外側”の検査

障害検出の設計を完全にするため，ここではとくに，電源とクロックの生成と分配の領域についての扱いに言及する．電源の完全な冗長性は設計上の目的でないので，各種の電力供給の1つだけを使う．

適当な検出器が電源の出力を監視し，障害を検出する．電源装置の出力は多重背面パネル・パス，PC ボードのピン，PC ボード・パス，および VLSI チップのピンを通して，各 VLSI チップに供給される．これにより，個々の分配された電力はピンの信号として VLSI チップ内部で比較され，電源の障害が検出される．この技法によると，非常にめんどうな障害(電力分配ネット)をかなり厳密に分離することができる．

4フェーズ・クロックが，ほとんどすべての VLSI チップに分配される．生成または分配中の障害は，クロックを受ける各チップにおいてマスタ・オシレータのパルスを送り，チップの各々で局所的な検出回路を組み込むことによって，検出される．下記の型の障害

は，クリスタル・オシレータのバッファ化出力により動かされる回路で検出される．

1) オシレータの欠陥
2) 単数または複数フェーズの欠落
3) 許容フェーズ(単数または複数)からの極端な逸脱
4) フェーズ(単数または複数)の順番外れ

クロック障害の検出のための VLSI チップをすべて監視することにより，EHC は障害を，クロック生成器，分配ネットとピン，または受取りチップに分離させることができる．

4.2.6 検査計画の考察

4.2.5 項で説明した内部検査方式に対して，様々な利点と欠点をあげることができる．まず，利点を以下に示す．

1) すべての単一障害，および多くの多重一時的または固定的障害が検出できる．
2) 多くの橋絡または短絡した障害が検出できる．
3) チップ内部の 2 個の異なった極性の信号を用いた二重化によって，多くのタイミング上の問題と競合状態で起きる障害が発見できる．

これとは反対に，欠点もいくつかあげられる．

1) アルゴリズムが同じとしても，二重化のために，より多くの設計時間と努力が必要である．(ただし，設計中にどんな耐故障性でも取り入れるための努力は常に必要である．)
2) このアプローチは，論理の分割化に関係なく，内部ゲートを相当利用することができる．(ただし，VLSI 内のピンの限界のため，使用可能な内部ゲートの個数は制限されることもある．)
3) 内部の重複は，チップ内部の障害の診断を簡単にするだけである．(この問題は，チップ間ネットとピンの障害を検出するために，従来の検査技法を取り入れることによって処理されてきた．また，冗長性方式は本質的に，他の標準アプローチよりも障害分離を大いに増やすことに留意しなければならない．)
4) 相補論理回路は，論理レベルの追加が必要かもしれないので，機能的論理回路よりも多少遅くなることが考えられる．機能論理回路から相補論理回路への変換は，常に，一対一で対応して行われるものではない．(ただし，相補論理回路の実行にはしばしばもっと少ない論理レベルですむ場合もあるので，トレード・オフ効果が起きる場合もある．さらに，相補論理回路内の遅れはマシン全体の性能には影響せず，遅れは検出回路で考慮することにより許容することができる．)
5) VLSI 内のピン限界の問題が指摘されたが，この耐故障アプローチは，従来の技法よりも多くのピンを使うことが考えられる．(ただし，耐故障性のために使うピンが少し増加しただけで，誤りの検出，回復，および分離特性は従来の技法に比して劇的に改良されている．さらに VLSI ゲート密度のより高度な利用が実現されている．)

4.3 障害の回復

この設計における障害回復の基本原理は，記憶装置(主および制御)の固定障害には ECC によってマスクをかけ，機械内の他の部分で起きた断続的または一時的障害には，適当な再試行技法によって回復を試みるというものである．

ECC は，通常，単一誤り修正あるいは二重誤り検出機能を有しており，単一ビット誤りは障害分離のために記録することができる．ここで，1 つの原理について示しておく．これは，配線制御論理を利用するよりも，できるだけ多くの制御論理を制御記憶装置内に

入れるという原理である．これによって，ECC による制御記憶装置内での障害回復が容易となり，経済的にもなり，これが基本的な設計上の目的が果たされる．

記憶装置以外で起きた障害に対する根本的な回復技法は，演算の再試行である．障害を検出したときの重要な3種の機械状態は，マクロ命令の実行，入出力操作（チャネルは中央処理装置（CPU）とともに内蔵されているため），および割込みのデコードである．どの場合にも，適当な再開始点から再試行できるだけの十分なデータがストアされる．障害を検出したときのマシンの一般的な操作の流れを図6に示す．

障害を検出したその VLSI チップ（単数または複数）は，誤りコードを EHC に送る．それからいくつかの操作が起こる．誤りコードは，障害の固有の標識として状態レジスタに入れられる．コンソールに記号が送られ，障害回復処理が開始されている旨の警告がなされる（コンソールは自身のマイクロ・プロセッサとクロックにより制御され，論理的に

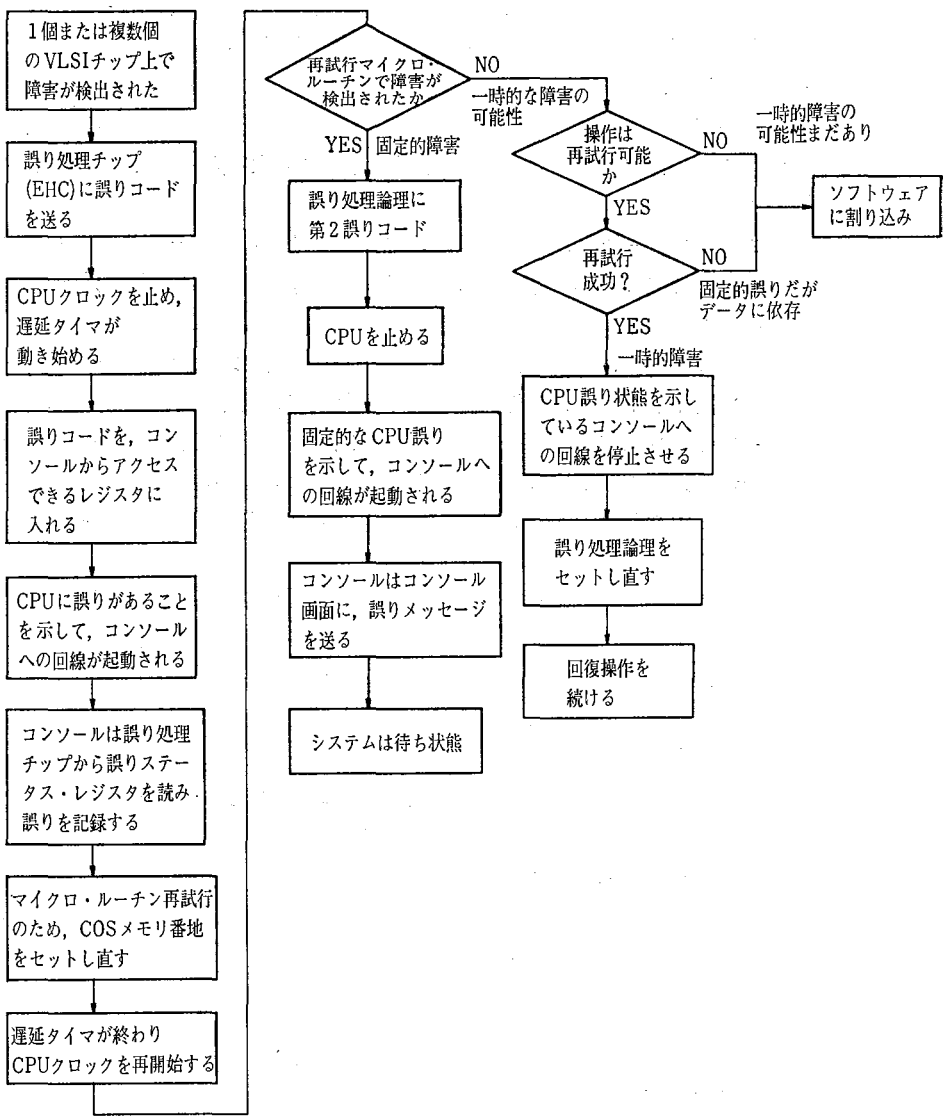


図 6　障害の回復手続き
Fig. 6　Fault recovery procedure

独立した装置である). コンソールは EHC 内の誤り状態レジスタを読んで，データを記録する．ストアされている状態は，障害の起きたチップを示しているか，または，チップ内に障害がなかったことを示しているかのいずれかであろう．一時的障害を通過させるために，CPU クロックは停止し，遅延タイマが起動される．

遅延タイマが終了すると CPU クロックが再開始されて，誤りの起きたマクロ命令，割込み，またはチャネル機能の再試行をする再試行マイクロ・ルーチンが実行される．このルーチンは，正しい再試行操作を遂行するために，命令実行中に保存されたデータを使用する．同じ誤りまたはもう1つ他の誤りが検出されたためにマイクロ・ルーチン自体が正しく操作できなくなると，EHC に対する2番目の誤りコードが固定障害手続きを起動させる．この場合，中央処理装置は停止しており，2番目の信号がコンソールのマイクロ・プロセッサに送られる．コンソール・マイクロプロセッサは，フロッピー・ディスクのような外部記憶装置を持つので，中央処理装置が休止中でも，誤り状態を取り出してコンソール画面にメッセージを出すことができる．障害検出の分離化がなされているために，フィールドで交換可能な単位を示すのに十分な情報が提供される．したがって，下記のようなメッセージが出されるであろう．

```
CPU SOLID FAILURE
CARD #28
CHIP #A14
```

これは，どのプリント基板上のどの VLSI チップに障害が起きたかを示している．保守作業は，したがって，チップの交換に限られる．EHC 自体の中で起きた障害についても，コンソールへの独立した信号を用いて，同じように障害の表示が可能である．

再試行ルーチンの残り部分は，比較的従来どおりである．再試行に成功すると，コンソール・マイクロ・プロセッサにその旨が知らされる．この場合，コンソール・マイクロ・プロセッサは，将来参照するために，回復可能誤り状態を記録するだけである．再試行に失敗した場合には，ソフトウェアに割込みが起きて適当な操作が行われる．どちらの場合でも誤り状態は記録される．

このように，VLSI による耐故障設計は，障害回復設計基準，記憶装置内の単一障害の修正，他の領域で障害が起きたときの操作の再試行，ロギング，および障害の分離を満足させることがわかる．考えられる次の問題は，障害が VLSI チップ内で検出されたときに片肺で動作も実行させうるかどうかである．これは，重複した回路を独立して動作させることのできる特別の回路を必要とするかもしれない．障害を検出すると，それぞれの回路にテスト・パターンが実行されて，障害を一方の回路，またはもう一方に分離することを試みる．こうして，チップが交換されるまで，片側の回路で操作を続ける．このアプローチでは，片肺動作しているチップでは障害の検出がされず，設計基準の1つに違反しているため，これまで積極的に考慮されなかった．ここで述べる詳細な障害分離機能により，コンピュータが比較的長時間使用できないことは前提からありえないと考えるので，片肺モードの操作の必要性はごくわずかであると考えられる．

4.4 障害の分離

システムの一生を通じて考えるとき，注意すべき重要なことは，システム保守費用が製造費用を超えることもあるという点である．この費用は主に，診断ツールの開発と支援，訓練，初期部品の設置，客先担当技術者の人件費，および交換用部品に当てられる．

この論文で述べている障害の検出，回復，分離の能力を，全く持たないシステムについ

ての最近の研究によれば，保守費用は次のように見積もられている．

人件費（移動時間を含む）	34%
初期部品	26
交換用部品	21
訓　練	6
診断の開発と支援	5
その他	8
合　計	100%

しかしこの新しいコンピュータの設計が，上記の全範疇の費用を減少させること，さらにいくつかの項目ではかなりの量まで減少させることが期待できる．

論理分割と障害検出ですぐに交換可能ユニットを指示するような原理は，ほとんどの障害に対して診断が必要でないことを意味している．4.3 節で，中央処理装置内で回復不能な障害が起きると，コンソール画面に，故障のあるチップを示すメッセージが生成されることを示した．このため，診断プログラムの開発作業は軽減されるが，保守作業の動作確認と一般的なコンピュータ動作の保証のためのプログラムは依然として必要である．

背面ボードのショート，ホイルのオープン，電源，クロック分配の障害等，ある種の障害では，交換ユニットを指摘できない場合がある．これらの場合には，保守操作では，交換の容易さが必要とされる．たとえば VLSI チップへの制御信号または入力データが正しくない場合（図4参照），障害はチップ・ピン入力にあるか，チップを含むカード上のホイル，背面ボード内，信号が発せられたカード上のホイル，または，信号が発せられたチップの出力ピンのどれかにあるだろう．コンソール画面は誤りの分離に基づいて適当な保守の手続きを指示する．たとえばチップまたはカードの交換を，あるいはさらに広範囲な作業，たとえば背面ボードと接続部分（コネクター，ケーブル）検査が指示される．しかし完成したプロダクトでは，障害の大部分はチップに原因がある．

4.3 節のような型の障害表示が，現在の訓練プログラムの大半を形成している．これにより，診断手法のための広範な訓練を不要にし，技術員の補修作業を軽減させることは極めて明らかである．修理は，通常，IC チップの交換により行われ，交換過程を簡単にするために恐らく，このチップはソケットに差し込みにされる．こうして，客先担当技術者の訓練に要する費用は大幅に減少するであろう．

初期設置部品および交換用部品について考える際に，多層プリントの価格がしばしば，プリント基板に含まれるチップの価格を越えることがある．現在のシステムは交換可能ユニットがプリント基板であり，各コンピュータまたは場所的に接近した数台のコンピュータに対して固有のプリント基板セットが必要なので，初期設置部品にかかる費用は非常に高い．交換したボードは修理して，できれば再テストするかそうでなければ廃棄しなければならないので，交換費用も高くなる．しかしここで説明する設計では，交換できるユニットはほとんどの場合チップであり，適当な地理的位置（おそらく各サイト）に1組のチップを置くことができる．当然ながら，チップ障害でなくボード障害が起きた場合には，部品補給拠点にボードのセットが要求されることもある．しかしこの障害の起こる率はチップ障害の率よりもはるかに低い．したがって，初期設置部品の費用は著しく軽減され，また，ボード修理の必要が少なくなるために交換の費用も削減できる．

最後に，人件費が減るものと予想できる．機械の故障の起きる率が現在のマシンと同じ

であったとしても，問題の診断に要する時間は概して非常に短くなる．診断はしばしば修理時間の 85 パーセントに達している．診断時間の短縮は，ほとんどの間欠的誤りから修理なしで回復すること，都合のよいときに修理が行えるようにどのチップが間欠的かを示してその回復をロギングすること，1回の診断時間を減少すること，および多くの人間と時間を必要とする“困難な”問題をほとんど除去できることによって実現できる．

4.5 テ ス ト

テストの領域，とくにテスト・セットの生成については，まだ広く研究していない．しかし，テストの必要性がこの設計によってはなはだしく減少したことは明らかである．客先サイトの汎用コンピュータのテストは現在，診断の目的のために広く使われているのに対し，ここで提案する設計のテストは主に一般的なシステムの動作の保証のために，通常1日の作業の始まり，または修理の行われた後に使われるものである．前述のとおり，交換可能ユニットの診断と分離がほとんどの場合，自動的に行われることを目的としている．不完全なテスト・セットを使ったため受ける影響は，障害が発見できずにデータの完全性を危くすることではない．むしろ，検査ハードウェアによって，使用者の操作中障害を検出してしまうことである．

内部のテスト個所をアクセスするため，VLSI チップ上に入出力ピンを追加する必要が生じる等，詳細なテスト・アプローチの開発が幾分設計に影響を与える可能性がある．ただし，自動的障害検出があれば，テストの出力を予め知らされた正しいデータと比較する必要がないので，ピンが追加されなくても，テストは従来の高度にマイクロプログラム化された中央処理装置のテストよりも容易であることに注目すべきである．

5. VLSI チップを用いた新しいコンピュータの利点

5.1 障害の起きる確率に対するこの設計の効果

ここで提案している VLSI 設計は，検査をしない VLSI 設計よりも多くの回路素子とチップを持っているため，絶対的な故障率は検査しない場合よりも幾分多いであろう．VLSI チップの故障については限られた経験しかなく，また検査しない VLSI 設計を詳細に分析したことがないために，平均故障間隔 (MTBF) の差異を正確に見積もることは困難である．それでも，現在のシステムにおいては1個の物理的障害が複数のシステム障害を引き起こすことがあるので，必ずしも絶対的な故障率が最も重要な物差しではない．物理的障害は間欠的であったりデータ依存であることがあり，その結果として障害診断がむずかしくなって，複数のシステム障害を起こす．

ここで提案する設計には，システム障害率を引き下げるために役立つ2つの利点がある．第1に，間欠的障害はほとんどシステム障害を引き起こさないであろう．これは，高度の即時障害検出と効果的な再試行設計のために，ほとんどの場合回復が成功すると予想されるからである．第2に，詳細な自動的誤り分離は一時的障害であっても一般に障害を起こした交換可能ユニットを示すであろうから，最初に障害を検出した後，容易に交換できる．

VLSI チップの障害に関する情報はわずかであるが，設計の不備（ノイズ，ローディング，競合状態による障害）と内部チップの障害モードによる間欠的障害が故障のほとんどを占めると考えられる．したがって，平均システム故障間隔 (MTBS) を大いに上げるために，MTBF を多少下げるかもしれない検出/回復/分離回路を付加することは，極めて妥当な対処法である．

5.2 費用の有効性

この論文で論じているアプローチの付加的費用を考える場合，VLSI の設計ではピンの要素が非常に大きな制限を与えていることを忘れてはならない．内部の重複は，しばしば使わない論理スペースを利用しており，同時に，オフ・チップ・ネットのための信号ピンをほとんど必要としない．この予備的研究の結果では，3重モジュラ冗長度 (TMR : triple modular redundancy) もまたこの技法の延長上であり，役立つ技術になりうることを示している．ピン制限問題が，チップ間ネットのための時間多重化，バス結合，直列化およびコード化方式の使用を必要とする点は注目に値する．このような技法も，この論文で述べた各種の誤り検査コード機構の使用を可能にする．

MSI/LSI 論理と比較して，VLSI は論理ボードの総数とボードの型を減少し費用を安くし，少なくとも同程度の性能を与える．ただし，従来の障害検出技法のみ（内部パリティ，またはその他の単純コード等）を使っている VLSI の場合と異なり，ここで述べている設計は，より高度の分離，より大きい障害セットの検出，より迅速な検出と回復，診断の必要性の徹底的な減少，およびより単純な保守作業を実現する．

費用についての考察をみると，いくつかの驚くべき結果が明らかになる．障害の許容範囲に関連する論理オーバヘッドは，重複相補ゲート，比較器とその他の障害検出回路，および誤り処理チップ (error handling chip) 論理からなる．これらの回路は，中央処理装置内のゲート総数のおよそ 55 パーセントを含んでいる．しかし，従来のような検査を行う VLSI コンピュータから，ここで述べているコンピュータに進むために設計に加えられたゲートの増加数を調べると，VLSI チップ数の増加は 5.5 パーセントにすぎない（従来のように検査を行うコンピュータは，メモリ上にパリティ・コードを持ち，データ・パス内で1つの障害に対して完全に検査を行い，制御論理内では障害検出を行わないものと仮定される）．このように増加の割合が小さい理由の1つとして，前に述べた制限の影響があげられる．現在の VLSI チップの設計は，ゲート密度よりもピンの個数によりしばしば制限されるのである．したがって，ここで述べる技法は，使用しない多数のゲートを利用しているので，従来のように検査を行う VLSI コンピュータに，多数のチップを追加する必要がない．チップ数増加の 5.5 パーセントという数字は，仮定されるゲートとピンの個数およびここで提供する設計で実行される機能に基づいたものである．他の仮定または機能であればこの割合は変わるであろうが，費用についての有効性の一般的結論は変わらないだろう．また，この提案で現在述べる範囲の中央処理装置は，システム全体のハードウェア費用の 15 から 25 パーセントのみを占めており，周辺装置が残りの大半を形づくっていることにも注意しなければならない．

6. おわりに

VLSI の出現は，汎用コンピュータの耐故障設計において，独自の問題とこれに対して独特の解決をもたらした．1ビットの障害を推定するという従来の障害検出技法は，極端に高密度のチップにおいて，ある障害モードには適していない場合がある．チップの外部的重複でさえも，物理的障害と障害検出の間の時間に多くの処理が行われるかもしれないので，回復は困難となる．しかし，制限された入出力ピンと組み合わされた VLSI で使えるゲートの個数が多いため，本稿で記述した一般的アプローチの使用が可能になり，費用の増加は少なく，非常に優れた障害の許容範囲特性を得ることができる．

したがって，ここで提案する設計アプローチは次のように結論できる．

1) 一時的または固定的な1個の障害の即時検出
2) 多重障害の大部分を即時検出
3) 連結した障害の大部分を即時検出
4) 電源およびクロック障害の即時検出
5) 一時的または間欠的障害の大部分からの回復の機会増大
6) 障害のあるチップまたはチップ間回路ネットワークの自動的な分離
7) 従来のように検査する VLSI 設計よりわずかに多いチップ数(5.5パーセント)

(プロダクト・サポート統括一部 伊東 玄 訳)

参考文献

[1] W. Carter, *et al.*, "Design of serviceability features for the IBM System/360," *IBM J.*, Apr. 1964, pp. 115-126.

[2] W. Carter and P. Schneider, "Design of dynamically checked computers," in *Proc. IFIP 68*, pp. 878-883, 1968.

[3] D. A. Anderson, "Design of self-checking digital networks using coding technique," Univ. of Illinois Coordinated Sci. Lab. Rep. R-527. Sept. 1971.

[4] M. A. Mehta, *et al.*, "Functions for improving diagnostic resolution in an LSI environment," *Spring Joint Comput. Conf. AFIPS Conf. Rec.*, 1972.

[5] J. F. Arnold, "The concept of coverage and its effect on the reliability model of a repairable system," *IEEE Trans. Comput.*, vol. C-22, Mar. 1973, pp. 251-255.

[6] M. Breuer and A. Friedman, *Diagnosis and Reliable Design of Digital Systems*. CA : Computer Science Press, 1976.

[7] N. Tanaka, *et al.*, "Computer with designed-in duplication at device, module level," Nikkei Electronics, Mar. 21, 1977.

[8] W. Carter, *et al.*, "Cost effectiveness of self-checking computer design," *Proc. Int. Conf. FTC*. Jun. 1977.

執筆者紹介 R. M. セドマック (Richard M. Sedmak)

1947年 Philadelphia 生まれ. 1971年イスラエル共和国 Lehigh 大学において電子工学で B. S. を取得. 1973年同大学において事務管理で M. S. を取得. 1971年 Sperry Univac 社に入社後, システム研究および開発組織内で種々の仕事を経験. 現在, 可用性, 信頼性, および保守性 (ARM) のマネージャとして, 主にハードウェアおよびソフトウェアの信頼性の仕事に従事.

H. L. リバゴット (Harris L. Liebergot)

1943生 Philadelphia 生まれ. 米国 Drexel 大学において, 1966年に物理学で B. S. を, 1971年に電子工学で M. S. を取得. 1967年に診断プログラマとして, Sperry Univac 社 (Blue Bell) に入社. 以来, ハードウェアおよびソフトウェア設計, コミュニケーション・システムの信頼性, システムの可用性の仕事に従事. 現在, 沖ユニバック社に出向中.

報告 MACRO: プログラム言語

MACRO: A Programming Language

S. R. Greenwood

要 約 Sperry Univac 社の Roseville 開発センターの言語システム・グループは，ここ数年来コンパイラの設計に力を注いできた．いままで主に行ってきた分野は，コンピュータ市場で広く使用されてきた言語群のコンパイラの開発である．MACRO はその成果であり，言語使用者および言語システム作成者のいずれの見地からも，非常に興味深い言語である．

MACRO は，パターン照合およびテキスト編集の機能を必要とするアプリケーションに対処するために開発されたプログラム言語である．本稿では MACRO の特徴および構成について言及し，また，SNOBOL のようなより一般的なテキスト処理言語との比較も行っている．さらに，MACRO 言語の設計および UNIVAC シリーズ 1100 のもとでの MACRO システムの作成についても論じている．

Abstract The Language Systems Group in Roseville has been in the business of designing compilers for several years. Their efforts to date have been primarily in the area of developing compilers for the Languages in common use in the computer marketplace. MACRO represents one of their attempts to design a language. MACRO is a very interesting language from both a user's and implementor's point of view.

MACRO is a programming language developed by Univac to cope with applications requiring pattern matching and text editing. By and large, the readers of this paper will be unfamiliar with MACRO. As a result, this paper covers in some depth the features and structure of the MACRO language. Parallels are drawn to more common text processing languages as SNOBOL. This paper concludes with a brief discussion of the design of the MACRO language and its implementation on Series 1100 equipment.

1. はじめに

MACRO は，PL/I に似た形式の手続き型の文字列取扱い用言語である．MACRO 言語では，PL/I に見られる通常の手続きのほかに，マクロと呼ぶ MACRO 言語特有の手続き的なものを定義している．MACRO 言語において，マクロは模様照合 (picture matching) 機能の基礎となるものである．

プログラミングに携わる人の大多数は，ALGOL, FORTRAN, PASCAL, PL/I 等の手続き型言語を用いている．したがって，これらの言語の構文を考慮した文字列取扱い用言語の構築は，非常に理にかなったことである．SNOBOL, COMIT, LISP 等，従来の文字列取扱い用の言語は，このような経緯をたどらなかった．COMIT や LISP における形式と制御の流れは，前述した手続き言語らときわめて異なる．SNOBOL は，ALGOL, FORTRAN 等の言語から派生的に生まれた言語にもかかわらず，文字列取扱いの機能は，文のレベルで統合されている．したがって，SNOBOL では文字列取扱いの機能が，代入文および制御の流れに影響する．一方，MACRO においては，従来の手続き型言語で定義している一連の文を変更することなしに，パターン照合の機能を追加している．したがっ

Reprinted from ACM SIGPLAN NOTICES, (December 1979), Vol. 14, No. 12, pp. 80-91.

て MACRO は，一般に使用されている言語に対して文字列取扱いの機能を追加する手法を示唆しているという意味で，注目に値する言語である．

MACRO 言語の主要部分は，IBM PL/I の持つ前処理機能に対抗するように設計された SHARE 提案 (1970年) の中で最初に形づくられている．提案の意図は，提案書によれば「アプリケーション・プログラマの負荷を軽減するために手軽な表記法の開発を可能とすること」であった．提案には，3つの主要目標が含まれていた．第1に，マクロの定義により，PL/I 言語の構文を拡張することができること．第2に，マクロによる前処理を施すことによって，PL/I の意味構造上に定義した特殊目的の言語を PL/I 言語に変換することができること．そして第3の目標は，プログラム生成機能の供給であった．たとえば，マクロを適切に使用することにより，複数組の一連の文を選択的に翻訳できることである．どの IBM PL/I コンパイラにもいまだにマクロ機能が供給されないことからみて，この原案は時期尚早な目標を掲げていたと思われる．

Sperry Univac 社の PL/I グループは，開発中の PL/I コンパイラにおける原始プログラムの前処理機能の必要性を認識した．標準 PL/I 自体はそのような機能を有していなかったが，PL/I におけるマクロ機能に対する要求は増加した．また IBM PL/I プログラムから新しい標準 PL/I に準拠したプログラムへの変換が必要となった．このようなプロセッサを設計するための起点として，SHARE での提案が用いられた．新言語 MACRO の概念の多くは，この提案に基づいている．しかし，設計に際して SHARE 提案と異なり，新言語を PL/I から独立させることと，効率を高めることを目的とした．その成果である MACRO は，各種言語の前処理機能を提供したり，一般のテキスト処理のために独立した形態で使用できる汎用の文字列取扱いの機能を具現したものである．

2. MACRO 言語におけるブロックの構築

MACRO プログラムはストリームとしてのシンボリック・テキストを変換するソフトウェアである．この考え方が MACRO 言語の基本設計思想である．MACRO システムへ入力されるテキストは入力ストリームと呼ばれ，その変換結果は出力ストリームと呼ばれる．使用者によって書かれた MACRO プログラムは，変換方法を記述した一連の定義である．これらの定義は，入力ストリームから検出すべきパターンを宣言すると同時に，これらのパターンを置換して出力ストリームを構成すべきテキストを宣言する．

MACRO において，パターン照合は“語彙 (token) の構築”と“模様 (picture)”との2つのレベルで行われる．このうち，語彙の構築での照合がより基本的である．語彙は文字のレベルでパターンを指定したもので，たとえば，整数を語彙として記述すると次のようになる．

('+'\'-') 〈'0'\'1'\'2'\'3'\'4'\'5'\'6''7'\'8'\'9'〉 ･･･

パターン照合のより複雑なレベルを模様と呼ぶ．模様によって記述するパターンは，語彙，つづり (literal)，および他の模様から構成される．たとえば，FORTRAN の代入文の構文を，模様として記述すると次のとおりである．

〈 変数\配列要素 〉'='〈 式 〉

語彙には，変数と同様に名前をつけることができ，模様の中から語彙を参照するときに，その名前を用いる．一方，模様には名前がつかない．模様はマクロの構成要素である．マクロは MACRO 言語における構文要素であり，模様が表すパターンとそのパターンに対する置換テキストとを結合させる機能を持つ．この置換テキストはマクロ本体の中で生成

される．マクロの構文は PL/I の手続きに類似している．マクロはマクロ見出しで始まる．マクロ見出しは，マクロ名，オプションの並び，模様で始まる．すなわち，オプションによりマクロの実行の仕方を制御し，模様によりテキストの置換に先立って照合すべきパターンを指定する．マクロ見出しに続いて現れるマクロ本体は，置換テキストを生成するためのアルゴリズムを記述した一連の文から構成される．次の例は，入力ストリームに現れる連続した間隔文字を，1つの間隔文字で置き換えるための簡単なマクロである．

```
TOKEN  SPACES 〈' '・・・〉;
MACRO  REPLACE  TRIGGER 〈SPACES〉;
   ANSWER ' ' ;
END ;
```

マクロ REPLACE は，模様としてひとつの語彙 SPACES を指定している．入力ストリームから語彙 SPACES を検出するごとに，マクロREPLACE が起動し，模様照合を試み，成功後にマクロ本体を実行する．上の例では，ANSWER 文によって模様照合に成功したテキストを1つの間隔文字に置き換えている．TRIGGER は，マクロに対して指定できるオプションの1つであり，引き金 (trigger) マクロと構文マクロの2種類のマクロを区別するために用いる．TRIGGER オプションを指定して宣言したマクロは引き金マクロであり，入力ストリームの中に引き金語彙と呼ばれる特定の語彙が現れたときに起動する．起動すると，模様に指定したパターンの残り部分を入力ストリームから検出すること(模様照合)を試みる．入力ストリーム中の一連の文字が，引き金マクロの模様と一致すると，マクロ本体を実行する．

マクロには構文マクロと呼ばれ，TRIGGER オプションを指定せずに宣言できるものがある．このマクロは，式の中から呼び出される関数に類似している．模様中に指定した構文マクロ名に模様照合の過程で到達したとき，構文マクロは起動し，入力ストリームに対してその構文マクロの模様照合を行う．この模様照合に成功すると，構文マクロのマクロ本体を実行し，置換テキストを生成する．構文マクロが生成したテキストは，その構文マクロを起動させた模様の中へ返される．構文マクロから制御が返されると，構文マクロを起動させた地点の次から模様照合を継続する．

3. MACRO 処理系

すべてのプログラム言語は，ある処理系のもとに定義されている．この処理系は，言語の構文を解析し，適切な意味処理を行う．コンパイラの場合は，この処理を2段階の処理に区分している．まずプログラムの構文を解析し，適合する意味を踏まえたうえで，より単純な言語（機械語）へ翻訳する．

ALGOL, FORTRAN, PASCAL および PL/I の強みの1つは，これらの言語が非常によく似た処理系を必要としていることである．これは明らかに言語使用者にも言語システム作成者にも好都合である．この特長により，ある言語での概念がそのまま他の言語に適用可能になる．しかしながら，LISP はこれらの言語とは違って種々の点ではるかに単純で，非常に異なる処理系のもとに構築されている．これが，LISP がプログラマに広く受け入れられなかった主な理由である．

MACRO 処理系は，ALGOL, FORTRAN, PASCAL, および PL/I における従来の処理系の拡張である．MACRO 処理系は4つの主機能から構成され，その1つが従来の処理系に対応している(図1)．図1における矢印は，呼出しの関係を表している．語彙構築

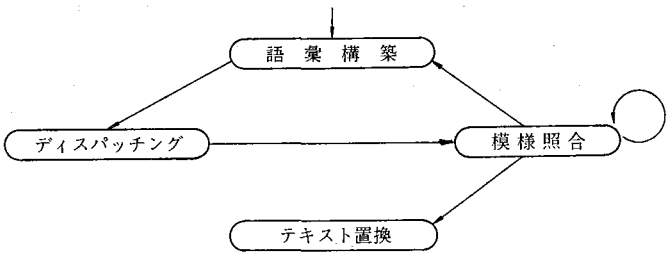


図 1　MACRO 処理系の機能
Fig. 1　MACRO machine activites

の機能は，ディスパッチングの機能(以降ディスパッチャと略称)を呼び出す．ディスパッチャは，模様照合の機能を呼び出す．各機能は，その処理を完了すると自分を呼び出した機能へ制御を返す．MACRO 処理系に関して，図1はさらに次の2点を示している．まず，第1は各機能が互いに再帰的なこと，第2は語彙構築の機能に制御が授けられることである．

語彙構築は入力ストリームの文字列から語彙を構築する．この機能へ引き渡されるパラメタは，入力ストリーム内において走査を開始すべき位置，および原始プログラムにより与えられた語彙の定義である．語彙の定義は互いに重復した部分があってもかまわないため，その時点の位置から入力ストリームを走査して構築可能な語彙が複数存在してもよい．MACRO における語彙照合のアルゴリズム(入力テキストを走査して語彙を構築する方法)は簡潔である．また，すべての語彙の定義は広域的で，MACRO は原始プログラム内で定義されている順序で語彙を検索する．入力テキストから語彙構築がなされると，ディスパッチャが呼び出される．

ディスパッチャは，“成功”(入力ストリームが変更されたことを示す)または“失敗”(入力ストリームが変更されなかったことを示す) のいずれかの情報を，語彙構築の機能へもどす．“成功”のときには，語彙構築の機能は新たに入力ストリームを走査して語彙を構築する．“失敗”のときには，語彙構築の処理は完了する．語彙構築の機能が模様照合の機能から呼び出された場合には，前者は後者に対して構築した語彙を返す．模様照合の機能から呼び出されなかった場合には，構築された語彙に関連づけられたテキストが出力ストリームへ転送される．

ディスパッチャは，種々の点において MACRO 処理系の中心部であり，引き金マクロの起動を制御するとともに，入力ストリームが模様として照合したか否かを語彙構築の機能に知らせるスケジューラである．語彙構築の機能は，入力ストリームから構築した語彙を，ディスパッチャへ引き渡す．この引き渡された語彙が，原始プログラムで定義した各引き金マクロの引き金語彙であるか否かを検査できる．ディスパッチャは，語彙構築の機能により構築された語彙を引き金語彙として持つマクロを順次起動させ，その模様が入力ストリームの一部と照合するマクロを探す．同じ引き金語彙を持つ引き金マクロが，ディスパッチャにより起動させられる順序は，原始プログラムに基づいて決められる．

MACRO 言語では，マクロのマクロ本体の中に他のマクロを定義することができる．ただし，マクロ本体の中で定義したマクロは，そのマクロを直接含むマクロの起動中（模様照合中)にのみ，起動することができる．入れ子になっていないマクロは，常に起動可能である．起動させるべきマクロの検索は，階層的に行われる．各時点で複数の引き金マクロが起動可能（模様照合可能）でありうるが，そのうち実際に起動している入力ストリ

ームを走査し，模様照合を行っているものはただ1つである．起動させるべきマクロの検索は，この模様照合中のマクロを基準に行われる．まず，模様照合中のマクロのマクロ本体で定義した引き金マクロの起動が試みられる．マクロ本体で定義した引き金マクロが複数あるときは，原始プログラム内で定義されている順番に試みられる．あるレベル内のすべての引き金マクロの起動および模様照合に失敗したときには，その外側のレベルにある引き金マクロが試みられる．各マクロの起動は，模様照合の機能の呼出しを意味する．模様照合の機能は，入力ストリームを走査して模様照合に成功したか否かをディスパッチャに知らせる．模様照合の“成功”情報が返ると，制御はディスパッチャから語彙構築の機能へ返され，ディスパッチング“成功”の情報が渡される．模様照合の“失敗”情報がもどると，ディスパッチャは起動可能な引き金マクロの検索を続ける．どの引き金マクロの起動および模様照合にも成功しなかったとき，ディスパッチャは語彙構築の機能へディスパッチング“失敗”の情報を返す．

ディスパッチャは，入力ストリーム中の走査を開始すべき位置や照合すべきマクロの模様を指定して，模様照合の機能を呼び出す．模様照合の機能は，指定された走査位置および模様に基づいて，入力ストリームが模様に合致するか否か(模様照合に成功したか否か)を決める．模様中に語彙の指定があると，語彙を構築するために語彙構築の機能が呼び出される．この時点で，MACRO 処理系は再帰的に動作する．構築された語彙によって，ディスパッチャはさらに続けてマクロを起動させて，模様照合の機能を呼び出すことがある．模様の中から構文マクロが参照されて起動すると，やはり模様照合の機能が再帰的に呼び出される．この場合，模様照合の機能は自らを呼び出すことになる．模様照合に成功すると，テキスト置換の機能が呼び出される．

テキスト置換の機能は，ALGOL, FORTRAN, PASCAL, および PL/I に対する従来の処理系にきわめて似ている．テキスト置換用言語の文は2つのグループ，すなわちテキスト生成の文，および模様照合の機能と連結する文に分けることができる．テキスト生成の文には，手続きの定義，手続きの呼出し，BEGIN/END ブロック，IF 文，WHILE および通常の代入文がある．模様照合の機能と連結する文には，ANSWER 文，FAIL 文および RESCAN 文の3つがある．ANSWER 文は置換テキストを指定するための文である．FAIL 文は，模様照合に“成功”したにもかかわらず，強制的に“失敗”であったことにする文である．RESCAN 文は，ANSWER 文等が生成したテキストを再び走査することにより，マクロを起動させるか否かを指定する文である．

以上の概観はもちろん，MACRO 処理系の一部に触れているにすぎないが，MACRO を他のテキスト処理系と対比するための基本事項はとりあげた．MACRO は，文脈自由の構文解析方法を大いに利用している．この処理系の模様照合の機能は，おおむねプッシュダウン有限オートマトンである．2つのレベルに分かれたパターン照合方式，および模様照合に続くテキスト置換処理の実行は，語彙構築にかける文脈自由の構文解析の概念，生成規則を介しての構文解析，生成規則と意味解析ルーチンとの相互作用に，そのまま対応する．ただし，マクロ起動の概念は，前述した構文解析の方式には無い概念である．文脈自由文法による生成規則の間の関係は，構文マクロの間の関係に限定される．MACRO 処理系は，一連のプッシュダウン・オートマトンの起動と考えることができる．引き金マクロの起動は，新たなオートマトンが起動するメカニズムである．これらの起動は，順次(あるマクロの実行が終了した後に他のマクロが起動する)行われるか，または，入れ子の形で行われる．置換テキストと入力ストリームの相互作用により，あるオートマトン

が，それに続くオートマトンの入力を変更することができる．

ここで MACRO と SNOBOL 4 を比較してみよう．これらの 2 つの言語は，手続き型言語という点に関して，それぞれ異なった方法で対処している． SNOBOL 4 は，基本的な従来の処理系を少しだけ変更している． SNOBOL 4 における模様照合および置換は，従来の代入文を拡張したものである．

サブジェクト　パターン　＝　オブジェクト

上例に示すように，SNOBOL 4 では，代入文にパターンの指定を追加している． 指定したパターンが指定したサブジェクトの部分列と合致すると，その部分列は指定したオブジェクトに置き換えられる． サブジェクトの初期値は，代入文または INPUT 関数によって設定される．SNOBOL 4 処理系では，制御の流れ方が若干変更されている． 各模様は，成功または失敗する．制御は，暗黙的に前の文から直後の文へと移るが，パターンの“成功”または“失敗”を条件として，任意の文へ制御を移すこともできる．入力ストリーム，出力ストリーム，マクロ，語彙，および引き金マクロの起動ということは，この言語では定義されていない． SNOBOL 4 においては，手続き型言語に対して文レベルの変更を行っている．他方 MACRO においては，基礎となる言語を何ら変更せず，その代わりに従来の処理系を大幅に機能拡張することにより，その言語の既存の機能とともに模様照合の機能を構築している．

4. MACRO 言語の構文

MACRO 言語の具体的な構文は，全く慣用的なものである．また，その構文規則を記述するために用いる超言語も，慣用的なものである．

模様を構成するための構文の記述は，まず基本的な構文の記述から始め，これらの基本的な構文を用いた複雑な構文を構成していく．集合（set）は語彙を構成するために用いられ，語彙は集団（group）を構成するために用いられ，また，語彙および集団はマクロの模様を構成するために用いられる．

4.1 集　　合

MACRO における集合とは，MACRO 文字集合の部分集合に名前をつけたものである．集合は，言語機能をさほど増やすものではないが，複雑な構文の定義を簡明にするためにきわめて有用である．

```
SET 〈集合名〉 [ NOT ] '〈' 〈集合項目〉 [ '\' 〈集合項目〉 ] … '〉' ';'
〈集合項目〉 ::= 〈単長つづり〉 [ '..' 〈単長つづり〉 ]
```

集合の定義は，〈単長つづり〉というただ 1 種類のオペランドから構成される．〈単長つづり〉は，MACRO 文字集合に含まれる任意の文字であり，標準の文字集合には現れない MACRO 固有の特殊文字も指定できる．シリーズ 1100 の MACRO においては，ASCII 文字集合を使用する．もちろん，ASCII 文字には印書できない文字も含まれていて，これらはプログラム中に現れないため，かわりにこれらの文字に対し特殊名を定義している．たとえば，S'CR', S'VT' および S'NUL' は，それぞれキャリッジ・リターン，縦方向のタブ，およびナルを表す特殊記号である．標準の文字集合には現れない 3 個の特殊文字もこの形式で表し，それぞれ S'SOT', S'EOL', S'EOF' と書く．これらの記号は，それぞれ入力ストリームの始り，入力ストリームの終り，ファイルの終りを表している．これらの ASCII 以外の文字はパターンにおいて，しばしば便利な区切り記号となる． これらの任意の文字を，集合の定義のオペランドとして用いることができる．

集合に関連して，3つの演算子が定義されている．第1の演算子“\”は，集合内の一連の選択文字を区切るのに用いる．次の例の集合は，2つの文字Aおよびaを表している．

```
SET a 〈'A' \ 'a' 〉;
```

第2の演算子“‥”は一連の文字の範囲を定義する．この“範囲演算子”の左側のオペランドが示す文字から，右側のオペランドを示す文字までの，照合順序におけるすべての文字がその集合に含まれる．たとえば，次の例の集合 digit は，文字0から文字9までの10文字を表す．

```
SET digit 〈'0'‥'9'〉;
```

ときには，ある文字を除外した集合が必要となる．この目的のために，第3の演算子 NOT がある． NOT 演算子を使用して定義した集合には，その定義で指定した文字以外のすべての ASCII 文字が含まれる．次の例の集合 alpha と beta は，同等である．

```
SET alpha 〈'A'‥'Z' \ 'a'‥'z' 〉;
SET beta NOT 〈S'NUL'‥'@' \ '['‥'`' \ '{'‥S'DEL' 〉;
```

4.2 語　彙

集合は，文字に関して定義されるが，語彙もやはり文字に関して定義される．ただし，語彙の場合は個々の文字よりも大きな単位で，ある字句に対するブロックを構成するためのものである．語彙の定義には，2つの型のオペランド，つづり，および集合が現れる．

```
TOKEN 〈語彙名〉 [ IGNORE ]
                    '〈' 〈語彙句〉 … [ '\' 〈語彙句〉 … ] … '〉' ';'
〈語彙句〉 ::= 〈語彙項目〉 ['…']
〈語彙項目〉 ::= { 〈つづり〉 | 〈集合名〉 |
                  '〈' 〈語彙句〉 … [ '\' 〈語彙句〉 … ] … '〉' |
                  '(' 〈語彙句〉 … [ '\' 〈語彙句〉 … ] … ')' }
```

〈つづり〉は，両端を引用符で区切った一連の文字である．引用符で区切った一連の文字の中に，ASCII 以外の文字，および印書されない ASCII 制御文字が現れてはならない．ただし，それらの文字を表す特殊記号を指定することにより，それぞれ1文字長の〈つづり〉として表せる．

語彙のオペランドは，3つの演算子および2組の区切り記号を任意に用いて，連結することができる．区切り記号は，代数における括弧と同様の機能を持ち，標準の優先順位を強調したり変更したりするために，演算子およびオペランドをグループ化する記号として用いる．さらに，区切り記号“(”および“)”は，その中に指定した〈語彙項目〉が現れても現れなくてもよく，任意であることを表す．

表 1　語彙演算子および区切り記号
Table 1　Token operators and delimiters

記　号	意　味
〈 〉	グループ化区切り記号
( )	任意グループ化区切り記号
\	選択演算子
…	繰返し演算子
	連結演算子

選択演算子は，オペランドの選択を表す．演算子“\”を用いて分離した一連のオペランドが括弧を用いてグループ化されているときには，そのパターンと照合されるテキストの中に，それらのオペランドのうちの1つが現れてもかまわない．もし，それらのオペランドが角括弧を用いてグループ化されているとき，パターンと照合されるテキストの中に，

それらのオペランドのうちの１つが必ず現れなければならない．省略符号 … は，繰返しの記号である．この記号が，任意の〈語彙項目〉の直後に現れると，その〈語彙項目〉が０回以上任意に繰り返し現れてよい．任意でない〈語彙項目〉の直後に繰返しの記号が現れると，その〈語彙項目〉が１回以上任意に繰り返し現れてよい．また，暗黙の語彙演算子として連結演算子がある．これは，語彙の定義において〈語彙句〉が互いに隣接して現れた場合を指し，それらの〈語彙句〉として照合するテキストが入力ストリーム中に隣接して現れなければならないことを意味する．これらの語彙演算子のうちでもっとも優先順位の高いのが繰返し演算子 "…" であり，以下，暗黙の語彙演算子，選択演算子の順である．グループ化の区切り記号は，この優先順位を変更するために使用することができる．

これらのオペランド，演算子および区切り記号を用いることにより，きわめて巧妙なパターンが記述できる．COBOL におけるデータ名の形式は，次のように指定する．

```
SET  alpha 〈 'A' .. 'Z' 〉 ;
SET  alpha-num 〈 'A' .. 'Z' \ '0' .. '9' 〉 ;
TOKEN  data-name 〈 alpha  ( ( '-' )  alpha-num ) … 〉 ;
```

上の例で定義した名前は，アルファベットの大文字で始まり，０個以上の英数字が続き，それらの間にハイフンを書くことのできる形式を持つ．ただし，ハイフンを連続したり，最後に書くことはできない．

また，語彙の定義の特長として IGNORE オプションがある．IGNORE オプションは，そのオプションを指定して定義した語彙が構築されたとき，その語彙を模様照合の過程で無視することを指定する．このオプションは，間隔文字の列のように何の情報も含まない区切り記号に対して適用できる．このような区切り記号は，他の語彙を区切るにすぎない．たいていの言語では，注釈もこの部類に属する．

4.3 集　　団

模様を定義するうえで用いるときにさらに複合化されたものとして，集団の定義がある．集団の定義を用いることにより，複数の語彙をまとめて１つのものとして取り扱うことができる．各〈集団名〉は，その〈集団名〉が現れた時点で構築されるべき語彙の集合を表す．

```
GROUP 〈集団名〉 [ NOT ] '〈' 〈語彙名〉 [ '\' 〈語彙名〉 ] … '〉' ';'
```

この場合の演算子 "\" および NOT の意味は，集合の定義における意味と同様である．記号 "\" はその集団内で選択の対象となる語彙を区切る．各 MACRO プログラムにおいては有限個の語彙の集団が定義できる．NOT を用いると，その集団の定義で指定された以外の語彙からなる１つの集団が表せる．

4.4 模　　様

語彙の定義において使用できる演算子，区切り記号およびオペランドは機能的に完結したものであり，それらを使用すれば，MACRO が検出できるすべてのパターンを記述できる．したがって，マクロの模様は，ただ１つの語彙で表すように限定してもよいが，そのような限定はしていない．実際，模様では，MACRO におけるどの構文よりも豊富なオペランドおよび演算子が指定でき，この結果，パターンの構成が容易となっている．

```
〈模様�〉 ::= '〈' 〈模様句〉 [ '\' 〈模様句〉 ] … '〉' .
〈模様句〉 ::= { 〈模様項目〉 [ '…' ] } … .
〈模様項目〉 ::= { 〈語彙つづり〉 | 〈文字つづり〉 | 〈集合名〉 | 〈語彙名〉 | 〈集団名〉 |
                  〈マクロ名〉 | 〈NOT 句〉 | 〈括弧句〉 } .
〈NOT 句〉 ::= NOT { 〈語彙つづり〉 | 〈文字つづり〉 | 〈集合名〉 | 〈語彙名〉 |
                  〈集団名〉 } .
```

〈括弧句〉 == [〈変数名〉':'] 〈包含句〉.
〈包含句〉 == {'<'〈複合句〉'>' | '('〈複合句〉')'}.
〈複合句〉 == {〈模様句〉|〈模様句〉{'\'〈模様句〉} … |〈模様句〉'/'〈模様句〉}.

模様の定義においては，6種類のオペランド，5種類の演算子，および2種類の区切り記号を使用することができる．オペランドは模様内のサブ・パターンとみなせる．演算子および区切り記号は，模様内でサブ・パターンが配置される順序を制御するために用いられる．

模様におけるオペランドは，3つに分類される．第1類としては，語彙の構築を引き起こす〈語彙名〉，〈集団名〉および〈語彙つづり〉がある．模様照合の過程でこれらオペランドのいずれかに到達すると，語彙構築の機能が呼び出されて語彙が構築される．とくに〈語彙名〉に到達し，その結果構築された語彙の名前がその〈語彙名〉であったときに，サブ・パターンとして指定した〈語彙名〉は入力ストリームと照合される．また，〈集団名〉に到達し，その結果構築された語彙の名前がその集団を構成する〈語彙名〉であったときに，サブ・パターンとして指定した〈集団名〉は入力ストリームと照合される．〈語彙つづり〉の場合には，定義した語彙が特定なときにのみ照合する．〈語彙つづり〉が表すサブ・パターンは，両端をアポストロフィで区切られた一連の文字から構成される．その一連の文字が，構築された語彙を構成する一連の文字と一致したとき，サブ・パターンとして指定した〈語彙つづり〉は入力ストリームと照合される．

模様におけるオペランドの第2類としては，〈文字つづり〉および〈集合名〉がある．これらのオペランドは，単に入力ストリームから所定の文字列を検出することを指示するのみであり，語彙の構築を引き起こさない．〈文字つづり〉で指定した文字列と，長さおよび内容において同一の文字列を入力ストリームから検出できたとき，その〈文字つづり〉に対する照合に成功する．〈文字つづり〉は，その両端を区切る文字としてアポストロフィではなく，引用符 " を用いることによって，〈語彙つづり〉と区別される．実は，〈文字つづり〉と〈語彙つづり〉が機能的に区別されるのは，模様の中に現れた場合のみである．オペランドとして〈集合名〉が現れると，入力ストリーム中の次の1文字が検査される．入力ストリーム中の次の1文字が，その集合を構成する文字であったときに，〈集合名〉に対する照合に成功する．

マクロにおけるオペランドの融通性を示すために，MACRO 言語の注釈を記述する3通りの方法を次にあげる．これらは，それぞれオペランドを異なった方法で結びつけている．ここで，MACRO 言語の注釈のパターンは，両端を二重斜線 "//" で区切られた任意の MACRO 文字の列である．ただし，MACRO 言語の注釈は，複数行にわたることができないので，行終了文字によっても，注釈は終わる．

方法A

```
SET  non-slash  NOT  < '\' > ;
TOKEN  com  < '//'  ( ( '/' ) non-slash ) … < '//' ( '/' ) s'eol' > > ;
MACRO  comment  TRIGGER  <com> ;
```

方法B

```
SET  non-slash  NOT  < '/' > ;
TOKEN  body  <non-slash … >;
TOKEN  delimit  < '//' ( '/' ) s'eol'> ;
MACRO  comment  TRIGGER  < '//' ( ( ''/'' ) body ) … delimit> ;
```

方法C

```
SET  non-slash  NOT  < '/' > ;
TOKEN  delimit1  < '//' > ;
```

```
TOKEN  delimit2 〈 ( '/' ) s'eol' 〉 ;
GROUP  delimit 〈 delimit 1 \ delimit2 〉 ;
MACRO  comment  TRIGGER
       〈 delimit1 ( ("/") non-slash ) ··· delimit 〉 ;
```

方法Aにおいては，各注釈全体を1つの語彙として包み込んでいる．方法Bおよび方法Cにおいては，注釈の講文をマクロの模様によって表している．方法Bにおいては，マクロの引き金語彙として，区切り記号“//”を用いている．この〈語彙つづり〉は，語彙 delimit における特定の例である．方法Bにおいては，注釈の本体の大部分は，語彙 body によって照合される．方法Cにおいては，注釈を終わらせる語彙を表す集団を定義している．このマクロの引き金語彙として，語彙名を指定している．また，方法Cでは，注釈の中心部分を集合 non-slash で表している．

模様におけるオペランドの第3類として，構文マクロがある．構文マクロは，式の中で関数が使用される場合と同様の目的で，模様の中で使用される．構文マクロは，任意のパターンに名前をつけ，その名前を他のパターンの中から参照することにより，名前のつけられたパターンを参照することを可能とする．語彙はこの機能をある程度まで備えているが，次のようないくつかの大きな違いがある．語彙のオペランドは〈文字つづり〉または集合に限定されているが，構文マクロは模様の6種類のオペランドから構成される．したがって構文マクロは，他の構文マクロをその構成要素とすることもできるし，さらに再帰的であってもよい．次に示すブール代入文の構文を解析する一連の構文マクロの例によって，これらの差異が明らかになろう．

```
SET    alpha 〈 'A'..'Z' 〉 ;
TOKEN  var 〈 alpha ··· 〉 ;
TOKEN  spaces  IGNORE 〈 ' ' ··· 〉 ;
TOKEN  delimiters 〈 '=' \ '(' \ ')' 〉 ;
MACRO  term 〈 'NOT' term \ var \ '(' exp ')' 〉 ;
  ⋮
MACRO  factor 〈 term ( 'AND' term ) ··· 〉 ;
  ⋮
MACRO  exp 〈 factor ( 'OR' factor ) ··· 〉 ;
  ⋮
MACRO  asg  TRIGGER 〈 'LET' var '=' exp 〉 ;
  ⋮
```

上の例と同等のパターンを語彙を用いて書くことは，非常にむずかしい．語彙には再帰的機能がないので，このパターンを構成するための語彙は，極度に複雑になる．

表 2　模様演算子および区切り記号
Table 2　Picture operators and delimiters

記　号	意　　味
〈 〉	グループ化区切り記号
( )	任意グループ化区切り記号
\	選択演算子
···	繰返し演算子
	連結演算子
/	並び演算子
NOT	否定演算子

マクロの模様における演算子および区切り記号としては，語彙の構成のために使用する演算子および区切り記号以外に，いくつかの演算子が使用できる．模様や語彙に共通な演算子および区切り記号は同じ意味を持つ．模様を構成できる演算子として，これら共通なもののほかに2つある．演算子 NOT は，集合または集団における使用法とほぼ同じで，サブ・パターンの照合の結果を逆転させる．すなわち，NOT の直後のサブ・パターンが入力ストリームとの照合に“成功”したとき，それは“失敗”を意味する．入力ストリームとの照合に“失敗”したとき，それは“成功”を意味する．NOT は，構文マクロまたは括弧句に対して指定できない．模様の中で指定できるもう1つの演算子は，並び演算子“/”である．模様の中に並びが頻繁に現れるため，並びのための演算子が付け加えられた．次の2つの模様は同等である．前者は，並び演算子を用いた例であり，後者は，並び演算子を用いない例である．

```
〈 OP1 / OP2 〉
〈 OP1 ( OP2 OP1 ) … 〉
```

並び演算子を用いる非常に一般的な例としては，コンマまたは間隔文字で分離した一連の項目をあげることができる．また，前述のブール代入文の例にも，並び演算子を適用できる．たとえば，例における2つの構文マクロは，次のように書く．

```
MACRO  factor 〈 term / 'AND' 〉 ;
  :
MACRO  exp 〈 factor / 'OR' 〉 ;
  :
```

5. 概念の相互結合

本稿の内容の進め方についての前提事項の1つは，MACRO における模様照合の機能と MACRO の基底言語（base-language）の機能とが互いに独立していることである．このことは，単に言語の構文で分離していたり，MACRO 処理系の機能が分離していること以上に，優れた点である．言語におけるこの2つの機能は形式の上できわめて印象的な対照をなしている．MACRO におけるパターンを記述するための構文は機能的言語である．すでに述べたように，オペランドおよび演算子はパターンの構成を記述する．これらのパターンは，ある文字列がパターンの基準を満たすか否かを検査する詳細なアルゴリズムについては何ら指定していない．MACRO における集合，語彙，集団，模様には受け継ぐべき制御の流れの概念は無い．このことは，基底言語で書かれるマクロ本体と比較して，対照的である．マクロ本体の構文は，すべての von Neumann 型コンピュータの基本である分岐命令，テスト命令，ロード命令およびストア命令の高級な場合である．マクロ本体は，機能的記述というよりもむしろアルゴリズムの記述である．このように MACRO 言語は，機能的構文およびアルゴリズム的構文の両方を併せ持つ特異な併合体である．本章の主題は，これら2つの部分のデータを授受する方法についてである．

5.1 模 様 変 数

模様変数は，模様によって照合したテキストをマクロ本体へ引き渡すための手段である．たとえば，あるパラグラフに現れるすべての語を識別して，各々の語が現れた行を印書するために，MACRO 言語を利用するものとする．パラグラフを構成する各々の語を検出するには，そのテキストを照合する模様句を指定し，かつ，その直前に模様変数名を指定しておく．こうすることで，模様句によって照合したテキストは，模様変数に保存される．この例を示すと，次のとおりである．

```
SET alpha 〈 'a' .. 'z' \ 'A' .. 'Z' 〉 ;
TOKEN word-form 〈 alpha ... 〉 ;
TOKEN spaces IGNORE 〈 ' '... 〉 ;
MACRO find-words TRIGGER 〈 word : 〈word-form〉 〉 ;
  ⋮
```

この例において，入力ストリームから検出された各々の語は，模様変数 word へ保存される．変数 word は，マクロ find-words のマクロ本体から，他の変数と同様に参照される．

ここで，MACRO における変数について若干触れておこう．MACRO において基本的なデータ型は文字列である．文字列を保存するために用いる変数の長さは，可変である．実際に代入された文字列の長さがその時点の変数の長さとなる．UNIVAC シリーズ 1100 のもとで作成された MACRO において，1つの変数に保存することのできる文字列の長さは25万余字までである．この可変長という特長は，テキストを模様として検出するときに非常に有用である．

5.2 模 様 配 列

入力テキストを保存するために用いる模様変数が，繰返し構文の中に現れた場合，どうなるか．前節の例のマクロ find-words を次のように変更した場合が，その例である．

```
MACRO sentence TRIGGER PROTECT
                〈 word : 〈word-form〉 ... 〉 ;
```

上の例の場合，模様変数 word は配列を構成する．繰り返し現れる各模様句 〈word-form〉 は，それぞれ別個の要素 word (1)，word (2)，... へ保存される．配列要素の数は，模様の指定からは決まらず，実際に照合した入力テキストによって決まる．模様変数はまた，入れ子の形で構文中に現れることもある．この場合，構成される模様配列の次元の数は模様の指定によって決まる．次元の数は，その模様変数名を入れ子の形で含む繰返し模様句の数に等しい．次の模様は，この例を示している．

```
〈 c : 〈 〈 '(' b : 〈 a : 〈 word-form 〉 / ',' 〉 ')' 〉 ... 〉 〉
```

この例で，変数 c は入れ子となっていないので，スカラーである．変数 b は，演算子 "..." によって，1次元の配列である．変数 a は，演算子 "..." および "/" によって，2次元の配列である．入力テキスト (x, x x, x x x)(y)(z, z z z z) が与えられたとき，a，b，c はそれぞれ次の値を持つ．

変 数	値
c	(x, x x, x x x) (y) (z, z z z z)
b(1)	x, x x, x x x
b(2)	y
b(3)	z, z z z z
a(1, 1)	x
a(1, 2)	x x
a(1, 3)	x x x
a(2, 1)	y
a(3, 1)	z
a(3, 2)	z z z z

5.3 変数および配列

変数および配列は，模様として照合した入力テキストを保存するためにのみ使用されるわけではない．それはマクロ本体で検査するための1つの応用にすぎない．変数および配列は，従来の言語と同様に，値を代入および参照することができる．MACRO における

変数および配列と，他の言語におけるそれとの主な差異は，使用法ではなくむしろ形式にある．

PL/I および FORTRAN のような言語では，変数は値を保存する領域に対応して定義される．また，配列は同一形式の一連の副領域から構成される．これらの領域に通常付与される特性は，名前，データ型（整数，文字等），長さ(36 ビット長，10 文字長等)，次元および上下限，記憶域クラス(静的，自動的，ベースつき等)である．

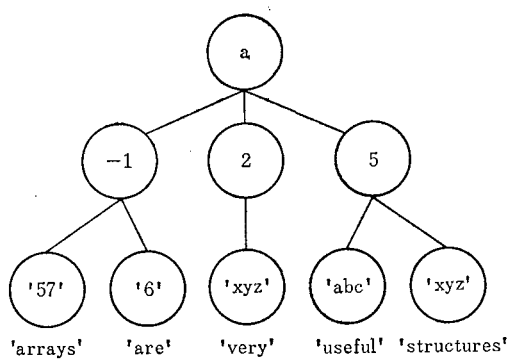
これらの特性は，原始プログラム，言語定義，あるいはコンパイラ作成基準によって決定される特性であり，翻訳時に確定する． MACRO の変数および配列に対しても，これらの特性が適用できる．ただし，これらの特性のすべてが，MACRO プログラムから決まるわけではない．MACRO における変数および配列の型と長さは，SNOBOL 4 と同様，プログラム実行時に代入されているデータのそれと同じである． しかし，MACRO における変数および配列の特性のすべてが，動的に決まるわけでもない．たとえば，記憶域クラスは，言語定義によりつねに自動的クラスである．次元性も翻訳時に決まる．したがって，MACRO における変数または配列の宣言は，非常に簡潔である．

```
DECLARE 〈変数項目〉 [',' 〈変数項目〉 ] … ';'
〈変数項目〉 ::= 〈変数名〉 [ 〈次元並び〉 ]
〈次元並び〉 ::= '(' 〈次元〉 [',' 〈次元〉 ] … ')'
〈次元〉     ::= '*' | '**'
```

変数は，〈次元並び〉を省略することにより，宣言される． 配列は，その形式を指定するための〈次元並び〉を指定することにより宣言される．〈次元並び〉の形式は従来の配列に対する形と幾分異なる．第 1 の点は，MARCO における配列の各次元式には上下限が無い．第 2 に，配列要素の添字は連続的でなくてもよい．実際，MACRO における配列には，値が代入されるまで配列要素が存在しない．第 3 の相異点は，配列の指標が整数に限定されず，文字列であってもよい点である．この目的のために，2 通りの形式の〈次元〉を定義している．記号“*”を指定した次元は，その指標として整数値をとり，記号“**”を指定した次元は，その指標として文字列値をとる．

木構造は，MACRO における配列を説明するうえで便利である．木の葉が配列内の値に対応し，節が添字に対応し，根が配列全体の名前に対応する．この点について例を示すと，次のとおりである．

```
DECLARE a(*, **);
a(2, 'x y z')='very';
a(-1, '6')   ='are';
a(5, 'x y z') ='structures';
a(-1, '57')   ='arrays';
a(5, 'a b c') ='useful';
```



木構造の各レベルは，各次元に対応する．各レベルの節は，順序づけられていることに留意すべきである．記号“*”の次元に対応するレベルは，指標の整数値によって順序づけられる．記号“**”の次元に対応するレベルは，指標の文字列値によって順序づけられる．各々の葉を参照するためには，木を検索してその葉へ到達するための節を表す添字を指定する．さらに MACRO では，その木を検索するための一群の組込み関数を供給し

ている．これらの関数を使用することにより，指定した部分木の最初の節および最後の節，指定した節を基準とした直後の節，そして直前の節の添字を検索することができる．また，配列からその任意の部分木を除去する機能もある．次の例では，これらの関数を使って配列aの木構造をたどり，定義されているすべての配列要素の名前およびその値を印書している．

```
DECLARE a(*, **);
   ...
sub1 = FIRST (a);              // 第1次元の最初の節を得る.
WHILE sub1 NEC '' DO           // 第1次元内で定義した節が無くなるまで,
   BEGIN                       // BEGIN/END ブロックを実行する.
   sub2 = FIRST (b, sub1) ; // sub1 に従属する第2次元の最初の節を得る.
   WHILE sub2 NEC '' DO
      BEGIN
      WRITE 'a(' & sub1 & ',' & sub2 & ')='
            & a(sub1, sub2) ;       // & は連結演算子である.
      sub2 = NEXT (a, sub1, sub2) ; // 第2次元内の次の節を得る.
      END ;
   sub1 = NEXT (a, sub1) ; // 第1次元内の次の節を得る.
   END ;
```

5.4 入出力ストリーム

模様変数および模様配列は，入力ストリームからマクロ本体へ情報を引き渡すために用いる．マクロ本体から出力ストリームへテキストを転送するための補充的メカニズムとして，ANSWER 文の機能がある．これらの処理手順をもう少し注意深く吟味するならば，入力ストリームから出力ストリームへの情報の流れはかなり複雑である．引き金マクロは，語彙が構築されるたびに起動する可能性を持っている．さらに各マクロは，その実行中の各時点で失敗に終わる可能性を持っている．1つのマクロが“不成功”に終わると，それに関連する他の多くのマクロが“不成功”に終わることになる．マクロが成功または失敗に終わった時点の入力ストリームおよび出力ストリームの状況の定義は，MACRO言語の一部として含まれている．

MACRO と入出力ストリームの関係は，さらにいくつかの概念的構成を導入することによって，もっとわかりやすくなる．概念的構成とは，開始ポインタ，現ポインタおよび置換バッファを考えることである．各マクロが起動するたびに，そのマクロ起動に対して開始ポインタ，現ポインタおよび置換バッファが対応づけられる．模様照合の開始時の置換バッファは空であり，2つのポインタはともに，模様照合を開始すべき入力ストリーム中の位置を指している．開始ポインタは，マクロの模様照合の間，この位置を指し続ける．他方，現ポインタは，照合済みテキストと未照合テキストの境の位置を保持するために，入力ストリーム中を前後に移動する．したがって2つのポインタは，模様照合に成功してマクロ本体の実行を開始した時点において，照合に成功したテキストの両端を区切る．マクロ本体の実行を開始すると，置換バッファは，ANSWER 文によって生成されるテキストを収容するための待ち行列となる．引き金マクロの実行が成功に終わると，そのマクロの2つのポインタで区切られた入力ストリーム中のテキストは，そのマクロの置換バッファ内のテキストにより置き換えられる．この処理が終わると，その引き金マクロが起動した時点で他のマクロが実行中であったならば，そのマクロの実行が再開される．実行を再開したマクロの現ポインタは不変であり，入力ストリーム中の置換テキストは，実行を再開したマクロの模様によって走査されるテキストの一部となる．

構文マクロの実行の終わり方は，引き金マクロと異なる．置換バッファ中のテキストはそのマクロの2つのポインタで区切られたテキストを置き換えるが，その構文マクロを起動させたマクロの現ポインタの位置に違いがある．すなわち，起動させたマクロの現ポインタは，その構文マクロの現ポインタの位置まで進む．したがって，置換テキストは再走査されない．この差異は，ちょうどこの2種類のマクロの性質に対応している．引き金マクロは，動的に起動し，入力ストリーム中のセグメントを動的に置換する．構文マクロは，そのマクロを起動させるマクロの模様の拡張部分である．したがって，構文マクロの置換テキストは，それを起動させたマクロによって走査されたテキストの一部分となっている．

マクロが“失敗”に終わると，入力ストリームは，そのマクロが起動する直前の値へ引きもどされる．あるマクロが“失敗”したとき，そのマクロが起動してから“失敗”するまでの間に，起動した引き金マクロまたは構文マクロによって置き換えられた入力ストリーム中の置換テキストは，マクロが起動する直前の状態にもどされる必要がある．引き金マクロが“失敗”に終わると，その引き金マクロが起動しなかったかのごとく，元の模様照合を再開する．構文マクロが“失敗”に終わると，その構文マクロを起動させたマクロへ“失敗”の情報がもどる．この“失敗”の情報は，模様中の同じ位置で語彙，集団またはつづりに対する照合に“失敗”であった場合と同じ効果を持つ．構文マクロに“失敗”したときには，その構文マクロの代替となるものに対する照合が試みられる．代替となるものが無いとき，模様照合を行っているマクロは，“失敗”に終わる．

出力ストリームは，どのように生成されるか．出力ストリームは，入力ストリームを変換したものである．マクロによって照合しなかった部分については，入力ストリームと出力ストリームの内容は同じでそのまま残る．入力ストリームのうち模様として照合したテキストは，初めの引き金マクロが生成した置換テキストによって置き換えられる．初めの引き金マクロとは，活性な引き金マクロが無いとき，最初に起動した引き金マクロである．

6. MACRO 言語の特徴

各々の言語には長所と短所があり，すべての問題を記述するのに最良であるような言語は存在しない，と多くの言語設計者は考えている．MACRO の設計者も同様の認識から特定クラスの問題を解決する際に非常に有用となる言語を定義するように努めた．MACRO には，2つの主な長所がある．まず第1に，文字列取扱い用の機能が供給されている．この機能によって，文字列の解析および合成を必要とする問題を解決するための文は非常に簡明となる．すでに本稿 p. 16 であげた従来の言語と比較して，MACRO は，この分野では非常に高級な言語である．MACRO には，高級クラスのパターン演算子およびオペランド，高級な文字列取扱い用の入出力文，および文字列の効率の良い検索と分類のできる特性を持つ高級な配列が定義されている．第2の長所は，これら従来の言語との整合性があることである．文字列の解析および合成の部分を除いて，MACRO は ALGOL 型言語の構文を採用した．これにより MACRO の学習が容易になるとともに，一般的な拡張性も備えている．すなわち，他の言語の構文に対応させることにより，MACRO の手続き呼出しでこれらの言語と連絡をとり，一般拡張性をもたらすことが期待される．

MACRO 言語にも短所がある．たとえば，マクロ起動の概念により，MACRO 原始プログラムを複数の翻訳単位に分けることはむずかしい．実際，UNIVAC シリーズ 1100 MACRO においては，複数の翻訳単位をリンケージ・エディタにより集めて1つのプロ

グラムにする機能は，現在供給されていない．問題の難点は，構築されたり起動したりするであろうすべての引き金語彙，語彙および引き金マクロを識別する主テーブルが必要となることである．この情報は，語彙構築やディスパッチングの機能で必要となる．複数の翻訳単位を許すためには，各単位からの情報を集める仕組が必要である．

7. シリーズ 1100 MACRO

MACRO 言語システムは UNIVAC シリーズ 1100 のもとで，使用可能である．システムを作成する際，可能な範囲で実行速度を犠牲にするかわりに，プログラム開発の期間を極力短縮できるような文字列取扱い用システムを供給することを主眼とした．一般に主だった文字列取扱い用アプリケーションは，頻繁には使われないプログラムを開発するためのものである．UNIVAC シリーズ 1100 MACRO も，この種のアプリケーション処理を目標としている．

UNIVAC シリーズ 1100 MACRO システムは，コンパイラとインタプリタから構成されている．コンパイラは，リロケータブル・エレメントを生成せずに，インタプリタが直接解析できるテキストを生成する．プログラム準備のために通常行う連結編集処理は完全に省略できる．UNIVAC シリーズ 1100 コレクタの処理を省略することによる時間の節約に加えて，インタプリタの実行中の高度なデバッグおよび追跡情報を供給することにより，開発に要する時間を最小限にしている．インタプリタは，MACRO の構文を認識していて，原始プログラムの文脈の中の追跡およびデバッグ情報を把握することができる．

複数の引き金マクロの間の関係は，従来の言語で書いたプログラム中の手続きの間の関係のように，厳密に MACRO プログラムで定義されない．しかし，引き金マクロの間におけるこの弱い依存関係はまた，プログラム開発の改善に役立つ．手続きの場合には，各手続きが，どの手続きを呼び出し，プログラム中のどこから呼び出されるかを明記する．この情報はすべて，原始プログラム中に現れる．また，ある手続きを追加することは，プログラムにおいてその手続きをその他の手続きと統合することになる．しかしながら，MACRO プログラムにおいて引き金マクロは，互いに他を参照しない．それらの引き金マクロの相互作用は，入力ストリームから検出される語彙によって制御される．この独立性により，MACRO プログラムを段階的に構築することが容易となる．すなわち，1つのマクロを設計し，コーディングし，デバッグした後に，他のマクロを設計して，2つをまとめて一緒にテストすることができる．さらに，引き金マクロの独立性により，あるマクロを追加または削除するときに，原始プログラムの変更が最小限ですむ．このような技法は，MACRO を使ったプログラムの設計に対して，一般に使用できる．

8. おわりに

MACRO で作成したアプリケーションには，様々なものがある．MACRO プログラムは，UNIVAC の構文解析部生成システムである GSA (General Syntax Analyzer)[2]に対して書かれた文脈自由文法を入力として受け，MACRO 言語の規則に適合する文法を生成する．構文の変換に加えて，COMADS として知られる自動文書化システムに対しては書式指示も挿入されている．MACRO で書かれたその他のアプリケーションとしては，IBM PL/I プログラムから標準に準拠した PL/I プログラムへの変換用プロセッサ，UNIVAC の FORTRAN 用 DMS/1100 データベース指示文から標準 FORTRAN の CALL 文への変換用プロセッサ，DOC とか COMADS のような各種文書作成用プロセ

ッサのための原始言語間のトランスレータ，および原始プログラムから各種の情報を摘出するプロセッサ等がある．さらに，UNIVAC シリーズ 1100 エディタの拡張版としても，MACRO が使用されている．エディタで容易に認識できないパターンに対しては，短い MACRO プログラムを作成すればよい．

コンピュータ利用の費用のうちで，ソフトウェア開発費用が占める割合は，時とともに増大している．この費用増大を抑える努力の過程で，高級言語が，ますますソフトウェア開発のために必要とされている．高級言語を用いることにより，人員の教育，製品の作成，保守および他の処理系への変換のための費用を削減することができる．MACRO は，このようなソフトウェア経費削減の手段として，とくに適している．MACRO 言語の長所は，高級言語の立場から文字列の取扱いに取り組んでいることであり，言語の使用が簡便になった点にある．また，高級言語であるとともに，従来の形式にも合致するという点からも，MACROは将来，ソフトウェア製品の中で確かな地位を占めるであろう．

（システム・ソフトウェア開発部　山下　喬樹 訳）

参考文献 [1] MACRO 解説書，日本ユニバック，1970.
[2] T. N. Turba, "General Syntax Analyzer (GSA)," *ACM SIGPLAN NOTICES*, Vol. 14, No. 12, Dec. 1979, pp. 92-109. (本誌 pp. 42 を参照)

執筆者紹介 S. R. グリーンウッド (Stephen R. Greenwood)
1975年 Wisconsin 大学において M. S. を取得．以後 Sperry Univac 社の Roseville 開発センターの言語システム・グループの一員として開発に従事．PL/I および PLUS のコード生成，MACRO 言語の主要機能の設計および作成，ならびに，CODASYL FORTRAN データベース操作言語の標準化に参加．現在 PL/I 実行時ライブラリに関するスーパバイサ．

訳者注 現在，日本ユニバック(株)が提供している MACRO のリリース・レベルは 6R1 であるが，本稿はそれ以前のレベルに基づいて書かれている．

報告 高水準会話型デバッギング・システム AIDS

The Advanced Interactive Debugging System

J.J. Hart

要 約 AIDS (Advanced Interactive Debugging System) は，どの高水準言語で記述されたプログラムでもデバッグすることのできる単一システムであり，会話型によりデバッグするための強力で融通性のあるツールとなっている．これには，多数の機能が備わっている．業務プログラムについても誤りが発生した時点で，事前の準備なしに，またプログラムの再コンパイルや再実行を行わずに使用できる．すなわち，AIDS を含む環境内にその辞書(記号表)を用意するだけで，機能が付与できる．さらに，デバッグ指示文の集合や高級言語の手続きを使用することによって，機能の拡張も可能である．

Abstract The Advanced Interactive Debugging System (AIDS) is a powerful and versatile tool for the symbolic interactive debugging of programs written in high-level languages. It provides a multitude of functions. It can be used with production programs at the point of failure without requiring prior use, and without requiring recompilation or re-execution of the program. It can be extended by the use of sets of debugging commands and high-level language routines.

1. はじめに

UNIVAC では，高級言語の互換性を目的とする UCS (万能コンパイラ系)[1]の一部として，原始プログラムを会話型でデバッグする言語に独立な手順を考え出した．これは AIDS と名付けられ，次のような特色を持つ．

1) UCS により目的プログラムに組み込まれた AIDS の記号表から，原始プログラム中の名前による参照ができる．また，機械上の番地のような名前以外による参照は記号表の有無にかかわらず可能である．
2) デバッグの方法が言語にかかわらず1種類の方法ですむよう，指示文はすべての言語に共通である．
3) 異常の捕獲，データの表示，プログラムの追跡などの基本的機能が簡単な指示文で行え，使用者の技能および要求に応じた使用ができる．
4) プログラムの構造上，セグメント化，多重タスク化に対応して，いずれでも同じ様にデバッグすることができる．
5) バッチでも会話型でも使用でき，また，その指示文は外部ファイルからでも端末装置からでも入力できる．
6) プログラムの実行開始前に呼び出すことができる．
7) 事前に準備されていなくても，プログラムに誤りが発生した時点で自動的に呼び出される．
8) 原始プログラム中のデバッグのための文や，コレクション時の特別な指示が不要である．
9) デバッグが開始されない間は，デバッグされるプログラムの効率は低下しない．ま

Reprinted from ACM SIGPLAN NOTICES, (December 1979), Vol. 14, No. 12, pp. 110-112.

た，AIDS の存在によってプログラムの大きさや動作が変化せず一定である．

10) AIDS は主に UNIVAC のシステム記述言語 PLUS[2]で記述され，GSA (Generalized Syntax Analyzer)[3] を用い，これにより移植性を保っている．

11) 機能と指示文は，基本機能，デバッグ制御機能，拡張機能，その他の機能（編集機能，補助情報機能）の4つに分類される．

12) 使用が容易で，融通性が高い．

なお，AIDS は現在のところ，顧客には提供されていない．

2. 基 本 機 能

基本機能としては，以下の8つの機能がある．

2.1 プログラムの実行の中断

プログラムの実行をある条件のもとで中断するために，3種類の指示文が用意されている．ブレークポイント，データ参照補捉，異常捕獲である．プログラムの中断が起こると，これら3種類のどれかと組み合わされた指示文の並びがあれば，それらが実行される．その後プログラムの実行を再開する指示文が実行されるまで，AIDS は入力指示文並びからの指示文を要求する．一方，ブレークポイント，捕捉または捕獲に対しての指示文が組み合わされていないと，AIDS はその中断を識別する句と中断の原因となったプログラムの番地，データの番地または異常を表すメッセージを出力する．その後，入力指示文並びから指示文の入力を要求する．

中断の指示を解除するために CLEAR 指示文が用意されている．この指示文はすべてまたは一部のブレークポイント，データ参照補捉および異常捕獲を解除する．

上記以外でプログラムの実行を中断する4番目の方法として，端末のけん盤から中断する方法がある．この機能を使用するためには，実行中のプログラムに関する知識をまったく必要とせず，その時点で即座にプログラムの実行を中断させることができる．

2.1.1 ブレークポイント

ブレークポイントをプログラム中に設定することによってプログラムを中断し，AIDS に制御が移るようにできる．ブレークポイントのための指示文には，BREAK，SETBP，STEP の3種類がある．

BREAK 指示文の一般形は次のとおりである．

BREAK *id* AT *location* BEGIN *stmt* END

ここで，*id* はデバッグ用の句で省略できる．*location* はプログラム中の行番号，プログラム中のラベルまたは命令語の番地である．BEGIN *stmt* END 節も省略可能である．*stmt* は，ブレークポイントに設定された場所の実行後に行われるデバッグ指示文の並びである．

SETBP 指示文も BREAK 指示文と同様のものである．指示文に指定されたオプションに応じて，実行される場所ではなく，データの参照に対してブレークポイントを設定する．この機能は，ハードウェア・ブレークポイント・レジスタを利用するため，記憶領域上の語単位の参照しか対象にできず，また，このようなレジスタを持たない機械では使用できない．

STEP 指示文は，ブレークポイントの一般化された形である．現在の場所から AIDS に制御が移るまでに実行される原始プログラムの文の数を指定する．AIDS に制御がもどるとさらに指示文の入力を要求する．

2.1.2 データ参照補捉

データ参照補捉とは，指定されたデータ項目または番地が参照されたときに中断が起こるようにするものである．読み込みのため，または変更のため等の参照の種類をデータ参照補捉の設定時に指定することができる．また，ブレークポイントと同様に，中断が起きたときに実行される指示文の並びを指定することもできる．

データ参照補捉の指示は，主に TRAP 指示文で行い，その一般形は次のとおりである．

TRAP *type id* AT *location* BEGIN *stmt* END

ここで，*type* は中断を起こす参照の種類を表す．他は中断のときと同様の意味を表す．

設定された参照は，命令語を解釈することで見つけられる．AIDS はそのときに有効な TRAP 指示文に指定されている参照を探して，命令語をチェックする．そのような参照の指定がない命令語は平常どおりに実行される．

SETBP 指示文も，データ参照補捉に使用することができる．しかし，SETBP 指示文は語単位の参照に限られている．また，ハードウェア・ブレークポイント・レジスタを使用するので，同時には1つだけのブレークポイントまたはデータ参照補捉しか設定できない．

2.1.3 異常捕獲

ハードウェア・エラー状態が発生すると，AIDS は自動的に制御を得て指示文の入力を要求する．使用者も，発生時に AIDS に制御が移るようなプログラムの異常を指定することができる．これらの異常は，言語の標準仕様のエラー処理ルーチンや使用者プログラム内のルーチンで発見されるものと同じでもよい．異常はまず ADIS で，つぎに必要ならば，言語または使用者のルーチンで取り扱われる．

異常捕獲のための CAPTURE 指示文の一般形は次のとおりである．

CAPTURE *exception* BEGIN *stmt* END

ここで，*exception* は，桁あふれ，下位桁あふれ，除算障害，添字範囲違反などの異常で，BEGIN *stmt* END 節は省略可能な指示文の並びである．

2.2 データの表示

デバッグ中にプログラムの記憶領域や，レジスタの内容が表示できると都合がよい．原始プログラム中の名前でデータの表示を指定した場合には，その内容は原始プログラム中での項目の属性に従って編集される．機械上の番地で指定した場合には，その内容を使用者の指定で編集してもよい．

AIDS からの出力には4種類ある．①情報，エラーまたは警告を知らせるメッセージ，②原始プログラム中の属性に従って編集された変数，③機械上の番地で指定され使用者の指定で編集された項目，④変数の属性と番地の情報，である．メッセージは AIDS が必要に応じて生成する．その他の出力は，特定の指示文による．

2.2.1 変数の表示

変数の出力には，主に DISPLAY 指示文が使用される．その一般形には2種類あり，次のとおりである．

1) DISPLAY NAMED *item*, *item*, ...

2) DISPLAY ALL VARIABLES

一般形 1) はいくつかの変数を選んで表示するために使用され，2) は実行中のプログラムまたは特定の副プログラム中のすべての変数を表示するために使用される．1) での NAMED 句は，出力の際に原始プログラム中での名前をつけて表示することを指示する

手掛り語で省略可能である．*item* は文字列またはプログラム中の変数名である．もし変数が構造体であれば，その指定に BY ITEM 句を含むことができる．この場合の出力は，基本項目が個々の様式で編集された変数である．2）の形式を使用するときには，変数の表示をするコンパイル単位を選択することができる．

2.2.2 記憶領域のダンプ

記憶領域の内容表示の際に，その場所を機械上の番地またはレジスタを指定して参照したり，変数に特定の編集指定ができると都合がよい．これらの機能が DUMP 指示文に備えられている．その一般形は次のとおりである．

DUMP *address-list* FORMAT *format*,
address-list FORMAT *format*, ...

ここで，*address-list* は1個以上の番地，レジスタ名または変数名である．FORMAT *format* 節は省略してもよい．*format* は，その項目の編集方法を指示する書式の並びである．そこには出力される文字列が含まれていてもよい．*address-list* に続く *format* がない場合には，その内容は機械に特定の編集がされる（たとえば，UNIVAC シリーズ 1100 では8進数）．

編集を指示する書式は FORMAT 指示文によって定義されていてもよい．その一般形は次のとおりである．

FORMAT *name format*

ここで，定義された *format* は，DUMP 指示文から *name* によって参照することができ，また他のデバッグ・セッションでも使用できるよう保存される．

2.2.3 属性情報

変数や番地に関する属性を INQUIRE 指示文によって得ることができる．その一般形は次のとおりである．

1) INQUIRE VARIABLE *variable*
2) INQUIRE ADDRESS *address*

1）によって変数の属性や原始プログラム中での配置に関する情報が，2）によってその番地に関する機械に特定の記述が出力される．

2.3 記憶領域の変更

デバッグ中に記憶領域内の値が変更できると都合がよい場合がある．これは，誤った値を修正したり，プログラムの一部分の再実行の前に値を変えたり，プログラムの部分的実行の前に変数の初期値を与えたりするためである．この機能のために，名前による参照のためのものと，番地による参照のためのものの2種類の指示文がある．

2.3.1 変数への代入

変数は主に ASSIGN 指示文によって修正される．その一般形は次のとおりである．

ASSIGN *target-item*:=*source-item*

ここで，*target-item* はデバッグ用変数名，または変数名で *source-item* は文字列，数値定数，デバッグ用変数，変数名，番地，または定数とデバッグ用変数からなる式である．

この指示文によって，*target-item* の値は記憶領域上の *source-item* の値に，記憶領域上で置き換えられる．記憶領域上での修正は，その変数の以後の使用に反映されない場合がある．もし記憶領域上の修正された項目が変更されたときにその値がレジスタ内に保存されていたならば，その後の変数の参照にレジスタの内容が使用されたり，記憶領域上の変数にかぶせられたりするおそれがあるためである．

2.3.2 機械上の番地による変更

番地の参照による記憶領域上での変更で，主に使用される ALTER 指示文の一般形は次のとおりである．

ALTER *address*:=*source-item*

ここで，*source-item* は定数でも番地でもよい．

この指示文によって，目的の番地の内容が右辺の番地の内容，または定数に変更される．

2.4 プログラムの実行の追跡

流れの追跡は，TRACE 指示文によってラベル，文，手続きの呼出し，に対して指定して行われる．また，必要に応じて TRACE 指示文の対象となるコンパイル単位も指定できる．

指示文により追跡が開始されると，指定されている項目が現れるのを見つけるために，命令語が調べられる．命令語にその追跡項目がなければ平常どおりに実行される．もし見つかれば，メッセージが出力され，プログラムの実行は続けられる．ラベルと手続きの呼び出しのためのメッセージには，それに出合う直前に実行された文を示す情報も含まれている．

2.5 統 計 収 集

文，ラベル，手続き呼出し，ライブラリ呼出し，そして入出力機能について，その使用回数の集計と参照リストが作成できる．

2.6 デバッグ用変数

使用者がデバッグ用変数の定義ができる．デバッグ用変数は，デバッグ手続きやコンパイルされた手続きへ値を渡したり，指示文の実行を条件によって制御するために使用する．また，デバッグ用変数はある値を一時的に保存するためにも役立つ．これらによって AIDS の多くの機能が促進できる．

デバッグ用変数は，広域的に（すべてのデバッグ・セッションに）も，局所的に（特定のデバッグ手続きに）も定義できるよう，融通のきくものとなっている．また，可変長の文字型としても1語または2語の符号つき整数型としても定義できる．後者の場合には，算術式の一部となったり，番地や変数の値を代入したりできる．また，デバッグ用変数は1次元の配列として定義することもできる．

2.7 プログラムの実行の再開

プログラムの実行は，AIDS が制御を得た次の実行番地，指定されたラベルやプログラムの行番号，番地，またはハードウェア・コンティンジェンシや異常捕獲の起きた位置から，再開できる．これによって，コンティンジェンシや異常に対する言語の処理が避けられる．

2.8 プログラムの実行の終了

プログラムの実行は，いかなる場合にも終了させることができる．この場合，デバッグ・セッションは自動的に終了する．多重タスク・プログラムの個々のタスクを終了させることもできるが，この場合はデバッグ・セッションは自動的に終了することはない．

プログラムの実行を終了させずに，デバッグ・セッションを終了させることができる．そのためには，すべてのブレークポイント，データ参照補捉，異常捕獲を解除して，追跡と統計収集を終了させればよい．

3. デバッグ制御機能

デバッグ制御機能として，次の機能がある．

3.1 デバッグ・セッションの呼出し

デバッグ・セッションは3とおりの方法で呼び出せる．すなわち，①プログラムの実行前に使用者が呼び出すか，②ハードウェア・エラー状態の発生によって自動的に呼び出すか，③プログラムから直接に呼び出すか，である．①の呼出しは，プログラムのテスト段階での有効な方法である．テスト中は，プログラムの一部分または全体を監視するために，ブレークポイントを設定する場所をあらかじめ選ぶことも容易だからである．②の呼出し機能は，どのような場合にも有効であるが，誤りがない業務プログラムにとってはとくに重要である．この場合，誤りが発生するまではデバッグについても何も考慮する必要がない．誤りが発生した場合，プログラムの再コンパイル，再コレクション，再実行なしでデバッグが開始される．③の機能は，テストにも業務プログラムにも有効である．テスト中であれば，ブレークポイントの指示文を使用せずに，ブレークポイントを設定することになる．業務プログラムにとっては，異常捕獲の拡張が可能となる．

プログラムの実行開始前にデバッグ・セッションが呼び出されたときも，ハードウェア・エラーが発見されたときとプログラムの終了したときに，自動的に AIDS が制御を得る．

3.2 指示文の条件による実行

IF 指示文を用いることにより，デバッグ指示文を条件付き実行にすることができる．IF 指示文の一般形は多くの言語と同様で，次のとおりである．

IF...THEN...ELSE

ここで，ELSE 節は省略してもよい．テストされる条件は変数，番地，デバッグ用変数そして関数引用を演算要素とした関係式である．演算子は“<”(未満)や“=”(等価)のような比較演算子である．さらに，関係式を論理演算子 AND または OR と組み合わせることもできる．THEN 節と ELSE 節は対応する分岐によって実行される指示文の並びを含んでいる．IF 指示文は入れ子にでき，両方または片方の指示文の並びに，他の IF 指示文があってもかまわない．

3.3 指示文の繰返し実行

AIDS には，指示文を繰り返して実行する機能がある．そのためには DO 指示文を使用するが，その一般形には3種類ある．それぞれは次のとおりである．

1) DO *debugger-symbol=initial-value*
   TO *final-value* BY *increment-value*
   BEGIN *stmt* END
2) DO WHILE *relation* BEGIN *stmt* END
3) DO *debugger-symbol=initial-value*
   REPEAT (*repeat-value*) WHILE *relation*
   BEGIN *stmt* END

1) の形は FORTRAN や，PL/I のようなプログラミング言語に共通の増加形である．2) は DO WHILE 形と呼ばれるものである． 3) は REPEAT 形と呼ばれるもので，連結された記述をたどるためにとくに有効な指示文である．

一般に，上に現れている *value* は定数，または定数とデバッグ用変数からなる式である．しかし，REPEAT 形の *repeat-value* は常に式である．増加形での BY *increment-value* 句と，REPEAT 形での WHILE *relation* 句は省略してもよい．それぞれの繰返しの範囲は，指定された指示文の並びによって定義されている．DO 指示文も入れ子にすることができ，指示文の並びの中に DO 指示文が含まれていてよい．

3.4 分　　岐

デバッグ手続き中の指示文の実行を，前または後へ方向づけることができる．このような指示は，JUMP 指示文によって行われる．一般形は次のとおりである．

JUMP *debugger-label*

ここで，*debugger-label* は JUMP 指示文と同じ手続き内の指示文につけられたラベルである．分岐は，IF 指示文と組み合わせて条件付き分岐とすると一層有効である．

3.5 入力並びの指定

デバッグ中に，デバッグ指示文を標準の入力並びからでなく外部ファイルから入力することができる．DIRECT INPUT 指示文を使用すると，デバッグ指示文だけでなく，使用者プログラムにおける入力要求に対する入力源も指定することができる．AIDS およびプログラム中での入力要求に対する入力源をそれぞれ別々に，外部ファイルとしたり，標準にもどしたりすることができる．

3.6 出力先の指定

プログラムおよびデバッグの出力のために，DIRECT OUTPUT 指示文を使用して，種々の装置や印書装置を指定することができる．デバッグの出力の指定では，主に DISPLAY 指示文，DUMP 指示文，TRACE 指示文による情報の出力場所を指定することになる．プログラムの出力の指示では，会話型端末へ出力されるはずであった出力場所を指定することになる．この機能は，デバッグの出力が大量であると考えられる場合や，プログラムとデバッグの出力を分離したい場合に役立つ．

3.7 デバッグ・セッションの終了

デバッグ・セッションの終了は，すべてのブレークポイント，データ参照補捉，異常捕獲を解除し，追跡と統計収集を終了させることによって行われる．また，AIDS によってプログラムを正常にまたは異常に終了させることで，デバッグ・セッションを終了させることができる．

4. 拡 張 機 能

拡張機能としては次の機能がある．

4.1 デバッグ手続き

デバッグ手続きとは，デバッグ指示文の集合である．手続きは，1つのデバッグ・セッションの中で繰り返し使用したり，他のデバッグ・セッションで続けて使用したりすることもできる．デバッグ手続きに引き数を渡すことも，デバッグ手続きから引き数または関数引用の結果として値をもどすこともできる．デバッグ手続きの利用は，すでにある機能を拡張するための有効な手段である．

デバッグ手続きのための指示文は，手続きの定義，手続きの外部ファイルへの保存，手続きの外部ファイルからの回復，手続きの呼出しの4つに分類することができる．

デバッグ手続きを定義するには2つの方法がある．まず，手続きの本体を UNIVAC の標準のテキスト編集プログラムを使用して作成し，ファイルにシンボリック・エレメント（原始モジュール）として保存することができる．使用者はその手続きをデバッグ・セッションの中から回復し，使用することができる．また，デバッグ手続きをデバッグ・セッションの中で定義することもできる．これは PROC 指示文を使用して行われ，その一般形は次のとおりである．

PROC *name* (*parameter-list*)

ここで, *parameter-list* は仮引き数となるデバッグ用変数の並びであり，省略できる．PROC 指示文によって AIDS は手続き定義モードに入り，手続きの本体となるデバッグ指示文を要求する．この要求は，END PROC 指示文に出合うまで続けられる．もし手続きの本体に対して何らかの変更が必要であれば，AIDS のシンボリック・テキスト編集機能によって変更する(5章参照)．手続きが実行されるときには，仮引き数がサブルーチンまたは関数の引用の実引き数とその位置によって結びつけられる．手続きとその引用とで，引き数の属性の一致を保証することは，使用者の責任となる．手続きの本体に PROC 指示文を含めることはできない．また，直接にも間接にも手続きの再帰的呼出しはできない．

デバッグ手続きを外部ファイルに保存するためには，PUT 指示文が使用され，手続きを外部ファイルから回復するためには GET 指示文が用いられる．これらの指示文は，FORMAT 指示文で定義されている書式の指定を保存，回復するためにも使用される．

デバッグ手続きの呼出しは，CALL 指示文を使用してサブルーチンとして呼び出す方法と，関数の引用として手続きの名前を使用する方法の2つがある．CALL 指示文の一般形は次のとおりである．

CALL *name* (*parameter-list*)

ここで，*parameter-list* は，変数，デバッグ用変数，番地，文字列の並びで，括弧とともに省略することもできる．関数の呼出しは，その名前の参照，必要ならば引き数並びをつけた参照により行われる．演算後，関数はその呼出しの位置に値をもどす．

デバッグ手続きはサブルーチンでも関数でもよいし，同時にその両方であってもよい．手続きから，その呼出しの場所へ制御をもどす RETURN 指示文が，それを左右する．RETURN 指示文の一般形は次のとおりである．

RETURN *debugger-symbol*

ここで，*debugger-symbol* は省略できる．もし指定されていれば，その値が関数値としてもどされる．手続きには RETURN 指示文をいくつでも指定でき，それぞれは値をもどしてももどさなくてもよい．もし手続きがサブルーチンとして呼び出され，値をもどす RETURN 指示文が実行されると，その値は無視される．もし手続きが関数として呼び出され，値をもどさない RETURN 指示文が実行されると，2進数のゼロが関数値としてもどされる．

4.2 コンパイルされた手続き

コンパイルされた手続きも，CALL 指示文と関数呼出しの対象とすることができる．引き数もデバッグ・セッションと，その手続きとの間で受け渡しができる．コンパイルされた手続きも，再帰的には呼び出せない．

コンパイルされた手続きを呼び出すためには，CALL 指示文または関数引用に使用する名前は，処理可能な入口名でなければならない．すなわち，AIDS のシンボル・テーブルを利用できる手続き中の入口名でなければならず，またデバッグ中のプログラムと一緒に連結編集されている必要がある．さらに，手続きを記述している言語の実行時の環境が，呼出しの前に整えられていなければならない．通常，デバッグ中のプログラムとその手続きは同じ言語で記述されているので，実行時の環境はデバッグ中のプログラムの実行が始まったときに整えられている．その他の場合でも，適切な実行時の環境を整えることは，デバッグ中のプログラムからその言語で記述された手続きを呼び出すというような簡単なことで可能となる．

デバッグ・セッションからコンパイルされた手続きを呼び出すことは，いろいろな面で

便利なことがある．たとえば，サブルーチンの単体テストを行ったり，プログラム中の手続きの代用としたりするときに，この呼出しが利用できる．

5. おわりに

以上に述べたように，AIDS はデバッグのための強力で融通性のあるツールとして多くの機能を備えている．その他の機能として，シンボリック・テキスト編集機能，プログラムとデバッグ・セッションに関する情報を得る機能，デバッグ指示文の文法と使用法についての補助機能がある．

シンボリック・テキスト編集機能のために，EDIT 指示文と編集のための副指示文が用意されている．EDIT 指示文に指定された手続き，ファイル，エレメントに対して，編集のための副指示文に示された行の挿入，削除，置換え，シンボリック・テキストにおける行番号の振直しが行われる．

指定されたプログラムまたはデバッグ・セッションについての情報を得る機能が，単純な INQUIRE 指示文として備わっている．この機能により，すべての有効なブレークポイント，データ参照補捉，異常捕獲のリスト，原始プログラム中の変数の属性と配置，プログラム中の副プログラムの流れの記述，ハードウェア・ジャンプ・ヒストリー・スタックの内容，そして AIDS が最後に制御を得たときに出力された情報メッセージなどの情報が得られる．

HELP 指示文により，すべてのデバッグ指示文，指定された指示文，指示文を形成するための基本的文法要素と，それぞれの指示文に特有な要素とについての文法とその使用法の表示が得られる．この他，疑問符“?”を入力する方法もある．

（システム・ソフトウェア開発部　大園　茂生 訳）

参考文献 [1] H. C. Gyllstrom, *et al.*, “The Universal Compiling System,” *ACM SIGPLAN NOTICES*, Vol. 14, No. 12, Dec. 1979, pp. 64-70. “言語処理系の生産技術 UCS,” 技報, 1981年2月, No. 0, pp. 3-10.

[2] F. W. Stodola, “The PLUS Programming Language,” *ACM SIGPLAN NOTICES*, Vol. 15, No. 1, Jan. 1980, pp. 146-155. “システム記述言語 PLUS,” 技報, 1981年2月, No. 0, pp. 11-19.

[3] T. N. Turba, “General Syntax Analyzer (GSA),” *ACM SIGPLAN NOTICES*, Vol. 14, No. 12, Dec. 1979, pp. 92-109. (本誌 pp. 42 を参照)

執筆者紹介 J. J. ハート (Jolene J. Hart)

1971年米国 Minnesota 大学において数学で B. A. を取得．以後，ビジネス・アプリケーション分野および健康管理システムのプログラミングにたずさわる．1977年に Sperry Univac 社に入社．1978年に Minnesota 大学おいてコンピュータ・サイエンスにより M. S. を修得．現在，Sperry Uhivac 社ソフトウェア開発センター (Roseville) の言語システム開発部に所属．

訳者追記 日本ユニバック (株) では，AIDS を PADS (Programmers Advanced Debugging System) という名称で提供する予定である．AIDS における機能や指示文の文法は，PADS において，OS 1100 の思想にもとづき，商品ソフトウェアとして適するように改善される．

報告 汎用構文解析プログラム GSA

General Syntax Analyzer (GSA)

T. N. Turba

要 約 汎用構文解析プログラム (General Syntax Analyzer, GSA) はコンピュータ言語の走査を必要とする言語プロセッサ, テキスト・プロセッサおよび他のアプリケーションの走査プログラムを構築する際に用いる道具である. GSA はソフトウェア生産の道具として使用されるため, 誤り回復 (error recovery) の問題と同様に, 場所と時間の効率の問題も考慮されている. GSA で用いられているパージング (parsing) 方法は, パース (parse) の先読みおよび意味制御 (semantic control) が可能な拡張された下降型パーサ (top-down parser) である. このシステムは現在数多くの Sperry Univac 社, 日本ユニバック(株)のソフトウェア生産に使用され, 今後の新しい言語プロセッサ作成にも使用される.

本稿では GSA の言語仕様を中心に述べる.

Abstract The General Syntax Analyzer is a tool used in building scanners for language processors, text processors, and other applications that require the scanning of a computer language. GSA was designed for a production environment, and as such, addresses the problems of space and time effeciency as well as the problems of error recovery. The prasing method used in GSA is on extended top-down parser that allows look-ahead and semantic control of the parse. This system is currently used in a number of Sperry Univac/Nippon Univac Kaisha products and will be used in the construction of new language processors.

1. はじめに

このソフトウェアは汎用の構文解析 (General Syntax Analysis) を行い, 広範囲の言語プロセッサの構築の際, フロント・エンドとして使用される. このシステムは次のようなプロセッサの作成において有効である.

1) コンパイラおよびアセンブラ
2) データ・マネジメント言語
3) ハードウェアおよびファームウェア言語
4) テキスト編集およびドキュメンテーション言語
5) オンライン制御言語
6) シミュレーション言語
7) 専門的問題解決言語
8) 言語型式の入力の走査を必要とするアプリケーション

また, このシステムは MACRO プロセッサ, PLUS コンパイラ, FORTRAN プリプロセッサ (BFTN), PL/I DML プリプロセッサ, SDDL (Subschema Data Definition Language), QLP (Query Language Processor) 等の実装に使用された.

このシステムの使用による利点としては, 次のものがある.

1) 作成時間の短縮および費用の削減

Reprinted from ACM SIGPLAN NOTICES, (December 1979), Vol. 14, No. 12, pp. 92-109.

2) 誤り確率の減少
3) 言語仕様の容易な変更および更新
4) 仕様として単純化した設計によるモジュール化の促進
5) 高度の機械独立性
6) 表の使用によるコードの縮少

次に示した表は，GSA を使用して構築された各種の構文走査プログラムおよびコンパイラのフロント・エンドのパーサ用の表および誤り回復用の表のおおよその大きさである．

言　　語	表の大きさ（語数）
ALGOL 注1)	1175
COBOL (ANSI 74) 注2)	3176
FORTRAN (ANSI 78) 注3)	1811
MACRO 注1)	1218
PASCAL 注2)	1006
PL/I 注2)	3657
PLUS 注3)	3080

注 1) 誤り回復および意味呼出し (semantic call) を含んだ完全な言語仕様である．
注 2) 完全な言語の走査プログラムだが，最少の誤り回復のみで意味呼出しがない．
注 3) 広範囲な誤り回復を含んだ完全な言語の走査プログラムだが，意味呼出しがない．

2. 概　　観

図 1 は語彙表，手掛り語表，言語表，メッセージ表の生成フェーズを含んだ GSA の概

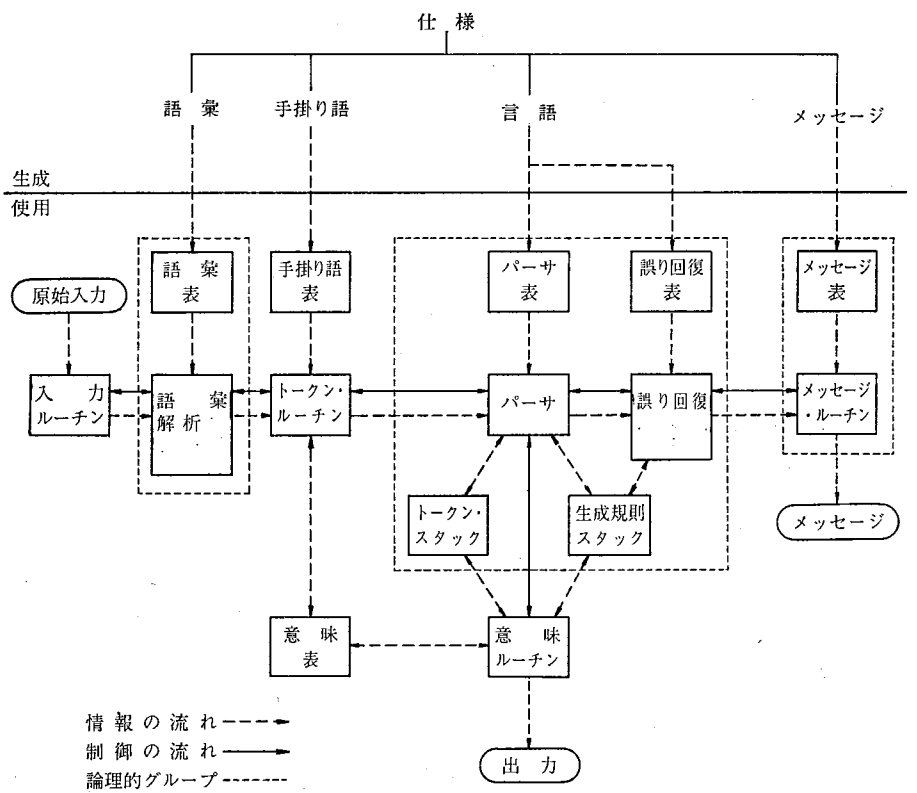


図 1　GSA の概念図
Fig. 1　GSA overview

念図である．これらの表の生成は簡単な仕様からリロケータブル表を生成する特定のプロセッサによって行う．これらの仕様は最終プロセッサの機械に独立な部分であり，その機械用の出力表 (output table) がそれらの仕様に対して生成される．

図1に示したように，システムはモジュラ構成である．語彙仕様は識別子 (identifier), 数，リテラル等の語の単位を定義する．手掛り語仕様は手掛り語およびその同義語とパース番号 (parse number) との対応づけを定義する．言語仕様は文，句，式，プログラム，言語の構成要素を形成するための文法を定義する．メッセージ仕様はプロセッサが出力する誤り，警告等のメッセージを定義する．

さて，言語に独立なルーチンは入力ルーチン，語彙解析プログラム，パーサである．入力ルーチンは語彙解析プログラムとシステムとの標準インタフェースである．入力を行イメージとして受け取り，行やファイルの終了および入力の誤り等を表す特殊文字を付加して，文字単位で語彙解析プログラムへ渡す．語彙解析プログラムは語彙仕様でつくられた表(語彙表)を用いて，語彙の単位を組み立て，分類した後に，それをトークン・ルーチンに渡す．パーサは言語仕様からつくられた表(パーサ表)を用いてパージングを制御する先読み下降型パーサである．これは基本的には認識，制御および処理の命令からなる命令群を持つインタプリタである．誤り回復は統合的にパーサに関係している．誤り回復表は言語仕様から生成され，また誤り回復は言語仕様の統合的な一部分である．

言語に従属なルーチンにはトークン・ルーチン，意味ルーチン，メッセージ・ルーチン，ドライバ・ルーチンがある．トークン・ルーチンは語彙の単位を言語の単位へ対応づけるパーサと語彙解析プログラムとのインタフェースである．意味ルーチンはパーサから呼び出され，意味解析を行う．これらのルーチンはパーサにあるトークン・スタック，および存在している意味表または辞書に納められている情報をアクセスする．メッセージ・ルーチンは印書すべき誤り，警告等のメッセージを組み立てる．また，ドライバ（図1には明示されていない)はプロセッサの初期状態設定を行い，パーサへ制御を渡す．

3. 表構築プロセッサ

表構築プロセッサ (table constructor) は，パーサおよび語彙解析プログラムで使用される表を生成するプロセッサである．

表構築プロセッサへの入力は，次のような予約記号で始まる複数の部門からなる．ほとんどの部門は任意であり，必要なときだけ存在すればよい．部門は，名前が用いられる前にすべて定義されている限り，どんな順序で現れてもよい．

3.1 選択項目指定

選択項目指定部門(予約語：SOPTS) は，仕様(語彙または言語)で用いられる選択項目を示す．

```
[例] $OPTS
        LEX     "lexical specification
        ASCII   "ASCII character set
        CODE    "assembly language output
```

3.2 終端記号定義

終端記号定義(予約語：$TERM，必須)は，仕様で用いられている終端記号とこれらの記号を表す整数値との対応をつくりだす．

```
[例] $TERM              "symbol number
        END-OF-FILE     "0
```

```
real            "1
integer         "2
  ⋮
ARRAY 11        "11
BEGIN           "12
CASE            "13
  ⋮
'.' 44 #        "44 noise word
',' #           "45 noise word
```

3.3 非終端記号定義

非終端記号定義（予約語：$NON-TERM，必須）は言語または語彙の仕様の生成規則名と対応している．言語仕様の生成規則はそれに関連した局所変数（意味解析上またはパース制御上の情報を保持するために用いられる）を持つことができる．

```
[例] $NON-TERM
        〈BLOCK〉 2      "two local variables
        〈CASE-FIELD〉
        〈CONSTANT〉
        〈EXP〉 1        "one local variable
```

3.4 先読み定義

先読み定義部門（予約語：$LOOK）は，後で定義される先読み生成規則名に対応している．先読み生成規則名は言語仕様でのみ使用することができ，複雑な先読みまたは再帰的な先読みを行うために使用される．

```
[例] $LOOK
        〈FIND-STMT〉 1   "one local variable
        〈PARENS〉
```

3.5 集 合 定 義

集合定義部門（予約語：$SET）は，集合を表すために用いられる生成規則に対応している．

```
[例] $SET  〈IDENTIFIER〉        "includes key words
           〈REL-OP〉            "relational operators
           〈KEY-WORD〉          "key word list
           〈PUNCTUATION〉 #     "noise words
```

3.6 生成規則定義

生成規則定義部門（予約語：$PROD，必須）は，パーサまたは語彙解析プログラムに対する言語または語彙の仕様をつくりあげる1個以上の生成規則からなる．

4. 言 語 仕 様

言語仕様は，文・句・式・プログラムおよび言語の他の構成要素を形成する文法を定義する．表構築プロセッサが受理する超言語は，Backus 標準形（BNF）の初期の記法の拡張形であり，先読み，誤り回復，意味呼出し等の概念が直接超言語で表現可能にように拡張されている．

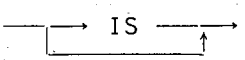
4.1 基本超言語

基本超言語は BNF 記法のすべての能力を持ち，もっと簡明で読みやすい形をしている．これは全体の超言語の中枢である．基本超言語では次の記号を使う．

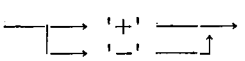
記　　号	名　　前
::=	生成規則記号
.	生成規則終了符号
[ ]	句括弧（左右）
( )	任意句括弧（左右）
\ または |	選択符号
/	並び符号
...	反復記号

GSA の記法の意味を，流れ図との関係で示すと次のようになる．

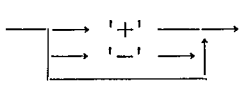
任意：(IS)



選択：['+'\'−']

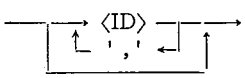


任意選択：('+'\'−')

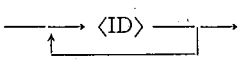


並び：[〈ID〉/',']

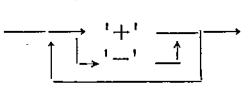
任意並び：(〈ID〉/',')



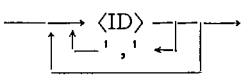
反復：〈ID〉...



選択の反復：['+'\'−']...



並びの反復：[〈ID〉/',']...



4.1.1 生成規則

生成規則は文法を記述する．

```
[例 1] 〈GRAMMER〉 ::= 〈PRODUCTION〉 ... .
       〈PRODUCTION〉 ::= nonterminal '::=' 〈PHRASE〉 ... '.' .
       〈PHRASE〉 ::= [terminal\
                      nonterminal\
                      '[' 〈COMPLEX〉 ']' \
                      '(' 〈COMPLEX〉 ')' ] ( '...') .
       〈COMPLEX〉 ::= 〈PHRASE〉 ... ( [ '\' 〈PHRASE〉 ... ] ... \
[例 2] 〈EXP〉 ::= 〈TERM〉 ( [ '+' \ '−' ] 〈TERM〉 ) ... .
       〈TERM〉 ::= 〈FACTOR〉 ( [ '*' \ '/' ] 〈FACTOR〉 ) ... .
       〈FACTOR〉 ::= 〈OPERAND〉 ( '**' 〈FACTOR〉 ) .
       〈OPERAND〉 ::= [ 〈NUMBFR〉 \
                       〈VARIABLE〉 \
                       '(' 〈EXP〉 ')' ] .
       〈VARIABLE〉 ::= 〈IDENTIFIER〉 ( '(' [ 〈EXP〉 / ',' ] ')' ) .
```

4.1.2 左再帰性の制限

パーサは逆戻りなしの先読み下降型パーサなので，超言語での記述には左再帰性が許されない．そこで，すべての左再帰性は反復によって置き換えなければならない．

```
[例 1] 〈TERM〉 ::= [ 〈FACTOR〉 \
                     〈TERM〉 [ '*' \ '/' ] 〈FACTOR〉 ] .
```

例1は〈TERM〉に関して左再帰的であり，

```
〈TERM〉 ::= 〈FACTOR〉 ( [ '*' \ '/' ] 〈FACTOR〉 ) ••• .
```

によって置き換えることができる．

4.1.3 曖昧さの再構成

逆戻りになしの下降型パージングにおいては，文法の曖昧さは許されていない．

```
[例] 〈BOOL-EXP〉 ::= 〈REL-EXP〉 ( 〈BOOL-OP〉 〈REL-EXP〉 ) ••• .
     〈REL-EXP〉 ::= [ 〈ARITH-EXP〉 〈REL-OP〉 〈ARITH-EXP〉 \
                      '(' 〈BOOL-EXP〉 ')' ] .
     〈ARITH-EXP〉 ::= 〈OPERAND〉 ( 〈ARITH-OP〉 〈OPERAND〉 ) ••• .
     〈OPERAND〉 ::= [ 〈NUMBER〉 \
                      〈VARIABLE〉 \
                      '(' 〈ARITH-EXP〉 ')' ] .
```

この式の構造では，生成規則 〈REL-EXP〉 は '(' に関して曖昧である．この曖昧さを解決するため，すべての括弧でくくられた式を1か所で取り扱うように構文を構成する．式の構造は，以下のように再構成できる．

```
〈EXP〉 ::= 〈EXP1〉 ( 〈BOOL-OP〉 〈EXP1〉 ) ••• .
〈EXP1〉 ::= 〈EXP2〉 ( 〈REL-OP〉 〈EXP2〉 ) ••• .
〈EXP2〉 ::= 〈OPERAND〉 ( 〈ARITH-OP〉 〈OPERAND〉 ) ••• .
〈OPERAND〉 ::= [ 〈NUMBER〉 \
                 〈VARIABLE〉 \
                 '(' 〈EXP〉 ')' ] .
```

4.2 先 読 み

先読みは先読み生成規則，または規則中の先読み記述を使用することにより行う．いずれも，先読みは，入力流れ中の終端記号の先読みを記述し，曖昧さを解決するために用いる．これが主な使用法であるが，再構成によって効率を向上させるため，または誤り回復と一緒に，言語の誤りやすい所または誤りが気になる所を制御するために使用できる．規則中の先読みを記述するために，次の記号を用いる．

記 号	名 前	機 能
?[ ]?	先読み括弧（左右）	先読み節および通常処理節を含む句の範囲を規定
:	モード終了符号	先読み括弧内で用い，先読みの停止，通常処理の再開
?	先読み符号	次の句が先読み句であることを指示
??	先読み検査記号	現在のパースの経路を取るべきかを確かめるため，この点で検査を行うよう要請

4.2.1 曖昧さの解決の例

先読みの使用例として，名札定義または代入文の始めで用いられる 〈IDENTIFIER〉 を終端記号として持つプログラム言語における名前の処理がある．先読みなしでは，これは以下のように記述される．

```
[例1] 〈STATEMENT〉 ::= ( 〈IDENTIFIER〉 ':' )
                        [ 〈IDENTIFIER〉 ':=' 〈EXP〉 \
                        ⋮
```

しかしながら，これは 〈IDENTIFIER〉 に関して曖昧である．先読みを用いると，以下のようになる．

```
[例2] 〈STATEMENT〉 ::= ?[ 〈IDENTIFIER〉 ':' : 〈IDENTIFIER〉 ':' ]?
                        [ 〈IDENTIFIER〉 ':=' 〈EXP〉 \
                        ⋮
```

曖昧さが解決され，〈IDENTIFIER〉 にコロンが続くときのみ，名札の一部分として認識される．すなわち，「〈IDENTIFIER〉 の後に ':' が続けば 〈IDENTIFIER〉 および

':' と認識し，そうでなければ次に行け」となる．

4.2.2 効率のための再構成

再構成を行ってパージングの効率の向上を図るために，先読みを用いることがある．

```
[例] 〈EXP〉 ::= 〈EXP1〉 ( 〈BOOL-OP〉 〈EXP1〉 ) ••• .
     〈EXP1〉 ::= 〈EXP2〉 ( 〈REL-OP〉 〈EXP2〉 ) ••• .
     〈EXP2〉 ::= 〈OPERAND〉 ( 〈ARITH-OP〉 〈OPERAND〉 ) ••• .
     〈OPERAND〉 ::= [ 〈NUMBER〉 \ 〈VARIABLE〉 \ '(' 〈EXP〉 ')' ] .
```

ここで，単一の数および変数を認識するのに多くのレベルの生成規則を通る必要がある．そこで，演算子が次に続くときにのみ余分の生成規則に入るよう，式の取扱いを再構成すると，次のようになる．

```
〈EXP〉 ::= 〈OPERAND〉 ( 〈OPERATION〉 ••• ) .
〈OPERATION〉 ::=
 [ 〈BOOL-OP〉 〈OPERAND〉
                ?[ [ 〈REL-OP〉 \
                     〈ARITH-OP〉 ] : 〈OPERATION〉 ]? ••• \
   〈REL-OP〉 〈OPERAND〉
              ?[ 〈ARITH-OP〉 ] : 〈OPERATION〉 ]? ••• \
   〈ARITH-OP〉 〈OPERAND〉 ] .
```

4.2.3 単一項目の先読み

単一項目の先読みは，単純なパージングの規則の例外事項を持つ言語を取り扱うのに有用である．実際，仕様に単一項目の先読みを追加することにより，大規模な再構成を避けることもできる．たとえば，文の後に ELSE とか END のような手掛り語が続かない限り，文の終りにセミコロンを必要とする言語がある．これらの例外事項は，セミコロンを見つけるために選択項目として，手掛り語に対する先読みを記述することにより処理できる．次の生成規則はこのことがいかに行われるかを示している．この生成規則はセミコロンを参照する代わりに，文の終りに置かれる．

```
[例] 〈;〉 ::= [ ';' \
              ? END \
              ? ELSE ] .
```

4.2.4 先読み検査

先読み検査は誤り回復とともに用いられることが多いが，複雑な句を単純化するのにも用いられる．たとえば，出現の順序は定まっているが出現するかどうかが任意である手掛り語の集まり (A)(B)(C)(D) を処理する場合に，先読み検査を使用できる．これは，次の関係が計算でき，少なくとも1個の手掛り語が存在するかどうかの検査を必要とする文脈に限定される．

```
[??(A)(B)(C)(D)]
```

4.2.5 先読み生成規則

$LOOOK 仕様によって定義される先読み生成規則は，再帰性等の他のファクタにより，生成規則を使用せざるをえないような複雑な先読みを行うために用いる．

```
[例] 〈P-EXP〉 ::= '(' [ 〈NOT-BRACKET〉 \
                       〈P-EXP〉 ] ••• ')' .
```

これは，大括弧 (bracket) を含まない入れ子の括弧でくくられた式の照合を指示する．

4.3 集　　合

集合はグループ内の終端記号の認識を記述するのに用いる．これを記述するのに用いられる記号は次のとおりである．

記 号	名 前	機 能
$[ ]$	集合括弧（左右）	終端記号の集合を形成
$( )$	任意集合括弧（左右）	通常の集合括弧と全く同様に使用されるが，項目が任意であるような種類の集合を作成
••	集合範囲記号	集合括弧内で使用し，終端記号の範囲を指示

[例] 〈REL〉 = 〈EXP〉 $[LT LE EQ GT NE]$ 〈EXP〉 .
〈EXP〉 = $('+' '−')$ 〈TERM〉 ($['+' '−']$ 〈TERM〉) ••• .
〈RESERVED-WORD〉 = $[ACCEPT •• ZEROS]$.

4.4 停止記号

停止記号はパージング中，あるいはパース (parse) 表を構築中に通常の処置を停止するのに用いる．停止を記述するには，以下の記号を使用する．

記 号	名 前
#	スタック停止記号
##	曖昧さ停止記号
###	関係停止記号

4.4.1 スタック停止

パーサは，各トークンのコピーをパーサにあるトークン・スタック上に置くことができるが，スタック停止記号 # はこの処置を停止することを示す．この記号は，次の生成規則のように，不必要な句読点を捨てるのに用いる．

[例] 〈VAR〉 = 〈ID〉 (# '(' [〈EXP〉 / # ','] # ')'] .

4.4.2 曖昧さ停止

曖昧さ停止記号 ## の唯一の効果は，表構築プロセッサが生成する曖昧さのリスティング上にその項目の生成を行わないことである．

[例] 〈LINAGE〉 = LINAGE (IS) 〈INTEGER〉 (LINES)
((## LINES) (AT) TOP 〈INTEGER〉) .

この記号は，LINES に対する曖昧さの印書を停止する．

4.4.3 関係停止

関係停止記号 ### の使用例は，次の生成規則である．

```
[例] 〈STMT〉 = [ GOTO 〈ID〉 \
               BEGIN 〈STMT〉 ••• END \
               ⋮
               ### ELSE 〈* 101 ] .
```

この記号は ELSE を 〈STMT〉 の第1関係から除外するために用いられ，ELSE が現れた場合，文レベルの余分な ELSE は捨られ，メッセージが出力される．

4.5 手続き呼出し

手続き呼出しによって，PLUS, COBOL, PL/I, および，他の言語で書かれた手続きをパージング中に直接呼び出すことができる．ルーチンの呼出しには以下の記号を用いる．

記 号	名 前
%% γ	外部名 γ の手続き呼出し記号
%% γ ι	手続き呼出し記号
% ? γ	外部名 γ の手続き制御記号
% ? γ ι	手続き制御記号

ここで，γ は外部名である．外部名の後に整数 ι が続くと，広域媒介変数にこの値がセットされ，そうでなければゼロがこの変数にセットされる．

4.5.1 標準呼出し

標準呼出しは，パージング中に行われる大部分の意味処理を行う．

以下の生成規則は，単純な式の構造へ手続き呼出しの組み込みを行った場合を示す．

```
〈EXP〉 == 〈TERM〉 ( [ '+' \ '−' ] 〈TERM〉 %% BINARY ) ••• .
〈TERM〉 == 〈FACTOR〉 ( [ '*' \ '/' ] 〈FACTOR〉 %% BINARY ) ••• .
〈FACTOR〉 == 〈OPERAND〉 ( '**' 〈FACTOR〉 %% BINARY ) .
〈OPERAND〉 == [ 〈NUMBER〉 \
                〈VARIABLE〉 \
                '(' 〈EXP〉 ')' %% PAREN ] .
```

〈EXP〉 は式の結果を表す 1 個の項目をトークン・スタック上に残すとし，〈NUMBER〉 と 〈VARIABLE〉 の各々はトークン・スタック上に 1 個の項目を残すとすると，手続き呼出しは以下の処理を行う．

手続き BINARY が呼ばれたとき，トークン・スタックのトップにはオペランド，演算子，2 番目のオペランドの 3 個の項目がある．この手続きは計算を行い，スタックからトップの 2 個の項目を除去し，第 1 のオペランドをその結果で置き換える．

手続き PAREN は，'(' のトークン項目を 〈EXP〉 によってスタック上に残されている項目で置き換え，それがトップの項目となるようにトークン・スタックのポインタをリセットする．

4.5.2 制御呼出し

制御呼出しは，意味手続きにより，パーサが取るべき経路を決めるときに用いる．

```
[例] [ %? INPUT-TEST 〈PROCESS-NORMAL〉 \
       〈PROCESS-SPECIAL〉 ]
```

この例では，手続き INPUT-TEST はパーサに既知の広域変数をセットする．復帰したときにその変数がゼロならば，生成規則 〈PROCESS-NORMAL〉 に入る．そうでないならば，生成規則 〈PROCESS-SPECIAL〉 に入る．

4.6 誤り回復

誤り回復記号は，共通なまたは予想される誤り回復を仕様中に組み入れるとともに，自動的な誤り回復用の誤り文脈を記述するために用いる．誤り回復を記述するのに，以下の記号を使用する．

記　号	名　　前
〈 ι … 〉	誤り文脈符号（左右）
〈 ι … 〉$[…]$	誤り文脈符号（左右）
〈 ι … 〉$(••)$	誤り文脈符号（左右）
〈( )〉	誤り処理括弧（左右）
〈* ι	誤り標識符号
〈〉	誤り符号
••••	反復記号

左誤り文脈符号 〈 は，誤り文脈の始まりを示すのに用いられる．後続の整数 ι は，もしこの文脈に対して回復がなされた場合に，印書されるべき誤りメッセージを示す．

右文脈符号 〉 は，誤り文脈の終りおよび回復点を示す．

右文脈集合符号 〉$[もまた，誤り文脈の終りおよび回復点を示す．

右文脈集合符号 〉$(は，通常の文脈集合符号と異なり，この符号内の項目集合がこ

の文脈の後続集合の項目と論理積を取るのに使用する．

誤り処理括弧 〈( )〉 は，誤り回復文脈の後のみに用いることができる．これは任意の妥当な句，または誤りが起こった後で行うべき処理の集合を含める．もし，この括弧の前の文脈に誤りが検出され，〈(の前の右文脈符号への回復がなされるとき，この括弧内が実行される．

誤り標識符号 〈* は，誤りが起こった際に誤りメッセージ・ルーチンを呼び出す．この標識符号に続く整数 i は誤りメッセージ番号に対応している．

誤り符号 〈〉 は，与えられた点で誤り回復を行うために用いる．

反復記号 •••• は，次の入力記号がこの反復記号の後の文脈で正しくない限り，反復を行うために用いる．

4.6.1 誤り検出

誤りは，必要句が満足されなかったときパーサによって検出される．また，誤りが検出されたとき，回復は検出点を含む右文脈符号に対して行われる．回復文脈符号は，通常誤りが検出された点の右である．しかしながら，回復は静的あるいは動的に誤り点を含む文脈符号に対して行われる．次の生成規則は，簡単な誤り回復の例である．

[例] 〈DEFINE〉 ::= DEFINE 〈 57 〈ID〉 〉 '=' 〈CONSTANT〉 ';' .

もし 〈ID〉 が DEFINE の後に続かなければ，誤りが 〈ID〉 で検出される．回復は，'=' が発見されれば，〈ID〉 に続く右文脈符号に対して行われる．この文脈が選ばれると，誤りメッセージ 57 が発せられ，パージングは '=' の認識へと続く．この場合，回復は誤りが検出された点の右に対して行われる．

さらに，誤り文脈は入れ子にすることができる．

[例] 〈DEFINE〉 ::= DEFINE 〈 58 〈 57 〈ID〉 〉 '='
〈CONSTANT〉 〉 ';' .

ここで，〈ID〉 が現れなかったとき，回復が ';' に対しても行えるようになる．加えて '=' または 〈CONSTANT〉 で誤りが検出されたとき，回復は ';' に対して行われる．

4.6.2 空の誤り文脈

空の誤り文脈は，文脈符号の特別な場合である．これを使用すると，回復が検出の点で行われる．

[例] 〈DEFINE〉 ::= DEFINE 〈ID〉 〈 56 〉 '=' 〈CONSTANT〉 .

〈ID〉 の後に '=' が続かなければ，誤りが '=' で検出され，入力の流れの中に '=' が発見されれば，回復は '=' の前の右文脈符号に対して行われる．空の文脈の効果は次の両生成規則と同じである．

[例] 〈DEFINF〉 ::= DEFINE 〈ID〉 〈 56 ?? 〉 '=' 〈CONSTANT〉 .
〈DEFINE〉 ::= DEFINE 〈ID〉 〈 56 ? '=' 〉 '=' 〈CONSTANT〉 .

4.6.3 誤り処理

誤りが起こった後で，誤り文脈に誤り処理括弧を付加することにより，特別な処理を行うことができる．

[例 1] SET 〈 29 name %% PROCESS-NAME 〉〈(%% FIX-SET)〉

name が認識されずに，誤り回復がこの文脈に関して行われたならば，手続き FIX-SET が呼び出される．この目的は，name がなくとも，処理を続行するのに必要な意味づけを FIX-SET が実行することである．

[例 2] 〈DEFINE〉 ::= DEFINE 〈 57 〈ID〉 〉〈(−> L)〉
'=' 〈CONSTANT〉 ∧ L ';' .

誤り処理括弧内に飛び越し ー〉を置くことにより，右文脈符号を ';' の直前に置くのと同じ効果がある．しかし，回復は '=' でなく ';' に対してのみ行われる．この種の構造は，メッセージを出す文脈を変えずに，回復をもっと広域的なレベルに対して行いたい場合に用いる．

4.6.4 誤り検査

誤り検査を，誤りがしばしば起こると思われる仕様中の場所とか，非常に致命的な誤りが起こる場所に組み込むことができる．

```
[例] 〈 37 ?? 〉( 〈HEADING〉 )
                ( 〈DATA-SECTION〉 )
                〈RECORD-SECTION〉
```

ここで，次の終端記号が 〈HEADING〉，〈DATA-SECTION〉，または 〈RECORD-SECTION〉 のいずれかで始まるかどうか検査される． もし，次の終端記号がこれらのいずれとも合致しなければ，誤り回復に入り，回復はこれらの句の 1 つで始まるものに対して行われる．もしこの検査を加えないと，パーサが 〈RECORD-SECTION〉 を認識しようとするまで誤りは検出されない．

4.6.5 式における回復

```
[例] 〈OPERAND〉 ::= [ 〈NUMBER〉 \
                       〈VARIABLE〉 \
                       '(' 〈EXP〉 〈 49 ')' 〉 %% PAREN \
                       〈 50 〈〉 〉 %% DEFAULT ] .
```

ここで，メッセージ番号 49 の誤り文脈は，右括弧の欠如を検出するために用いられる．右括弧を取り扱う手続きは，誤り回復が行われても行われなくとも，呼び出される． 2 番目の誤り文脈は，オペランド・レベルでの不適当な項目を検出するのに用いられ，代替項目および誤り符号を使用することにより，正しいオペランドから分離されている．もし，不適当なオペランドが検出され回復がこの点で行われると，手続き DEFAULT が呼び出され，正しいオペランドが置かれるべきトークン・スタック上の場所に省略時のオペランドを追加する．

4.6.6 文における回復

文レベルでの誤り回復は複雑で多様である．以下に簡単な言語での誤り回復の使用例を示す．まず，誤り回復なしの構文を示す．

```
[例 1] 〈STMT〉 ::= ?[ 〈ID〉 ':' : 〈ID〉 ':' ]?
                   [ IF 〈EXP〉 THEN 〈STMT〉 ( ELSE 〈STMT〉 ) \
                   [ GOTO 〈ID〉 \
                     〈ID〉 ':=' 〈EXP〉 \
                     BEGIN 〈STMT〉 ••• END ] ';' ] .
```

誤り回復を入れると，次の例のようになる．

```
[例 2] 〈STMT〉 ::= ?[ 〈ID〉 ';' : 〈ID〉 ':' ]?
                   〈 100 [ [ IF 〈EXP〉 THEN 〈STMT〉 ( ELSE 〈STMT〉 〉 \
                   〈 100 [ GOTO 〈ID〉 \
                           〈ID〉 ':=' 〈EXP〉 \
                           BEGIN 〈STMT〉 •••• END ] 〉 ';' ] \
                   ### [ ELSE 〈* 101 \
                   〈 102 〈〉 〉$[ 〈ID〉 ]$ ( ';' ) ] ]
                 〉$( 〈RESERVED-WORD〉 )$ .
```

誤り番号 100 を持つ 2 つの文脈で，文レベルの誤りを捕えることができる．最初の文では，通常 〈STMT〉 の後に続く任意の 〈RESERVED-WORD〉 に対して回復を行う．これ

には IF, GOTO, BEGIN, END, および ELSE が含まれる．2番目の文脈では，回復記号として ';' のみを持つ．そこで，これらの文の誤りは次の場所で検出される．

1) IF 文の〈EXP〉内
2) IF 文の THEN
3) IF 文の両方の〈STMT〉内
4) GOTO 文の〈ID〉
5) 代入文の ':='
6) 代入文の〈EXP〉内
7) BIGIN/END 文の〈STMT〉内
8) 文の終りの ';'
9) 最後の代替項目 〈〉

誤りは，生成規則の次の場所では検出されない．

1) 先読み句内
2) IF 文の IF および ELSE
3) GOTO 文の GOTO
4) 代入文の〈ID〉
5) BIGIN/END 文の BIGIN および END
6) 誤り仕様句の ELSE
7) 最後の代替項目の ';'

文の繰返し …· のためを用いることにより，誤りは必ず文レベルで検出され，誤り検出点として END を除外している．また，関係停止記号 ### により〈STMT〉の第1関係から ELSE を除外している．誤り番号101は誤り仕様句内に含まれ，不一致の ELSE が文レベルで検出されると常に出力される．誤り番号102の文脈もまた誤り仕様内に含まれ，正しい文が見つからなかった場合，誤り回復をこの点で行うようにする誤り符号 〈〉 からなる．誤り番号102に対する回復記号は ';' および〈ID〉を除く通常〈STMT〉の後に続く終端記号である．

4.7 変数操作

パーサは，パージング中に生成された意味処理のための値を，局所変数および広域変数に保持することができる．これらの変数は作成者に対し，パース情報の把握の便宜を図るばかりでなく，意味処理を行うべき点が選択できるという柔軟性をも提供する．

4.8 経路制御

経路制御記号により，すでに入力から引き出された情報に従って，パージングを制御することができる．局所または広域変数に含まれている値を比較し，その結果によって，パースの代替路を選ぶ．

経路制御で用いられる記号は

記号	名前
＝＝ ゥ ィ	等しい
^＝ ゥ ィ	等しくない
〉〉 ゥ ィ	大きい
〈 ＝ ゥ ィ	以上
〉 ＝ ゥ ィ	以下
〉〉 ゥ ィ	小さい

であり，変数 v と整数値 c の比較を行う．その結果が真であれば，パージングはその比較にすぐ続く経路を取る．そうでなければ，パージングは代替路をとる．もし代替路がなければ，誤り回復に入る．

```
[例 1] ⟨STATEMENT⟩ ::= [ == RESTRICTION 1 ⟨RESTRICTED⟩ \
                           ⟨UNRESTRICTED⟩ ] .
```

また，経路制御は構文と意味にまたがる誤りを捕えるのにも用いることができる．

```
[例 2] ⟨PROCEDURE-BLOCK⟩ ::= BEGIN ( == IMPORT 1 ⟨* 100 )
                               ⟨STATEMENT⟩ ... END .
```

4.9 制御記号

制御記号は定義または入力に基づいて制御を移すのに用いられる．これらは種々の理由から用いられる．たとえば，効率の向上，柔軟性の増大，誤り仕様の改善等である．以下の記号が制御を移すために使用される．

記　号	名　　前	機　　能
[#]	選択記号	数個の代替項目および事柄を局所または広域変数に従って選択.
[$]	分岐記号	句括弧の前に用いられ，この点で用いられる次の入力記号に従って仕様中の適当な場所へ分岐.
∧ λ	名札符号	飛越しのための生成規則中の名札の場所を定義．名札名 λ が名札符号の後に続き，その生成規則に局所的な名札として定義.
⟶ λ	飛越し符号	飛越し符号は，その生成規則中に定義されている名札名 λ の場所へ制御を移す.
⟹ c	出口記号	現在の生成規則から出る.

```
[例 1] 選択記号:
       ⟨EXP⟩ ::= [#] LANGUAGE [ 0 ⟨COBOL-EXP⟩ \
                                1 ⟨PL1-EXP⟩ \
                                2 ⟨FORTRAN-EXP⟩ ] .

[例 2] 分岐記号:
       ⟨STMT⟩ ::= ?[ ⟨ID⟩ ':' : ⟨ID⟩ ':' ]?
                  [$] [ IF ⟨EXP⟩ THEN ⟨STMT⟩
                                ( ELSE ⟨STMT⟩ ) \
                        WHILE ⟨EXP⟩ DO ⟨STMT⟩ \
                      [ GOTO ⟨ID⟩ \
                        CALL ⟨ID⟩ ( ⟨PARMS⟩ ) \
                        BEGIN ⟨STMT⟩ ... END \
                          :

[例 3] 名札符号と飛越し符号:
       ⟨BLOCK⟩ ::= ∧ BLK
         ( LABEL [ ⟨UNSIGNED-INTEGER⟩ / ',' ] ';' )
         ( CONST [ ⟨IDENTIFIER⟩ '=' ⟨CONSTANT⟩ ';' ] ... )
           :
         [ BEGIN [ ⟨STATEMENT⟩ / ';' ] END \
         ? $[ ⟨BLOCK⟩ ]$ ⟨* 37 -> BLK \ out of order
           :

[例 4] 出口記号:
       ⟨ITEM⟩ ::= ?[ END REPORT : ⟹ ]?
                  [ END SECTION ⟨ID⟩ \
                  :
```

5. 語彙仕様

語彙仕様は，語彙の基本単位を原始入力から形成する文法を定義する．その一般形式は言語仕様と同様である．

5.1 相 異 点

言語仕様との相異点は以下のとおりである．

1) 語彙の生成規則は局所変数を持つことができない．
2) 先読み生成規則は語彙仕様では定義できない．
3) 語彙解析を行うアセンブリ・プログラムを CODE 選択項目を使用することにより，表の代わりに生成することができる．
4) 語彙解析におけるすべての誤り回復は，代替項目として仕様中にはっきりと組み込まれていなければならない．
5) 変数操作記号 $，＋，－ は広域変数に用いられるときには同じ意味を持つが，そうでない場合には新しい意味を持つ．保存記号＊は語彙解析では未定義であるが，それ以外の残りの操作記号は語彙解析において新しい意味を持つ．語彙での変数操作を記述するのに，以下の記号を使用する．

記 号	名 前
＊＊	開始記号
＊＊ *i*	応答記号
$ *i*	セット記号
＋ *i*	増加記号
－ *i*	減少記号

これらの記号は以下のように使用される．

開始記号 ＊＊ は，この点を開始点として語彙の単位が組み立てられるべきだということを示す．これにより，列の長さおよびハッシュ・コードにゼロがセットされ，行および欄番号が捕えられる．

応答記号 ＊＊ *i* は開始記号と同じものだが，後に整数 *i* が続く．これは語彙解析プログラムからの復帰が行われ，出力の型がその整数でセットされることを示す．語彙解析プログラムの次の呼出し時には，この点から実行が再開される．そのとき，開始記号に通常関連することが行われる．整数を伴ったこの記号は，コルーチン復帰のため使用される．

セット記号 $ の後には整数または終端記号が続く．これにより，その整数または終端記号に対応した文字が出力列に追加される．

増加記号 ＋ の後には整数 *i* が続く．これにより，その整数値が現在の入力文字に加えられ，その結果が出力列に格納される．

減少記号 － の後に整数 *i* が続く．これにより，その整数値が現在の入力文字より減ぜられ，その結果が出力列に格納される．

6) 語彙の項目を記述するのに，以下の制御記号が追加される．

記 号	名 前	機 能
＝〉 *i*	復帰記号	語彙解析プログラムからの復帰が行われ，その出力の型は整数値にセットされる．
！	段移動記号	単一項目または句へ適用することができ，文字が認識されるとき，それに等しい上段の文字が出力列に格納される．

5.2 語彙仕様の例

以下の例は，語彙解析の最も共通なものを示している．標準的な名前の規則は次のとおりである．引用符で囲まれた記号は文字を表し，角括弧内の記号は生成規則である．主生

成規則である〈TOKEN〉を除いて，すべての生成規則名は集合名である．ASCII 選択項目は終端記号(文字集合)を定義するのに用いられる．

この仕様の特徴は，以下のとおりである．

1) 識別子内の逆斜線は下線に変換される．
2) 2文字の特殊記号が認識される．
3) 注釈行は行の欄1の星印によって示される．(変数 N$COLM は欄番号を保持していると仮定されている．)
4) 語彙の単位は，2行以上にわたることができない．
5) 2個の引用符は，1個の引用符を表すために文字定数内で用いられる．
6) 不適当な入力文字は入力から削除され，メッセージが出力される．
7) ファイルの終了に対して，@EOF が返される．

[例]
```
$OPTS LEX ASCII
$PARAM          "equate names to lexical types (numbers)
  END-FILE      0
  ILLEGAL       1
  INTEGER       2
  REAL          3
  IDENTIFIER    4
  LITERAL       5
  SPECIAL       6
$SET
  〈ALPHA〉
  〈CHAR〉
  〈DIGIT〉
  〈NON-QUOTE〉
  〈OTHER〉
  〈SPECIAL〉
$NON-TERM
  〈TOKEN〉
$PROD
  〈TOKEN〉 = ^ TOKEN ( # $[ SP EOL EOC ]$ ) ... **
   [$] [ ! 〈ALPHA〉 ( ! $[ 〈ALPHA〉 〈DIGIT〉 '-' ]$ \
                      # '\' $ '-' ) ... => INDENTIFIER \
        〈DIGIT〉 ... [ '.' ( 〈DIGIT〉 ) ... => REAL \
                                          => INTEGER ] \
        '.' [ 〈DIGIT〉 ... => REAL \
                         => SPECIAL \
        〈SPECIAL〉 => SPECIAL \
        ':' ( '=' ) => SPECIAL \
        '*' [== N$COLM 1 # 〈CHAR〉 ... -> TOKEN \
                           => SPECIAL ] \
        # ''' ^ LIT [ 〈NON-QUOTE〉 -> LIT \
                    # ''' ( ''' -> LIT ) \
                    〈* 50 ] => LITERAL \
        〈OTHER〉 ... => ILLEGAL \
        ERR 〈* 51 \
        BAD... 〈* 52 -> TOKEN \
        ? EOF $ '@' $ 'E' $ 'O' $ 'F' => END-FILE ] .
  〈ALPHA〉      = $[ 'A' .. 'Z' \ 'a' .. 'z' ]$ .
  〈CHAR〉       = $[ NUL .. DEL ]$ .
  〈DIGIT〉      = $[ '0' .. '9' ]$ .
  〈NON-QUOTE〉 = $[ NUL .. '&' \ '(' .. DEL ]$ .
    ⋮
```

6. パ ー サ

パーサは，トークン・ルーチンから入力を受け取る．入力および現在の状態に従って，パーサは認識，制御，意味処理，または誤り回復を実行する．パーサが入力を認識するとき，そのコピーをトークン・スタック上に置くので，入力が意味解析で利用可能である．さらに，パーサはパース情報を保持するための生成規則スタックを持っている．この2つのスタックは，パーサから呼び出される意味ルーチンおよびメッセージ・ルーチンからアクセス可能である．

6.1 入 力

パーサへの入力項目(トークン)の最初の4分の1語だけが，パーサにより使用される．これは，言語仕様中の終端記号に関連するトークン番号を表す．第1の語の残りの部分は，通常トークンに対する行および欄の番号を含んでいる．

トークン番号	欄番号	行 番 号

先読みが行われる場合，その入力はバッファに蓄えられる．このバッファの大きさはパーサ内の定義によって決められ，仕様中の最大の先読みの大きさと同じでなければならない．パーサが入力を必要なときは，トークン・ルーチンを呼び出すことにより，次の入力トークンを得る．手続き呼出しは，入力の走査中にパーサによって行われる．

6.2 トークン・スタック

パーサがトークンを認識する際，それらを捨てることもできるしトークン・スタック上に置くこともできる．トークン・スタックは意味ルーチンおよびメッセージ・ルーチンのためにあり，パーサでは使用されない．スタック上のトークンは，最後のトークンが置かれたスタック上の場所を示すトークン・スタック・ポインタにより参照する．またトークンを参照するのに，この生成規則の最初のトークンが置かれるべきトークン・スタック上の場所を示す生成規則項目の変数を用いることができる．

意味ルーチンもまた，トークン・スタック上にトークンを置くことができる．

6.3 生成規則スタック

生成規則スタックは，起動中の各生成規則に対する生成規則項目を含んでいる．生成規則項目は2語以上を要し，次の形式をしている．

復 帰 点	項目の大きさ
生成規則項目の連結	トークン・スタック・ポインタ
局 所 変 数 1	
⋮	
局 所 変 数 n	

図 2 生成規則項目の形式
Fig. 2 Production entry format

1) 復帰点は，その生成規則を起動させた命令が置かれている命令表の点へのポインタである．
2) 項目の大きさは生成規則項目の大きさを示す．
3) 生成規則項目の連結は，前の生成規則(この生成規則を起動した命令を含む)生成規則の項目を指し示す．

4) トークン・スタック・ポインタはその生成規則の起動後，最初のトークンが置かれるトークン・スタック上の番地に対応している．

5) 生成規則項目中の局所変数の数は，非終端記号または先読み生成規則として生成規則名を定義するときに決められる．

このスタック上の最後の生成規則項目は，生成規則スタック・ポインタにより指し示され，現在パージングが行われている生成規則に対応している．

6.4 誤り回復

誤り回復は，次の条件のいずれかが起こると開始される．

1) 入力記号が現在処理している仕様の部分と合致せず，他に代替項目もない場合
2) 制御手続きが呼び出され，復帰時に広域変数が零でない場合
3) 誤り符号〈〉が パージング中に見つかった場合
4) 経路制御の検査が合致せず，代替項目がない場合
5) 選択においてまた，変数の値が可能な選択番号に合致せず，代替項目がない場合

誤り検出の後は，次の順で回復を行う．

1) この点での回復記号はどの終端記号かを決める．これは，誤りの点が含まれている最小の文脈から最大のもののすべての回復記号の和集合を求めることにより行われる．
2) もし，回復集合中に終端記号がなければ，取扱い不能入力誤りメッセージを出し，呼び出したルーチンへもどる．
3) 最初の未使用記号から走査を始め，回復集合中の記号が見つかるまで走査は続けられる．（入力誤りの回復ならば，最初の未使用記号が誤りを起こした記号である．）
4) この記号が回復記号である最も近い文脈を見つける．
5) その文脈に対して，パーサを正しい生成規則中にあるようにリセットする．
6) 回復が行われた誤り番号を引き数として，メッセージ・ルーチンを呼び出す．
7) 選ばれた回復に対する右文脈符号にすぐ続く命令でパージングを再開する．

毎回，最良の回復を行うという保証はないが，十分考えて誤り文脈を記述すれば，この誤り回復の方法は非常に高い成功率を示すだろう．また，不適当な回復が発見されたとき，誤り回復の仕様をほんの少し変更するだけで修正することができる．したがって，まず一般的で全体的な誤り仕様を作成し，その後，そのシステムを使用した結果を参考にして，実際に行われた誤りの型を包含するように誤り仕様を改善することができる．

7. トークン・ルーチン

トークン・ルーチンはパーサと語彙解析プログラムとのインタフェースである．このルーチンは語彙解析プログラム（または同等の手書きのルーチン）からの出力を受け取り，パーサおよび参照するかもしれない意味，ルーチンにアクセス可能なトークン項目を生成する．また，トークン・ルーチンは以下のことを行う．

1) 手掛り語および予約語を認識する．
2) 認識された項目の辞書を維持する．
3) 数字定数の値を計算する．
4) 語彙の単位の大きさに関する制限事項を検査する．
5) 異なったトークンの単位へ対応づけられる特別な種類の語彙の単位（識別子のような）を変換したり，処理したりする．

トークン・ルーチンからの出力は，パーサの入力バッファに置かれる単一のトークンの項目からなる．各項目の最初の4分の1語は，記号の型を示すトークン番号用に予約されている．項目の残りの部分はアプリケーションに合うように自由に定義することができる．しかしながら，通常，最初の語の残りの部分はそのトークンに関連した欄番号および行番号を保持するのに用いられる．他の語は，辞書ポインタ，意味ポインタ，型情報等を保持する．

トークン番号	欄番号	行番号
意味ポインタ		辞書ポインタ

図 3　典型的なトークン項目
Fig. 3　Typical token entry

8. 語彙解析プログラム

語彙解析プログラムは語彙仕様からつくられた表を用いて，語彙の単位を組み立て，分類して，それをトークン・ルーチンへ渡す．

語彙解析プログラムは入力ルーチンから次の文字を得て，以下の項目を出力する．

1) 認識された文字列
2) 文字列の長さ
3) 最初の文字が現れた行および欄
4) 文字列の語彙の型を示す整数
5) 文字列のハッシュ語

9. 入力ルーチン

このルーチンは語彙解析プログラムとオペレーティング・システムとのインタフェースであり，オペレーティング・システムから，原始リスティングを印書できる行イメージとして，受け取る．入力イメージはこのルーチンによって分解され，数個の特殊文字を付加して，文字単位で語彙解析プログラムへ渡される．

行イメージはオペレーティング・システムから直接，または修正カードの機能あるいはその他の原始入力ルーチンを通して得ることができる．いずれの場合とも，行イメージは読込みと印書の双方に使用されるバッファ中に置かれる．

出力文字は，128個のすべての ASCII 文字および次の特殊文字からなる拡大 ASCII 文字集合内のものである．

8進数コード	機　能	記　号
200	行の終了	EOL
201	ファイルの終了	EOF
202	コピーの終了	EOC
303	入力の誤り	ERR
204	不適当な文字	BAD

9.1 リスキャン入力

入力が使用者定義バッファから読み込まれるように，入力の方法を一時的に変えることができる．この機能（リスキャン：rescan）により，シンボリックの列を容易に置き換えることができる．

リスキャンの要求後，入力ルーチンから返される次の文字は使用者定義バッファ内の最初の文字である．連続した入力ルーチンの呼出しごとに，そのバッファの次の文字が，最後の文字まで，次々と返される．最後の文字が返された後に行終了文字が返され，その後入力ルーチンはリスキャンが開始される以前に処理していた行にもどる．

9.2 コピー入力

コピー入力により，ファイル・エレメントからの入力を原始入力に追加できる．コピーはリスキャンによく似ている．コピー入力が終了するまで処理中の入力が保存され，次の入力行は指定されたエレメントから取られる．そのエレメントが空になったら，コピー終了文字が返され，入力は走査が中断された点から再開される．

9.3 会話型読込み

標準原始入力はオペレーティング・システムから1行の入力を得るために，中断されることがある．この機能はコピーおよびリスキャンによく似ているが，標準システム読込みから，1行分の入力イメージを得るため入力を行う．

10. 意味ルーチン

意味ルーチンは，パーサまたは語彙解析プログラムから呼び出される使用者作成ルーチンである．これらのルーチンは認識された入力に関し，必要とされるすべての意味処理を行う．これらのルーチンは通常，トークン・スタック，辞書表，語彙の変数，およびその処理を行うのに必要な情報等を参照する．

意味ルーチンに期待されている出力(たとえば，テキスト項目，文字列，逆ポーランド，表項目，等)はすべて意味ルーチンによって行われなければならない．アプリケーションごとに変化する出力を自動的に生成する方法はないが，ほとんどの出力は同様な方法で生成できる．またトークン・スタック上の情報は，内部表とともに出力を形成するのに使用できる．

11. メッセージ・ルーチン

メッセージ・ルーチンは誤りメッセージ，警告，見出し等を出力するため，集中化されたルーチンである．この集中化により，メッセージ表の圧縮および情報をアクセスし，メッセージを生成するのに必要なコードの縮少を可能にしている．このルーチンは，情報が追加される点を除いて，最終のメッセージに非常によく似たシンボリックの入力から，特定のプロセッサによって生成される表を使用する．メッセージ表の解読は，辞書のような言語に依存する情報の挿入を除いて，言語に独立である．

メッセージを組み立てるとき，アプリケーションに依存する処置が必要であろう．これらは辞書項目，トークン項目，意味表，およびメッセージを組み立てるのに必要な他の情報の参照を含むが，これらすべての処置は項目の構造に依存し，メッセージ・ルーチン内にコード化されていなければならない．なお，処置語または挿入語がメッセージ表中で見つけられた場合，これに対応する処理を行う．

12. メッセージ表構築プロセッサ

メッセージ表構築プロセッサは，最終のメッセージに非常によく似たシンボリックの入力からリロケータブルの ASCII メッセージ表を生成する．このプログラムの使用により，シンボリックの入力を容易に変更して，新しい表を生成することができる．したがっ

て，理解しやすいメッセージの作成および保守が容易となる．

加えて，このプロセッサはタイプ誤り，つづりの誤り等を見つけるのに用いられる相互参照表(アルファベット順)を生成する．

メッセージ表構築プロセッサへの入力は，処理のために個々の語に分解される行イメージの形式をしている．語とは1個以上の非空白文字列であるか，空白あるいは非空白文字を1個以上含み，1個の二重引用符で始まり，1個の二重引用符(その後二重引用符が続かない)で終わる文字列である．引用符項目内の連続した2個の二重引用符は1個の二重引用符に変換する．各々の語は1個以上の空白文字，または行の終了によって他の語と区切る．

12.1 特 殊 語

左大括弧 [または左角括弧 〈 で始まり，右大括弧] または右角括弧 〉 で終わる語はすべて特殊語と考え，メッセージ表に特殊項目を組み入れるために用いる．特殊語は，種種の長さのテキストがメッセージ・ルーチンによって挿入されるべきメッセージ中の場所を示すのに用いる．

挿入語はメッセージ・ルーチンにテキストの挿入場所を示す．挿入語はメッセージ中に何を挿入すべきかは示さない．これはメッセージ・ルーチンによって決められ，使用される方法もルーチンによって様々である．メッセージ中にテキストの挿入の4つの型を示すのに用いられる8個の予約語がある．

各々の型は2種類の表現法があり，空白を項目の前後に挿入すべきかどうかを示す．

記　　号	挿 入 の 型
[?**?**] または 〈?**?**〉	前後ともに空白を挿入する．
[?**!**] または 〈?**!**〉	前のみに空白を挿入する．
[!**?**] または 〈!**?**〉	後のみに空白を挿入する．
[!**!**] または 〈!**!**〉	前後ともに空白を挿入しない．

処置語は挿入語の拡張で，何を挿入するかをメッセージ・ルーチンに知らせる．これは処置語中に現れる処置番号とメッセージ・ルーチン内の処置表またはスイッチの対応により行われる．この対応は通常アプリケーションごとに変わるものもあり，実際の処置はメッセージ・ルーチン次第である．処置は必ずしもメッセージ中にテキストを挿入する必要はなく，標識をセットしたり，ルーチンを呼び出したり，他の処置を行ったりできる．これらは，一組の大括弧または角括弧内に，コンマで区切られた1個以上の整数の並びとして表現される．

12.2 メッセージ行

各メッセージ行は整数で始めなければならないし，1個以上のメッセージ語がその後に続く．

```
[例]   1  *ERROR* KEYWORD [??] AT [3, 2] IS OUT OF ORDER
     101  *WARNING* VARIABLE [??] IS NOT USED
     205  *MESSAGE* "END TEXT" AT [21, 013, 9] IS REDUNDANT
```

13. 手掛り語表構築プロセッサ

手掛り語表構築プロセッサは，終端記号の並びに似たシンボリックの入力からリロケータブルの手掛り語表を生成するプロセッサである．これらの表は，手掛り語，特殊文字，選択語，および他の項目に対してパス番号と意味番号を等しくするのに用いられる．

14. おわりに

GSA は，コンピュータ言語の走査を必要とする言語プロセッサ，テキスト，プロセッサ，および他のアプリケーション走査プロセッサを構築する際に用いられる．GSA ではソフトウェア生産の道具として使用するため，誤り回復の問題と同様に場所と時間の効率の問題も考慮されている．

（システム・ソフトウェア開発部　上尾　正明 訳）

執筆者紹介　T. N. ターバ (Thomas N. Turba)

Wisconsin 大学においてコンピュータ・サイエンスで MS を取得．1972年 Sperry Univac 社に入社．同社 Roseville 開発センターにおいて言語システム・グループのスーパーバイザ・プログラマで，GSA の主設計・作成者，かつ MACRO システムの最初の設計・作成者．加えて，PLUS コンパイラ等，多くの開発に尽力してきた．

訳 者 注　現在，GSA のメッセージ構築機能(11章, 12章)は大幅に改良されていて，だれでもが簡単に使用できるようになった．すなわち，本稿でいうところの特殊語が，もっと一般的にプログラミングできる形(各種の指令文)に改良されている．したがって，メッセージ・ルーチンは，使用者が作成すべきルーチンであったが，現在では標準ルーチンとなっている．

論文 OA のためのワークフロー・コントロール・モデル

Automated Workflow Control: A Key to Office Productivity

L. S. Baumann, R. D. Coop

要 約 最近の一連の調査によって，オフィスの生産性向上の隘路が作業受渡し (work exchange) の問題にあることが明らかになった．この背景として，下級管理者や専門職に対する支援スタッフの削減によって，タイプや事務に関する作業受渡しが滞り，それらの作業を彼等自身が行わざるを得なくなってきたこと，などがあげられている．そして，これらの問題の解決を契機として，作業受渡しとその管理を機械化するという考えが生まれた．

作業受渡しを一般化すると，ワークフロー(作業流れ)の問題としてとらえられる．ここで，ワークフローは，オフィス・プロセス間における特定の作業の移動のことをいう．ワークフローとしては，1つの作業域 (worker dominion) 内における作業の移動と複数の作業域間での作業の移動が考えられる．作業受渡しは後者を示す言葉である．作業受渡しにおける問題の多くは，複数作業域間における作業の移動の管理（あるいは統制）に関連しているため，機械化によってワークフロー・コントロールの問題が解決すれば，作業受渡しの問題も解決し，オフィスの生産性が向上することになる．

本稿は，1つのワークステーション内および複数のワークステーション間での作業の管理(ワークフロー・コントロール)が，分離可能かつ機械化可能なプロセスであることを述べる．このワークフロー・コントロール・プロセスによって，オフィスの諸組織と諸規定に合致した作業の定義や割当て，作業状況の報告，ワークステーション間での作業の移動などが実現する．

Abstract A number of recent studies have indicated that the lack of growth in office productivity in the last few years has been due to difficulties encountered in work interchange among workers. For example, reduced support to managers and professionals has created work interchange queues for typing and clerical services. Consequently, these services are often performed by the managers and professionals themselves. These difficulties suggest that electronic mechanization should be directed at the work interchange problem and corresponding management issues.

Work interchange is one aspect of a more general office activity called workflow. Workflow is the movement of particular segments of work through identifiable office processes. Thus workflow is involved with work movement under the administration of a worker or between worker dominions. Work interchange is the latter aspect of workflow. Most problems in work interchange are management (or control) problems associated with work movement between worker dominions. Thus, a solution to the workflow control problem solves the work interchange problems and leads to improved office productivity.

This paper shows that the coordination of work (workflow control) at a workstation and between workstations is an isolatable and mechanizable process. This workflow control process assigns and states work, reports on the status of work elements and transports work between workstations in accordance with office procedures and organizations.

1. はじめに

19世紀半ばまでの欧米における主な労働力は人間であり，生産性の向上はワーク・フロ

This article is reprinted, with permission, from AFIPS Conference Proceedings (1980 National Computer Conference).

ーの改良と賃金の引き下げによって達成されていた．しかし，産業革命の勃興とともに生産性にかかわる第3の要素として機械化（後にコンピュータの導入とともに自動化と呼ばれるようになった）が登場し，これによって労働者1人当たりの生産量が飛躍的に増大した．そして，構造化しやすくかつ手がけやすい作業，具体的には労働集約性の強い，農場や工場の労働を対象として機械化が進められ，その結果生産量が急激に増大した．

しかし，この革命からただ1つ取り残された労働集団に，いわゆるホワイト・カラー労働者があった．彼等の仕事の多くは構造化しにくいため，事務職および秘書（あるいはタイピスト）を対象とする業務の機械化に留まっていた．それでも今世紀の半ばまでは，ホワイト・カラーの生産性は，工場および農場労働者の生産性に追従してきた．しかしその頃になると，工場における自動化の発展によって両者間の差が顕著になり始めた．このことは，1965年～1975年の米国電子工業における工場労働者の生産性の伸びが年率3.6パーセントであるのに対して，一方のホワイト・カラーは2.9パーセントであることにも現れている[1]．このため，当然のことながら，全労働者の生産性は，ホワイト・カラーの生産性の低さによって制約されることになった．

最近の調査[2,5,6,7,8,11]によれば，オフィスの生産性は下級管理職（principal，管理者および専門職）の働きに依存することが明らかにされている．そして生産性停滞の原因は，過去数年にわたる彼等に対する支援スタッフの削減と，それによる仕事量の低下にあると考られている．一方，原因をさらに究明すると，オフィスにおける非生産的コストの最大項目は作業受渡しの損失時間であることがわかる．そこでオフィスの生産性向上にとって，作業受渡し，作業の管理統制，管理層に対する即応的支援の諸研究はその効果大であるといえよう．

本稿では，まずオフィス作業の性格を分析する．次に一般的なオフィス・プロセスの数学的モデルを与え，ワークフロー・コントロールが分離可能，かつ機械化可能なプロセスであることを明確に示し，さらにワークフロー・コントロール・プロセスの基本要素について略述する．これらの基本要素の機械化が，オフィスの生産性を向上させることになろう．

2. オフィス作業の性格

オフィスを構成するものは，企業の使命を遂行するために1つの環境内において相互に作用を及ぼし合う人々であるといえよう[2,10]．そして，人々の相互作用を支援する資源を提供するのが環境である。また，オフィスでの作業者と環境との**インタフェース**は重要であり，オフィスにおける作業プロセスの段階づけ，および各段階における作業は，作業者と環境との，あるいは環境を介しての他の作業者とのインタフェースによって完全に規定される．

さて，オフィスでの**プロセス**の概念は，見かけほど明確ではないが，データ（情報）あるいは物（ドキュメンテーション等）が流出入する継続的活動として定義される．そしてプロセスの開始から完了に至る各段階で扱われるデータあるいは物の状態は，**スナップ・ショット・インタイム** (snapshot in time, スナップ・ショットの時系列）によって見ることができる．

また，オフィス組織は，オフィス作業者を統制する管理者の階層としてとらえられ，そのどの断面をとっても管理者および，作業者が存在する．管理者は，作業者に仕事を割り当ててオフィス任務を遂行させ，一方，作業者は割り当てられた仕事の進捗状況を管理者

に報告し，計画に対する評価や意思決定のためのフィードバック情報を提供する．なお，以下では，作業割当てのような高水準プロセスを**オフィス・プロセス** (office process) と呼ぶ．オフィス・プロセスの定義は，各オフィスや，組織階層上の各部位ごとに異なる．

オフィス・プロセスは複数の並行サブプロセスからなる継続的活動であり，スナップ・ショットをとると，常時，複数の並行単一作業者プロセス (concurrent single worker process) が，複数の作業者によって並行して遂行されている．そして，スナップ・ショットのタイミングを適宜時間軸に沿って移動させること（スナップ・ショットの時系列）によって，これらの並行単一作業者プロセスを順序づけることができる．このスナップ・ショット・インタイムの考え方が，本稿のモデル構築の基礎となっている．さて，図1によって，具体的にオフィス・プロセスのモデリングを説明しよう．

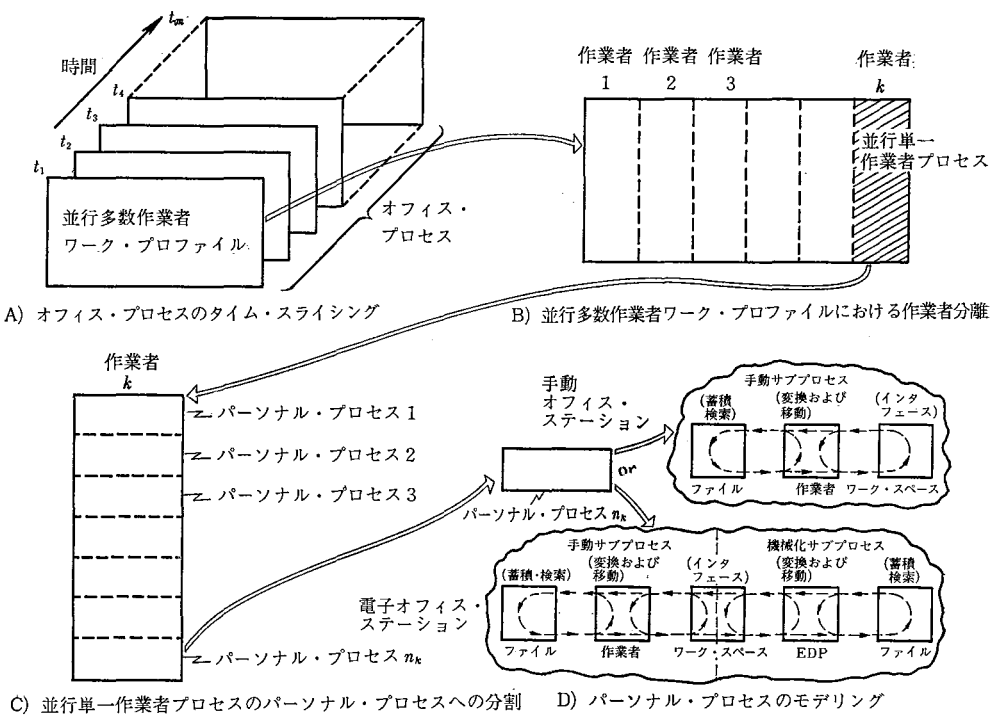


図 1　オフィス・プロセスの分割とモデリング

Fig. 1　Segmentation and modeling of the office process

まず，オフィス・プロセスは，図1のA部で示されるように m 回のスナップ・ショットによって，m 個のタイム・スライスに分けられる．これを**並行多数作業者ワーク・プロファイル** (concurrent multi-worker work profile) あるいは，単にワーク・プロファイルと呼ぶ．そして，この並行多数作業者ワーク・プロファイルは，図1のB部のように個々の作業者に対応する k 個の並行単一作業者プロセスに分割され，さらに，この並行単一作業者プロセスは，図1のC部のように n_k 個のユニークな**パーソナル・プロセス** (personal process) に分けられる．そして，個々のパーソナル・プロセスは，図1のD部のように手動オフィス・ステーション (manual office station) あるいは電子オフィス・ステーション (electronic office station) において実行される．そして，後者の場合，個々のパーソナル・プロセスは，さらに**手動サブプロセス** (manual subprocess) および**機械化サブプロセス**

(mechanized subprocess) に分けられる．電子オフィス・ステーションでは，すべてのパーソナル・プロセスは，手動および機械的手段の両方によって実行可能でなければならない．電子装置の故障の不可避性，および処理済み情報の損失の不経済性を考えると，障害発生時における，機械化サブプロセスから手動サブプロセスへの切替えによる処理の続行は不可欠である．そこで，両方のパーソナル・プロセスはよく似ているだけでなく，任意の時点での相互切替えが可能でなければならない．また，この切替えで必要となる回復操作は非生産的なため，最小限であることが求められる．

パーソナル・プロセスは，**サブプロセス変換** (subprocess transformation)，および **情報とそれが置かれる場所の状態** (information/topological state) で構成される．また，サブプロセス変換は，情報の変換 (transformation) とその位置の移動 (transport) に分けられる．そして，手動オフィス・ステーションにおけるパーソナル・プロセスは，**手動ワーク・スペース** (manual workspace)，**手動ファイル** (manual file)，**作業者** (worker) といった場所 (topology) とそこに含まれる情報，およびサブプロセス変換によって構成される．なお，この場合のサブプロセス変換は，作業者によって遂行される。一方，電子オフィス・ステーションにおけるパーソナル・プロセスは，手動サブプロセスと機械化サブプロセスに2分され，手動サブプロセスは前述の手動オフィス・ステーションの場合と同様である。また，機械化サブプロセスは，**機械化ワーク・スペース** (mechanized workspace)，**機械化ファイル** (mechanized file)，**EDP** (electronic data processing) システムといった場所とそこに含まれる情報，およびサブプロセス変換によって構成される。なお，この場合のサブプロセス変換は EDP システムで遂行される．

パーソナル・プロセスにおける情報の変換には，**構造的変換** (structured transformation) と**非構造的変換** (unstructured transformation) の2つの種類がある[12,13]．構造的変換は EDP システムだけで計算できるプロセス変換であり，非構造的変換はその計算に当たって作業者の助けを必要とするプロセス変換である．また，ワーク・スペースは構造的変換と非構造的変換とのインタフェースとして働き，両方の変換が合成される場所 (topological location) となっている．

3. オフィス作業のモデリング

次の目標は，ワークフロー・コントロールの概念を定義するオフィス作業のモデルを構成することである．以下では，前出の図1に基づきオフィス・プロセスの数学的モデルを作成する．本稿のモデルは M. D. Zisman[4]によるもので，「オフィス・プロセスは，知識領域 (knowledge domain) とプロセス(あるいはアクション)領域 (process domain) の2つの状態を持ち，プロセス領域の活動は知識領域によって制御される」という考えに立脚している．これによると，プロセスのモデルは計算可能な順序グラフ (ordered graph) $P[Q \times C]$ として表現される．ここで $P, Q, C, Q \times C$ は，それぞれペトリ・ネット[14]，知識領域状態集合，プロセス領域状態集合，Q と C の直積である．

さて，図1のD部によると，手動オフィスを利用する作業者が k 人いる場合の，作業者 i の**手動オフィス・パーソナル・プロセス**(手動サブプロセス)は，

$$P_i[Q \times C] \qquad (i=1, 2, \cdots, k)$$

によって定式化される．

一方，電子オフィスを利用する作業者が k' 人いる場合の，作業者 j の**電子オフィス・パーソナル・プロセス**は，**手動サブプロセス**である，

$$P_j'[Q\times C] \qquad (j=1, 2, \cdots, k')$$

と，EDP システムを意味する**機械化サブプロセス**，

$$P_j''[Q\times C] \qquad (j=1, 2, \cdots, k')$$

によって構成される．

図1のC部で見られるように，1つのタイム・スライスにおける1人の作業者に対して複数の並行パーソナル・プロセスが可能となる．この並行性は，手動オフィスにおける作業者 i に対する n_i 個の手動オフィス・パーソナル・プロセスと，電子オフィスにおける作業者 j に対する n_j' 個の電子オフィス・パーソナル・プロセスの結合 (union) によって表現されるため，1つのタイム・スライスにおける1人の作業者の全パーソナル・プロセスは，

$$U_{\alpha=1}^{n_i}\{P_{i_\alpha}\} \cup U_{\beta=1}^{n_j'}\begin{Bmatrix} P_{j_\beta}' \\ P_{j_\beta}'' \end{Bmatrix}$$

となる．また，1つのタイム・スライスにおける全作業者の作業モデル（ワーク・プロファイル）は，図1のB部によって，すべての作業者のパーソナル・プロセスの結合として表現され，次式が得られる．

$$U_{i=1}^{k}U_{\alpha=1}^{n_i}\{P_{i_\alpha}\} \cup U_{j=1}^{k'}U_{\beta=1}^{n_j'}\begin{Bmatrix} P_{j_\beta}' \\ P_{j_\beta}'' \end{Bmatrix}$$

さて，オフィス・プロセスは，順序づけられた m 個のワーク・プロファイルとして表現されるため，いま，ここでワーク・プロファイルを選択し順序づける操作 (operation) S を導入すると，オフィス・プロセスは，

$$S_{\tau=1}^{m}\left[U_{i=1}^{k}U_{\alpha=1}^{n_{i\tau}}\{P_{i_\alpha}\} \cup U_{j=1}^{k'}U_{\beta=1}^{n_{j\tau}}\begin{Bmatrix} P_{j_\beta}' \\ P_{j_\beta}'' \end{Bmatrix}\right]$$

として定式化される．また，ペトリ・ネットの特性を利用するとともに，パーソナル・プロセスの並行性および順序性が，知識領域上の知識状態モデル (knowledge state model) で表現されること (M. D. Zisman[4]) を用いると，オフィス・プロセスの形式的モデルは次に示す一連のグラフとなる．

いま，順序性および並行性を表現する知識状態モデルを P_c とすると，オフィス・プロセスは，

$$\begin{Bmatrix} P_c[Q] \\ P_{i_r}[Q\times C] \\ P_{j_s}'[Q\times C] \\ P_{j_t}''[Q\times C] \end{Bmatrix} \qquad \text{ただし，} \begin{cases} i=1, 2, \cdots, k \\ j=1, 2, \cdots, k' \\ r=1, 2, \cdots, (n_1+n_2+\cdots+n_m) \\ s=1, 2, \cdots, (n_1'+n_2'+\cdots+n_m') \\ t=1, 2, \cdots, (n_1'+n_2'+\cdots+n_m') \end{cases}$$

として表される．また，この P_c がオフィス管理者知識モデル (office manager knowledge model) のグラフ P_c' と，ワークフロー・コントロールのグラフ P_c'' に分解されることを用いると，最終的なオフィス・プロセス・モデルとして，

$$\begin{Bmatrix} P_c' \\ P_c'' \\ P_{i_r} \\ P_{j_s}' \\ P_{j_t}'' \end{Bmatrix} \qquad \text{ただし，} i, j, r, s, t \text{ の条件は上式と同じである．}$$

が得られる．また，手動作業者 (manual worker) と機械化作業者 (mechanized worker) が，それぞれすべての r とすべての s に関する1つの連結グラフ (consolidated graph) として P_i および P_j' によって表現され，かつ EDP システムを意味する機械化サブプロセ

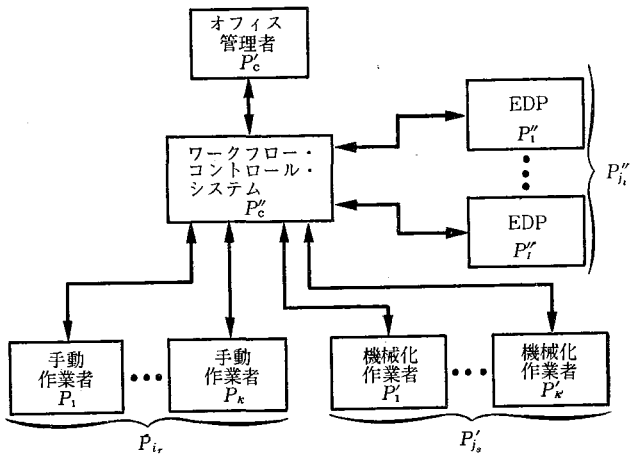


図 2　オフィス・プロセス・モデルの物理的表現
Fig. 2　Physical representation of office process model

ス P_{j_t}'' がすべての t に関する I 個のユニーク連結グラフ P_σ'' $(\sigma=1, 2, \cdots, I)$ によって構成されると考えると，最終的に得られるオフィス・プロセス・モデルは図2で示される．このオフィス・プロセス・モデルは，1人のオフィス管理者 P_c'，ワークフロー・コントロール・システム P_c''，k 人の手動作業者 P_i $(i=1, 2, \cdots, k)$，k' 人の機械化作業者 P_j' $(j=1, 2, \cdots, k')$，I 個の EDP システム P_σ'' $(\sigma=1, 2, \cdots, I)$ によって構成される．

4.　ワークフロー・コントロールの基本要素

3章でその存在と分離可能性について述べたワークフロー・コントロールについて，ここではその基本的な概念および基本要素を吟味し，これによってワークフロー・コントロールの機械化可能性を示す．オフィスは，企業の管制塔 (control tower) であり，情報の供給所 (service station) である．その一般的な使命は情報の収集・分析によって，企業における物流・生産・購買・財務の諸活動を統制すること，および文書(他の形式も含む)通信・会計・複写・ファイリングなどの諸サービスを提供することにある．さて，オフィスの特性は，**特定の作業に依存しないもの** (オフィス特性) と，**特定の作業に依存するもの** (作業特性) に大別される．そして，ワークフロー・コントールの基本要素は，これに準じて2分され，オフィス特性を与えるオフィス特性規定要素と，作業特性を与える作業特性規定要素に大別される．以下でオフィス特性規定要素について説明する．まず，**オフィスの使命** (office mission) によって，企業の統制とサービスに関するオフィスの職務 (responsibility) が定義されると，オフィス管理者 (office management) は，オフィスの基本要素としての人間と設備を組織化しその使命を達成するべく，**人間** (personnel)・**方法** (methods)・**設備** (facilities) に関する調整を行う．

また，**オフィス組織** (office organization) の形成によって，ジョブの関係から見たオフィス作業者の相互関係が定義される．一般に，オフィス組織は，トップに管理者を，ボトムにオフィス作業者(専門職と支援作業者)を有する垂直的・階層的なものであり，このため情報の水平的流れをどうするかという問題が発生する．そこで一連の**オフィス規定** (office procedure) によって，情報の流れの方法を定義する．なお，オフィス規定によって定義されるものは，特定の作業に依存しない一般的な情報の移動方法である．さて，オフィスの使命，組織および規定が備わったとしても，これだけでは不十分であり，**人間に対**

する設備の割当て (people/facility assignment) が必要である．作業者に設備を割り当てる場合，作業者と設備との対応は一対一とは限らず，複数の作業者による１つの設備の利用や，１人の作業者による複数の設備の利用も考えられる．また，設備の割当てに関連するものにオフィス・レイアウトがある．これは，空間を複数の部門や個人に割り当てることである．このほか**対オフィス外インタフェースと通信規定** (exo-office interface and procedure) の作成は，オフィスが企業の統制とサービスの中枢として機能するために不可欠である．オフィス外部とのコミュニケーションには，オフィスと工場間のように企業内に留まるものもあれば企業間の通信もあるが，とくに機密性の保持に関する配慮が重要であり，組織内における各階層での承認手続きの制定が必要である．

さて，これまでの話で作業が行われるフレームワークは用意されたが，特定の作業を実行するための準備や，その実行の統制法が設定されたわけではない．そこで，以下では，これらを行う作業特性規定要素について説明する．オフィスのフレームワークは，前述のオフィス使命，オフィス組織，オフィス規定，人間に対する設備の割当て，対オフィス外インタフェースと通信規定によって記述される．そして，このフレームワーク中を流れる特定作業の移動のことを**ワークフロー** (workflow) と呼ぶ．また，ワークフローのコントロールを完全に行うためには，フレームワークの設定だけでなく，オフィス使命と整合性のとれた作業アクティビティの定義が必要であり，これを**作業定義** (work definition) と呼ぶ．なお，作業定義は管理者の任務である．そして，いったんこれが決定されると，その**作業割当て** (work assignment) が関係する作業責任者に与えられる．なお，作業割当てを決定するためには，その作業アクティビティを実行する作業責任者と作業支援者からなる特定の**作業組織** (work organization) に関する知識が必要とされる．さて，作業割当てによって開始された作業アクティビティは作業責任者によって分解され，前もって記述された一連のオフィス・プロセスに還元される．そして，各オフィス・プロセスは，複数の離散的な段階によって構成され，作業は時間の経過とともに１つのプロセス段階から他のプロセス段階へと段階的に進む．これを**作業ステージング** (work staging) と呼ぶ．そして，作業ステージングと各段階における並行作業アクティビティの時間的枠組を，**作業スケジューリング** (work scheduling) と呼ぶ．また，作業は常に計画どおりに進むとは限らないため，管理者の期待に沿うように補正を行わなければならない．この機能を**作業実施** (work enforcement) という．また，作業のタイムリーな統制と管理には，作業アクティビティの進捗状況の完全な知識が要求され，このため**作業測定** (work measurement) が必要となり，作業基準との比較や作業の進捗状況を図る尺度が設定される．

5. ワークフロー・コントロール・モデルの構成

ワークフロー・コントロール・モデル P_c'' は，知識領域 Q 上の拡張ペトリ・ネット (APN, augmented petrinet) として表現され[4,12]，知識状態(プレース)，状態推移(トランジション)，マーキング・トークン，トランジションのファイヤリング条件となるプロダクション・ルールで構成される．なお，P_c'' のプロダクション・ルールとしては，プロセス領域の状態 C が用いられる．一方，人手作業者 P_i，機械化作業者 P_j'，EDP システム P_σ'' によって，プロセス領域 $Q\times C$ 上で操作される作業情報データは，$\langle q, D, \phi\rangle$ であり，q, D, ϕ は，それぞれ

q：データの知識領域の状態のことで，ドキュメンテーションに関し“何が”および“いつ”という質問に答えるもの(作業指令に相当)．

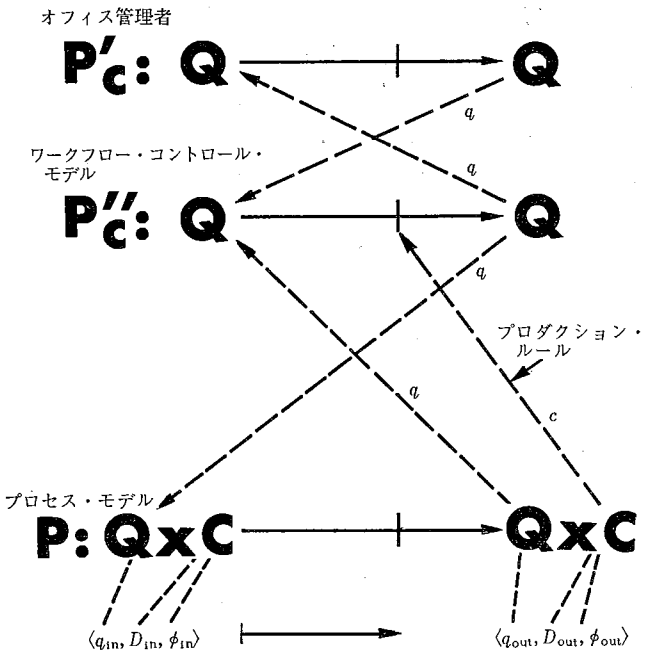


図 3　モデル間の相互関係
Fig. 3　Model interconnections

D：データのドキュメンテーションの状態（レコード，ページ，ドキュメンテーション，フォルダー等）のこと．

ϕ：データの所在場所や所有者の状態（ファイル，ワーク・ステーション，作業者の存在する場所等）のことで，ドキュメンテーションに関し，“だれが”および“どこに”という質問に答えるもの．

を意味するものとする．さて，次に図3に基づき，ワークフロー・コントロール・モデルとプロセス・モデルとの関係を説明する．まずワークフロー・コントロール・モデル P_c'' における出力知識状態 q_{out} によって，プロセス・モデル P_i, P_j', P_c'' のプロセス APN におけるプロダクション・ルールが与えられ，入力プロセス状態 (D_{in}, ϕ_{in}) は出力プロセス状態 (D_{out}, ϕ_{out}) に遷移する．そして，この (D_{out}, ϕ_{out}) をプロダクション・ルールとする知識 APN 上の状態遷移によってプロセス・モデルの入力知識状態 q_{in} は出力知識状態 q_{out} になる．さらに，このプロセス・モデルの q_{out} と (D_{out}, ϕ_{out}) は，ワークフロー・コントロール・モデルにおける入力知識状態 q_{in} とそのプロダクション・ルールとして用いられる．

さて，次にワークフロー・コントロール・モデル P_c'' を，4章で述べた以下の 12 の基本要素によって構築する．

P_1：オフィス使命　　P_2：オフィス組織
P_3：オフィス規定　　P_4：人間に対する設備の割当て
P_5：対オフィス外インタフェースと通信規定　　P_6：作業定義
P_7：作業組織　　P_8：作業割当て
P_9：作業ステージング　　P_{10}：作業スケジューリング
P_{11}：作業実施　　P_{12}：作業測定

まず，P_c'' を構成するためには，これらの基本要素だけでは不十分であり，これらを関

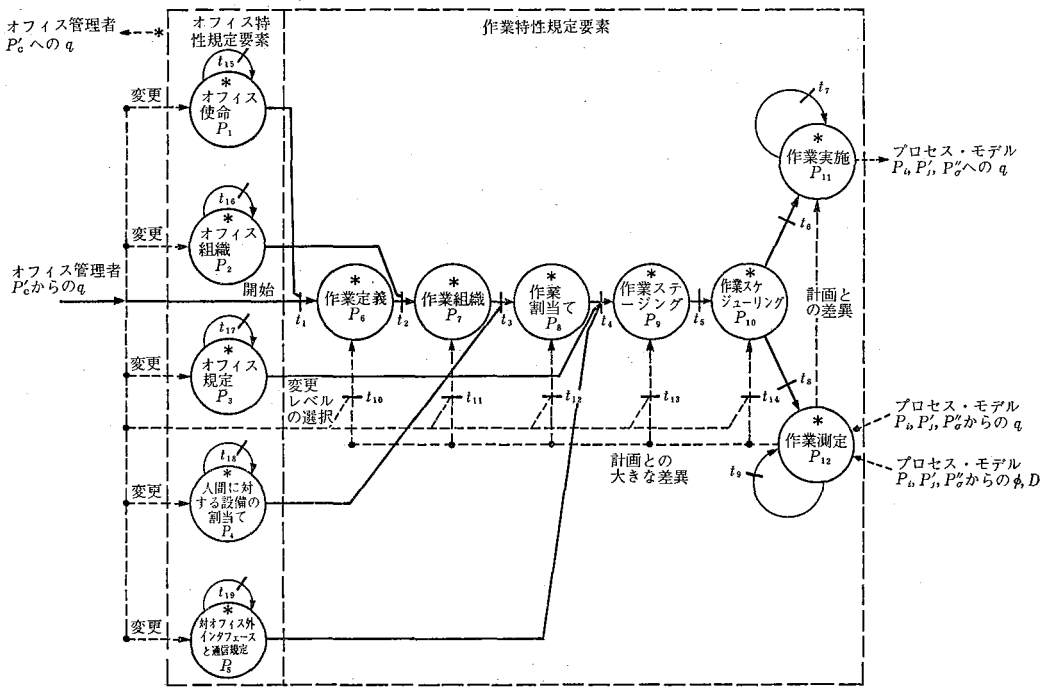


図 4　ワークフロー・コントロール・モデル P_c'' の第1層 P_0
Fig. 4　First layer P_0 of workflow control model P_c''

係づける上位のモデル P_0 の導入が必要となる．そして，P_0 の追加によって，P_c'' は最終的に，

$$P_c''=\{P_i\} \qquad (i=0, 1, \cdots, 12)$$

となり，図4のようになる．

P_c'' の第1層 P_0 は，状態 P_1〜P_{12} までを持つ基本的なペトリ・ネットであり，これらの各状態はそれぞれ対応する下位のペトリ・ネットの初期化のための引き金として働く．つまり，P_0 ネットにおける状態 P_i への遷移によって，P_i ネットが活性化され，さらに P_i ネットの完了によって，P_i 状態に引き続くトランジションがファイアされ，P_0 ネット上の次の状態へ遷移が行われる．なお，図4における実線は P_0 ネット上のトランジション・パスを示し，破線は P_0 ネットとその上位（P_c' ネット）および下位（P_i P_j', P_σ'' ネット）とのトランジション・パスを示している．

さて，次に図4に基づきワークフロー・コントロール・モデルと，プロセス・モデルおよびオフィス管理者モデルとの関係について述べる．いま，P_0 ネットの右側では，ワークフロー・コントロール・モデルと，プロセス・モデル P_i, P_j', P_σ'' との作業受渡しが行われる．このため作業実施 P_{11} は常に稼動しており，プロセスの開始や変更の情報 q をプロセス・モデルに引き渡す．なお，作業特性の変更決定は，作業測定 P_{12} によって計画との差異が発見されたときに行われる．つまり，P_{12} は設定済みの作業スケジュールとプロセス・モデルから得られたデータ (q, D, ϕ) とを比較し，計画からのずれが大きい場合には，それをオフィス管理者 P_c' に伝える．そしてオフィス管理者 P_c' からの変更指示入力 q によって，$P_6, P_7, P_8, P_9, P_{10}$ ネットで修正が行われる．

一方，P_0 ネットの左側では，ワークフロー・コントール・モデルとオフィス管理者 P_c' との作業受渡しが行われる．そしてオフィス管理者からの入力 q によって，トランジション t_1 を介しての作業アクティビティの開始，P_1, P_2, P_3, P_4, P_5 ネットによるオフィス特性の変更，$P_6, P_7, P_8, P_9, P_{10}$ ネットによる作業特性の変更などが行われる．なお，ワークフロー・コントロール・モデル P_c'' からオフィス管理者 P_c' への出力は，各 P_i $(i=1, 2, \cdots, 12)$ からの状態情報 q によって構成される．

また，作業特性規定プロセス P_6〜P_{12} のフレームワークを与えるオフィス構造は，オフィス特性規定プロセス P_1〜P_5 によって生成される．そしてオフィス管理者によって指定された作業アクティビティは，このフレームワークの中で作業特性規定プロセス P_6〜P_{10} によって分解され，スケジュール化されたパーソナル・プロセスとなり，作業者と EDP システムによって実行される．なお，作業者と EDP システムによるパーソナル・プロセスの実行は，作業特性規定プロセス P_{11} と P_{12} によって連続かつ直接に制御される．

6. おわりに

本稿では，オフィス・プロセスにおけるワークフロー・コントロールが，分離可能かつ機械化可能なサブプロセスとして存在しうることを示し，さらにオフィス・プロセスを構成する個々のパーソナル・プロセスからワークフロー・コントロールを切り離し独立させたオフィス・プロセス・モデルを作成した．

このワークフロー・コントロールの機械化によって，効率のよい作業受渡しや会話型による作業の管理統制が可能となるだけでなく，パーソナル・プロセスの機械化によって，下級管理者支援の即応性の向上が期待される．これまでの研究成果から，ワークフロー・コントロールの機械化とオフィス管理者によるパーソナル・プロセスの実行指定によって，オフィスの生産性が大いに向上するとみられている．

本稿のモデルは，オフィスの組織階層における単一監督範囲 (single control span) のみを考慮したものであるが，複数の独立したワークフロー・コントロール・モデルを接続し，職務分掌規定 (organizational charter) に基づき作業定義を分解することによって，垂直的あるいは水平的に複数の管理者が存在する複数監督範囲 (multiple control span) の場合も扱うことができる．このほか，オフィス管理者の定義を機能的管理者要素とプロジェクト管理者要素に分割することによって，マトリックス型組織の場合にもモデル化可能である．

(テクニカル・パブリケーション室　高橋　肇　訳)

参考文献

[1] Statistical Abstract of the United States, 1977.

[2] T. M. Lodahl, *et al.*, *Providing Management Support in the Automated Office*, Corporate Systems, Jun. 1979.

[3] T. M. Steinfatt, *Human Communication*, Bobbs-Merrill Educational Publishing, 1977.

[4] M. D. Zisman, *Representation, Specification, and Automation of Office Procedures*, Wharton School Working Paper 77-09-04.

[5] R. Wetzler, "Operational Analysis—A Team Approach to Productivity Improvement," *Bests Review*, Mar. 1978.

[6] H. Mintzberg, "The Manager's Job: Folklore and Fact," *Harvard Business Review*, July-Aug. 1975.

[7] J. Blair, "Productivity Assessment of Office Information Systems Technology," *Trends and Applications*, 1978 Distributed Processing Conference.

[8] W. Aiken, and J. Lewis, *Office Work Measurement*, Handbook of Business Administration, McGraw-Hill Book Company, 1970.

[9] M. Zisman, "Office Automation: Revolution or Evolution," *Sloan Management Review*, Spring 1978.
[10] J. Steely, "When Management is Automated," *Datamation*, Apr. 1978.
[11] R. Gibson, "Increasing Employee Productivity," *AMACON*, 1976.
[12] M. Kaplan, and S. Schwartz, *Human Judgement and Decision Process in Applied Settings*, Academic Press, 1977.
[13] D. Ness, *A Family of Systems which Process Semi-Structural Information*, Wharton School Working Paper 75-06-08.
[14] J. Peterson, "Petri Nets," *Association of Computing Machinery-Computing Surveys*, Sep. 1977.

執筆者紹介 L. S. バウマン (Lawrence S. Baumann)

Yale 大学で物理学を専攻し，1966年，1967年，1975年にそれぞれ BA, MS, PhD を取得．1974年から上級サイエンティストとして Control Data 社に入社し，X線，レーダー，多スペクトル，写真等のディジタル画像の自動クラスタリング，パターン認識，検査等のアルゴリズムの開発に従事．1979 年に上級システム設計技術者として，Sperry Univac 社に入社し，Systems Research Dept. の電子化オフィス・リサーチ・プロジェクトに参加し，実験的電子郵便システムおよび自動ワークフロー・コントロールの研究を担当．1980年からは，上級自動設計技術者として Computer Aided Design Dept. に所属し，組合せ論理回路の自動合成アルゴリズムおよびコンピュータ・ハードウェア記述言語の開発に従事．

R. D. クープ (Ronald D. Coop)

1960年および1965年にそれぞれ，Iowa 州立大学から BSEE, Minnesota 大学から MSEE を取得．1960年に電子設計技術者として Sperry Univac 社に入社．1962年には，上級コンピュータ・コンサルタントとして，Sperry System Management Organization 社に出向し，米国政府からの受託業務に従事．同社で通信・ミサイル誘導追跡・船舶航法，レーダー・データ処理，天文航法等のコンピュータ・アプリケーションの開発に参加．1972年からは，UNIVAC シリーズ 1100 の通信設計担当マネージャとして，DCA (Distributed Communications Architecture) プロダクトの開発に従事し，1978年に通信と電子化オフィスの研究調査を行うため Systems Research Dept. に所属し，ワークフロー・コントロールとデータベース・システムの研究を担当．1980 年からは，スタッフ・エンジニアとして UNIVAC シリーズ 1100 の将来システムの支援プロセッサ用ソフトウェアの開発に参画し，自動システム始動および自動保守の研究に従事．

論説 Kolmogorov-Chaitin の計算論的情報量と Lempel-Ziv 万能データ圧縮算法

Kolmogrov-Chaitin Computational Complexity and Lempel-Ziv Universal Algorithms for Data Compression

山田真市

要約 記号列のランダム性の測度としては，組合せ論的あるいは確率論的着想による C. E. Shannon の情報量が広く知られている[1]. 従来の離散情報の無雑音圧縮技法はこの Shannon の確率論的情報論に立脚しており，最良の理論圧縮比を与える符号化としては D. A. Huffman の最短符号がある[2]. これに対して，A. N. Kolmogorov と G. J. Chaitin は独立に[3,4]，記号列のランダム性を，その記号列を生成する2進プログラムの極小サイズによって計量する計算論的情報量を導入した. A. Lempel と J. Ziv は，記号の複製をプログラムの基本操作としてこの計算論的情報量の1つの構成的定義を与え[5]，この斬新な着想により離散情報の無雑音圧縮を行う逐次符号化算法を考案した[6]. 本稿ではこの Lempel-Ziv の計算論的情報量と万能データ圧縮算法を述べ，プログラム・パッケージによる圧縮の事例を紹介する.

従来の確率論的着想による符号化では確率的アンサンブルとして情報源の存在を仮定し，その統計的性質のアプリオリな知識を要する. そのため，情報源のエントロピーを計測する前処理が必要である. これに対して，計算論的情報量は個別の記号列に対して定まる. したがって，Lempel-Ziv 算法では情報源のエルゴート性の仮定や情報源のエントロピーを計測する前処理が不要である. この意味で，Lempel-Ziv 算法は前処理なしで任意の記号列に応用できる万能性を持つ[7]. また，本算法が与える理論圧縮比は，対象とする入力記号列の長さを無限に近づけるとき，漸近的に，任意の逐次符号化による無雑音圧縮技法によって到達される最良の圧縮比に等しくなる.

この意味で，本算法は漸近的最良性を示す. 一方，本算法の実現については，探索木データベースを用いた線形時間量の探索算法を使って線形時間量の実現を構成できる. 実際，相当に速い CUP を用いれば，本算法はオンラインや実時間の処理に利用できる. 他方，実現の空間量は探索木データベースのサイズによって支配されるが，これは情報源のエントロピーによって定まる量である. 本算法の応用領域はデータ処理系やデータ通信系の無雑音圧縮であり，さらに，万能性によってテレビ通信，レーダ，ソナー等の音声・画像情報を量子化した記号列の2次圧縮にも応用できる.

Abstract C. E. Shannon's probabilistic measure of information has been widely known as an approach to evaluating the randomness of sequences[1]. Based on Shannon's information theory, many noiseless data compression techniques have been implemented, among which D. A. Huffman's compact coding is known to achieve theoretically optimal compression ratio[2]. In contrast with probabilistic approach, A. N. Kolmogorov and G. J. Chaitin independently introduced algorithmic measure of information, or complexity measure, which evaluates the randomness of individual sequences by the minimal size of the binary programs generating the sequences[3,4]. A. Lempel and J. Ziv gave a new constructive definition of this algorithmic measure by employing a recursive production procedure of individual sequences[5] and developed a universal algorithm for sequential noiseless data compression based on

this innovative idea[6].

This paper outlines Kolmogorov and Chaitin's complexity measure, Lempel and Ziv's measure and their universal algorithms including the variants. Their implementation and compression performance are also briefly described. Every data compression technique based on Shannon's probabilistic theory inevitably presupposes an information source as the probabilistic ensemble of possible sequences and requires considerable effort to evaluate the source entropy for its effective application. On the other hand, the algorithmic measure of information is determined of an individual sequence. Hence, Lempel and Ziv's algorithms presuppose no information sources and require no preprocessing to evaluate their statistical properties. Lempel and Ziv's algorithms are universal in this sense and directly applicable to arbitrarily given sequences[7]. Lempel and Ziv proved that the compression ratio obtained by these algorithms asymptotically approaches to the optimal compression ratio attained by any finite state information lossless encoder, if the length of input sequences approaches to the infinity. Lempel and Ziv's algorithms are asymptotically optimal in this sense.[14]

Furthermore, their linear-time implementations can be constructed by making use of search tree data structure and linear-time pattern matching algorithms, hence applicable to on-line or realtime applications. The space complexity of their implementations is determined by the size of the search tree data base which is the bounded function of individual "source entropy".

1. はじめに

本論に入る前に，データ圧縮技法全般を概観して，Lempel-Ziv 算法の占める位置とその特性を述べる．

データ圧縮の目的は，データの伝送と蓄積の効率を向上し，同時にその所要コストの削減を図ることである．**データ圧縮技法**とは，この目的のためにデータの冗長度(規則性)を抑制し，データを効率のよい表現に変換してデータの量を圧縮する技法である．周知のように，従来から多数のデータ圧縮技法が考案されており，データベース等のファイル系やデータ通信，音声・画像通信等の情報伝送系で実用化されている[8,9,10]．

さて，データの圧縮において，対象とする原データからデータの**サイズ(長さ)**を短縮した圧縮データへの変換を一般に**圧縮**あるいは**符号**といい，逆に圧縮データから原データを再現する変換を**復元**あるいは**復号**という．また，圧縮，復元を実現する工学的変換器をそれぞれ**圧縮器**(あるいは**符号器**)，**復元器**(あるいは**復号器**)という．原データのサイズと圧縮データのサイズの比は圧縮の効率を示し，**圧縮比**という．ところで，データ圧縮技法は一般に**雑音**あるいは**歪**を許容するか否かによって分類できて，それぞれ別個の応用領域を持ち，また別個の理論に基づいている．歪を許容してより高い圧縮比を得る非可逆変換の技法は，元来連続情報であるテレビや，レーダ，ソナー等の信号の1次圧縮に応用され，正則でない圧縮変換によって**有雑音圧縮**を行う．有雑音圧縮を支える理論は情報速度－歪関数の解析理論であり，無雑音圧縮を支える Shannon の情報論の双対理論である[11]．また，圧縮変換には Karhunen-Loève 変換，Fourier 変換等の応用数学における諸変換が応用される[12]．これに対して，歪を許容せず正則変換器系によって圧縮データから原データを忠実に復元する可逆変換の技法を**無雑音圧縮**という．無雑音圧縮は離散情報を対象とし，データ処理系やディジタル・ファクシミリ等の可逆変換が要請される応用領域で用いられる．本稿では，データ処理系において一般的な有限アルファベット上の記号列で表現される離散情報を対象とし，その無雑音圧縮を考察する．

さて，上述のように圧縮変換はデータ(表現)の冗長度(規則性)を抑制する変換である．そこで，記号列の冗長度または逆に記号列のランダム性(無規則性)のとらえ方によって，多様な無雑音圧縮技法が案出される．実際，記号列のランダム性のとらえ方によって，無雑音圧縮技法を，組合せ論的技法，統計的技法，および本稿で述べる計算量論的技法に分けることができよう[3,8]．

組合せ論的技法は素朴に記号列の規則性に着目する圧縮技法である[8]．基本技法としては，ビット写像等によりデータ中の多くの空白やゼロを抑制する**ゼロ・スペース抑制**，データ中の同一記号の連 (run) の長さを符号化することによって連を抑制する**ラン・レングス符号化**，データ中に頻出する記号列のパターンをよりコンパクトな記号列のパターンに置き換える**パターン代入**等があり，これらを組み合わせて応用している．

ソフトウェア・パッケージとしては DCP や Shrink 等がある．この組合せ論的技法の利点は技法が素朴かつ簡潔であって簡単に応用できる点である．古くから，たとえばプログラム・テキストの圧縮等にも，この技法は利用されている．他方，圧縮比を改良するためには別にデータの統計解析が必要であり，また一般にデータの冗長度に見合う圧縮比の向上を実現するのはむずかしい[13]．

統計的技法 は Shannon の確率論的情報論に基づいてデータ中の記号列パターンの出現頻度の統計に着目する技法である．Shannon の**確率論的情報量**は可能な記号列全体のアンサンブルとしての情報源を仮定して定まるが，統計的技法はこの情報源のエントロピーによって圧縮を行う．いま，情報源にエルゴート性を仮定すると，無雑音圧縮の到達可能な最良の理論圧縮比は Shannon の第1符号化定理によって定まり，理論的に D. A. Huffman の最短符号がこの最良の理論圧縮比を実現する[1,2]．したがって，組合せ論的技法と異なり，統計的技法では(理論的に)最良の理論圧縮比を与える符号化算法が存在する．他方，統計的技法を決定するには，情報源のエントロピーを前処理によって計測しておく必要があり，一般に万能性に欠ける．

従来の無雑音圧縮技法は，すべて方法論的にここに述べた組合せ論的技法あるいは統計的技法であり，応用ではこれらの基本技法を適当に組み合わせ，修正して利用している．たとえば，ファクシミリの1次元標準符号化方式 (CCITT 勧告，G3機用) ではランレングス符号化が採用されており，ランレングスの表現には修正 Huffman 符号が使用されている．さらに複数走査線に拡張した2次元符号化方式としても，この方式の拡張形が採用される．

計算量論的技法とは記号列の**計算論的情報量**によって圧縮を行う技法である．記号列の計算論的情報量とは1965年頃 Kolmogorov と Chaitin が独立に導入した概念であり[3,4]，記号列のランダム性を，その記号列を生成する2進プログラムの極小サイズで計量する**計算量**である．A. Lempel と J. Ziv はプログラムの基本操作として記号の複製を使い，記号列の生成過程を用いてこの計算論的情報量の1つの構成的定義を与えた[5]．これを**Lempel-Ziv の計算論的情報量** という．

Lempel-Ziv のデータ圧縮算法はこの着想によって個別の記号列をそれを生成する“極小”2進プログラムに符号化して圧縮する算法である．したがって，Lempel-Ziv 算法は従来の無雑音圧縮技法とは方法論的に全く異なる着想によって圧縮を行う．本稿では Lempel-Ziv 算法とその変形である Lempel-Ziv の incremental 算法を紹介するが，これらの算法は次の特性を持つ[6,14]．

1) 万能性 (universality)[7]
2) 漸近的最良性 (asymptotic optimality)[14]
3) 線形時間量 (linear time complexity)[18-23]

すなわち，計算量論的技法の特徴として，これらの算法は個々の記号列のランダム性にのみ依存し，情報源の統計的性質に依存しない．したがって，組合せ論的技法や統計的技法と異なり，情報源のエントロピーの計測や記号列の規則性の観測を前もって行う必要が

ない．同様に，情報源の定常性あるいはエルゴート性の仮定を要しない．この意味で，これらの算法は，任意の記号列に適用できる**万能性**を持つ．

また，これらの算法によって到達可能な理論圧縮比は，対象とする入力記号列の長さを無限に近づけたとき，漸近的に，工学的に実現可能な逐次圧縮技法によって到達できる最良の理論圧縮比に等しいことが証明できる．この意味で，これらの算法は**漸近的最良性**を示す．

さらに，これらの算法の実現に関しては，探索木のデータ構造とその線形時間探索算法によって，時間量が入力記号列の長さの線形関数で実現できる．この意味で，これら算法は**線形時間量を実現**する．

2. Kolmogorov-Chaitin の計算論的情報量[3,4]

計算論的情報量の着想は1965年頃に A. N. Kolmogorov と G. J. Chaitin によって独立に提案され，P. Martin-Löf らによって議論されたが[15]，応用技術への利用例は少ない．そこでまず，平易にこの着想を述べる．

いま，次のような列の長さ n の10進記号列を考える．

$$22222222222222222222\cdots 2$$

この列は明らかに強い規則性を持ち，ランダムではないであろう．この列は記号2の生成を n 回反復するループ・プログラムで生成できよう．列の長さ n は $\log_2 n$ ビットで表現できるから，記号2の生成とループ制御に c ビットを要するとすれば，このループ・プログラムのサイズは高々

$$\log_2 n + c \text{（ビット）}$$

であり，n が十分大きい場合を考えれば，ほとんどすべての n に対して，プログラム・サイズの下界 $\log_2 n$ が得られる．次に，長さ n の列：

$$12671546889570350354\cdots$$

を見ると，この列から規則性を発見することは難しく，上の列よりランダムであるということができる．この列は自然対数の底 e の 980 桁以降の n 桁であるが，いまこの情報がないならば，この列を生成するには，列をこのまま生成する方法より簡単な方法はないであろう．そこで，この列を生成するプログラムは，この列自身を含み，この列の記号を順に生成するプログラムとなろう．列自身を記憶するための空間のサイズは，1記号に c_1 ビット掛かるとすれば，$c_1 n$ ビットであり，列の記号を順に生成するプログラムに c_2 ビットを要するとすれば，プログラムのサイズは：

$$c_1 n + c_2 \text{（ビット）}$$

であると考えられよう．したがって，ほとんどすべての n に対して，プログラム・サイズの下界 $c_1 n$ が得られる．

この観察から，有限記号列を生成するとき，規則性の強い列については列自身を生成するより簡単な方法があって，規則性が強い列ほど短いプログラムがあり，逆にランダム性が強い列ほどプログラム・サイズは大きくなることがわかる．また，2進記号列の生成を考えるならば，$c_1=1$ で，規則性が最も強い列の生成プログラムの最小のサイズは $\log_2 n$ ビット程度であり，逆にランダム性が強い列の生成プログラムの最小のサイズは n ビット程度であることが推測できよう．情報量の基本単位はビットであるから，プログラムの情報量はプログラムを2進表現したときの長さと考える．ある記号列を与えられたとき，それを生成する2進プログラムは可算無限個存在するが，ここではその中でビット数が最小

のプログラムをとり，その数と記号列の長さ(ビット数)を比較して，記号列のランダム性を計量しようというのである．このビット数が最小のプログラムを**極小プログラム**という．この着想を定式化する．いま，アルファベット $\{0,1\}$ 上の記号の有限列全体の集合をXとする．

$$X=\{0,1\}^*.$$

X上の帰納的部分関数 $\varphi: X\to X$ をとり，φ を計算する計算機を M_φ として，2進プログラム $p\in X$ が M_φ によって2進列 $y\in X$ を生成するとする．すなわち，

$$\varphi(p)=y\vee\varphi(p)\uparrow.$$

このとき，計算機 M_φ 上で列 y を生成する**極小プログラム** p の**サイズ** (情報量) $K_\varphi(y)$ (ビット)は次のように定義される．

定義 1

$$K_\varphi(y)=\begin{cases}\min\{l(p)\mid\varphi(p)=y\}, & (\varphi(p)=y \text{ となる } p \text{ があるとき}),\\ \infty, & (\text{その他のとき}).\end{cases}$$

ここで関数 $l: X\to N$ (Nは自然数)は列の長さである．さらに，列 $x\in X$ を M_φ に入力したときの列 y の条件つき情報量は，帰納的部分関数 $\varphi: X^2\to X$ をとり，計算機 M_φ にプログラム $p\in X$ と列 $x\in X$ を与えたとき，列 y が生成されるとして，

$$\varphi(p,x)=y\vee\varphi(p,x)\uparrow.$$

定義 2

$$K_\varphi(y\mid x)=\begin{cases}\min\{l(p)\mid\varphi(p,x)=y\}, & (\varphi(p,x)=y \text{ となる } p \text{ があるとき}),\\ \infty, & (\text{その他のとき}).\end{cases}$$

と定める．関数 $K_\varphi(y\mid x)$ は関数 φ に関する条件 x のもとでの y の条件つき情報量を与える．ここで，$K_\varphi(y)$ は $K_\varphi(y\mid 1)$ と見なすことができよう．以上の定義は帰納的部分関数 φ に依存するが，S. C. Kleene の標準形定理によって φ に依存しない形に一般化できる．すなわち：

定理 1 (**Kolmogorov-Solomonoff の基本定理**[3,16])

任意の2変数の帰納的部分関数 φ に対して，

$$K_\alpha(y\mid x)\leqq K_\varphi(y\mid x)+c_\varphi$$

を満たす2変数の帰納的部分関数 α が存在する．

ここで c_φ は φ によって定まり，x,y とは独立な定数である．

系

任意の1変数の帰納的部分関数 φ に対して，

$$K_\alpha(y)\leqq K_\varphi(y)+c_\varphi$$

を満たす変数の帰納的部分関数 α が存在する．

ここで c_φ は φ によって定まり，x,y とは独立な定数である．

この関数 α を A. N. Kolmogorov は漸近的に最良 (asymptotically optimal) と呼び，R. J. Solomonoff は万能 (universal) と呼んだ．

この定理と系によって，定数の差を無視すると φ に依存しないで，K_φ が“最小”となる帰納的部分関数 α が存在する．そこで，この $K_\alpha(y\mid x), K_\alpha(y)$ を単に $K(y\mid x), K(y)$ と書き，それぞれ，条件 x のもとでの y の条件つき情報量，y の情報量という．

これを **Kolmogorov-Chaitin の計算論的情報量**あるいは単に **Kolmogorov-Chaitin の計算量**という．

3. Lempel-Ziv の計算論的情報量

Lempel-Ziv の計算論的情報量[5]は，1976年に A. Lempel と J. Ziv が記号の複製を基本操作として記号列の生成過程の計算量として導入した情報量である．平易にこの情報量を紹介する．いま，記号列：

$$R=12323232$$

を考える．$S=123$, $Q=23232$ として，$R=SQ$ とおいたとき，列 Q は列 S の部分列 23 の複製を反復すれば得られる．したがって，列 S と S の部分列 23 の始点2が与えられれば，列 R は列 S から**再生可能**である．すなわち，まず S を複製し，つづいて始点2の S の部分列 23 を繰り返し複製し，列 R の長さだけの記号列をつくれば，実際列 R を得る．いま，一般に $R=SQ$ で Q が R の右端の記号（ここでは2）を除いた列（ここでは 1232323）の部分列であれば，R は S から再生可能である．

次に，記号列：

$$R=12323234$$

を考える．$S=123$, $Q=23234$, $R=SQ$ と置いたとき，列 R は S 列の部分列 23 の複製を反復することによっては再生できない．しかし，R の（すなわち Q の）右端の記号4を除いた列 1232323 を見れば，S の部分列 23 の複製の反復によって Q の右端の記号4を除いた部分列 2323 を再生できる．このような場合，もし列の右端の記号（ここでは4）を記憶しておけば，列 R は S の部分列 23 の2回の複製を行い，つづいて記憶した1記号4を生成することによって得られる．このような場合に，列 R は部分列 S から**生成可能**であると考える．実際，列 R が部分列 S まで生成されたと仮定したとき，S の中の複製に用いる部分列（ここでは 23）の始点の位置と複製する記号数（ここでは4）と R の右端の記号4の3つ組によって列 R は生成可能である．この着想を定式化する．

A を α 個の記号からなるアルファベットとして；

$$A=\{0,1,2,\cdots,\alpha-1\},\qquad \alpha=|A|\ \ (|\cdot|\text{ は濃度}).$$

A 上の記号の有限列全体の集合を A^* とする．長さ $l(x)=k$ の A 上の**語**あるいは**列** x は A^* の元：

$$x=x_1\cdots x_k,\qquad x_1,\cdots,x_k\in A$$

である．いま，列 x の位置 i から位置 j までの部分列を $x(i,j)$ と書く：

$$x(i,j)=x_i\cdots x_j,\qquad (i\leqq j\ \text{のとき}),$$
$$x(i,j)=\lambda,\qquad (i>j\ \text{のとき}).$$

ここで，λ は**空列**を表す．また

$$j\leqq l(x).$$

さらに，各 j，$0\leqq j\leqq l(x)$ に対して，$x(1,j)$ を x の**語頭** (prefix) といい，$x(j,l(x))$ を x の**語尾** (postfix) という．とくに，$j<l(x)$ のとき $x(1,j)$ を x の**真の語頭** (proper prefix) ということにする．また x が y の語頭であることを，次のように書く．

$$x\prec y$$

列 $x=x_1\cdots x_n$ に対して，A^* から A^* への関数 d (delete) を，

$$xd=x_1\cdots x_{n-1}$$

とする．d の関数結合 $\underbrace{d\circ\cdots\circ d}_{n}$ を d^n と書くと，

$$xd^0=x$$
$$xd^n=x(1,l(x)-n)$$

で，とくに，$n\geqq l(x)$ ならば，

$$xd^n=\lambda$$

である．また，列xの部分列全体の集合を $v(x)$ と書く．たとえば，

$$v(0\,1)=\{\lambda, 0, 1, 0\,1\}\subset A^*.$$

この記法のもとで，上述の再生と生成を定式化する．

定義 3[5]

$R, S\in A^*$, $S\prec R$, $R=SQ$ とする．

$v(Rd)\ni Q$ のとき，RはSから**再生可能**であるといい，$S\to R$ と書く．複製に用いる列Sの部分列の始点位置pを再生 $S\to R$ の**ポインタ**という．すなわち，

$$Q=R(p, l(Q+p-1)).$$

例 1 $R=12323232$, $S=123$ では，$Q=23232$, $Rd=1232323$ で，$Q\in v(Rd)$. よって，$S\to R$. 再生のポインタは2である．

定義 4[5]

$S\to Rd$ かつ $l(S)<l(R)$ のとき，空でない列RはSから**生成可能**であるといい，$S\Rightarrow R$ と書く．

例 2 $R=12323234$, $S=123$ では，$Qd=2323$, $Rd=1232323$ で，$S\to Rd$, $l(S)=3<l(R)=8$. よって，$S\Rightarrow R$ となる．

$S\to Rd$ の再生のポインタは2である．ただし，$S\nrightarrow R$ ($S\nrightarrow R$ は $S\to R$ の否定を表す)．

いま$S\Rightarrow R$のとき，SをRの**基**ということにする．任意の列 $x\in A^*$ は初期状態で空列 λ を基とし，生成関係$\Rightarrow$を反復することによって生成できる．たとえば，

$$x=0001101001$$

を考えると，

1) $\lambda\Rightarrow 0$
2) $0\Rightarrow 0001$
3) $0001\Rightarrow 000110$
4) $000110\Rightarrow 000110100$
5) $000110100\Rightarrow 0001101001$

によって，

$$0\cdot 001\cdot 10\cdot 100\cdot 1$$

と語に**分解**できる．このような分解は一意的ではないことに注意する．一般に列 $x\in A^*$ が生成関係 $\Rightarrow$ によって帰納的に，

$$x\mapsto H(x)=X_1X_2\cdots X_m,\quad X_i=x(n_{i-1}+1,\ n_i),\quad 1\leqq i\leqq m,\quad n_0=1.$$

ここで，$x(1, n_{i-1})\Rightarrow x(1, n_i)$, $2\leqq i\leqq m$ と分解されるとき，$H(x)$ をxの**分解歴**という．また各

$$X_i=x(n_{i-1}+1, n_i)$$

を $H(x)$ の**成分**という．いま，生成 $x(1, n_{i-1})\Rightarrow x(1, n_i)$ が，$x(1, n_{i-1})\nrightarrow x(1, n_i)$ を満たすとき，この生成を**最長成分生成**という．また，分解歴で，右端の成分 (X_m) を除くすべての成分が最長成分生成で定まるとき，この分解歴を**最小成分分解歴**という．任意の空でない列xの分解歴は一意ではないが，最小成分分解歴は一意に定まる．これを $E(x)$ と書く．$E(x)$ の成分は相異なるxの部分列となる．

例 3 列 $x=0001101001$ の最小成分分解歴は $E(x)=0\cdot 001\cdot 10\cdot 100\cdot 1$ である．

いま，列xの任意の分解歴 $H(x)$ 中の成分の数を $c_H(x)$ とするとき，Lempel-Ziv の計

算論的情報量は生成の回数を考慮して次のように定められる．

定義 5

Lempel-Ziv の計算論的情報量 $c_{LZ}(x)$ は，

$$c_{LZ}(x)=\min\{c_H(x)\,|\,H(x)\}.$$

ここで，$H(x)$ は列 x のすべての分解歴を変わるとする．

このとき，次の定理が成り立つ．

定理 2

$c_E(x)$ を最小成分分解歴中の成分数とすれば，

$$c_{LZ}(x)=c_E(x).$$

よって，c_{LZ} は構成的である．また，

定理 3

列長 n の任意の列 $x\in A^n$ に対して，

$$c_{LZ}(x)<\frac{n}{(1-\varepsilon_n)\log(n)}. \tag{3-1}$$

ここで，$\varepsilon_n=2\dfrac{1+\log\log(\alpha n)}{\log(n)}$，さらに $\log(x)$ は $\alpha=|A|$ を底とする x の対数である．

系

エントロピーが h の α 記号のエルゴート情報源において，ほとんどすべての列 $x\in A^*$ に対して，列長 $l(x)$ を十分大きくしたとき，漸近的に，

$$c_{LZ}(x)\leqq\frac{hn}{\log(n)}.$$

したがって，Lempel-Ziv の情報量は 2 章の直観的考察をよく反映していることがわかる．

4. 漸近的最良性[14]

工学的に実現可能な変換器モデルとして，有限状態の情報損失のないオートマトンを考える．

定義 6

有限状態符号器 E とは 5 つ組 (S, A, B, g, f) である．ここで，

S：状態の有限集合．$z_1\in S$ を初期状態とする．

A：入力アルファベット，$A=\{0,1,\cdots,\alpha-1\}$，$\alpha=|A|$

B：2 進アルファベット上の出力語の有限集合，$B=\{y_1, y_2, \cdots, y_m\}$

g：$S\times A\to S$：状態推移関数

f：$S\times A\to B$：出力関数

すなわち，符号器 E に入力列 $x=x_1x_2\cdots$，$x_i\in A$ を入力するとき，関係：

$$y_i=f(z_i,x_i),\quad z_{i+1}=g(z_i,x_i),\quad (i=1,2,\cdots)$$

によって，列 $y=y_1y_2\cdots$，$y_i\in B$ を出力する．f, g を拡張して，次のようにも書く．

$$f(z_1,x(1,n))=y(1,n),\qquad g(z_1,x(1,n))=z_{n+1}$$

いま，任意の $x(1,n)\in A^n, n\geqq 1$ に対して，$f(z_1,x(1,n))$ と $g(z_1,x(1,n))$ から $x(1,n)$ が一意的に定まるとき，符号器 E は**情報を損失しない** (information lossless) といい，単に **IL 符号器**ともいう．また，任意の列長 m の入力列 $x(1,m)\in A^m$ に対して，$f(z_1,x(1,m))$ から最初の入力記号 $x(1,1)=x_1$ が一意的に定まるような数 m があるとき，符号器 E を**有限次で情報を損失しない**符号器といい，単に **ILF 符号器** と呼ぶ．ILF 符号器は IL 符

号器であるが，逆は成り立たない．

次に，符号器 $E=(S, A, B, g, f)$ と入力列 $x(1, n)\in A^*$ を与えられたとき，E による $x(1, n)$ の圧縮比を次のように定める．

定義 7

$y(1, n)=f(z_1, x(1, n))$ とし，$y(1, n)$ の列長(ビット)を $L(y(1, n))=\sum_{i=1}^{n} l(y_i)$，$y_i\in B$ とするとき，列長 n の入力列 $x(1, n)\in A^n$ の**圧縮比** $\rho_E(x(1, n))$ を次のように定める．

$$\rho_E(x(1, n))\triangleq L(y(1, n))/(n\log_2\alpha).$$

すなわち，符号器 E の入力列長(ビット)で出力列長(ビット)を割った値が圧縮比である．

さて，α 記号の入力アルファベットを持ち，状態数が s 以下の有限状態 IL 符号器全体のクラスを $E(s)$ とし，$E(s)$ における $\rho_E(x(1, n))$ の最小を $\rho_{E(s)}(x(1, n))$ とする：

$$\rho_{E(s)}(x(1, n))\triangleq\min\{\rho_E(x(1, n))\,|\,E\in E(s)\}.$$

列長 n を無限に近づけたときの圧縮比を，

$$\rho_{E(s)}(x)\triangleq\lim_{n\to\infty}\sup\rho_{E(s)}(x(1, n))$$

とし，さらに状態数が任意有限個でよいようにする：

$$\rho(x)\triangleq\lim_{s\to\infty}\rho_{E(s)}(x).$$

この $\rho(x)$ は，x にのみ依存する有限状態符号器による x の圧縮比を示す．このとき，次の定理によって $\rho_{E(s)}(x(1, n))$ の下界が与えられる．

定理 4 (符号化逆定理)

列長 n の任意の列 $x(1, n)$ に対して，$x(1, n)$ の相異なる部分語 X_i への分解の中で部分語の最大数を $c(x)$ とするとき，$c(x)=\max\{m\,|\,x=X_1\cdots X_m, X_i$ は相異なる$\}$，

$$\rho_{E(s)}(x(1, n))\geqq\frac{c(x)+s^2}{n\log_2\alpha}\log\frac{c(x)+s^2}{4s^2}+\frac{2s^2}{n\log_2\alpha}\tag{4-1}$$

定理 3 の (3-1) より任意の $x=x(1, n)\in A^n$ に対して，

$$c_{LZ}(x)-1\leqq c(x)<\frac{n\log_2\alpha}{(1-\varepsilon_n)\log_2 n},\tag{4-2}$$

かつ，$\lim_{n\to\infty}\varepsilon_n=0$ である．よって (4-1) から，

$$\rho_{E(s)}(x)\geqq\limsup_{n\to\infty}\frac{c(x)\log_2 c(x)}{n\log_2\alpha}.$$

したがって，

$$\rho(x)=\lim_{s\to\infty}\rho_{E(s)}(x)\geqq\limsup_{n\to\infty}\frac{c(x)\log_2 c(x)}{n\log_2\alpha}\tag{4-3}$$

を得る．$0\leqq\rho(x)\leqq1$ であるから，Lempel-Ziv の情報量を同様に正規化し，列長 n を無限に近づければ，

$$\limsup_{n\to\infty}\frac{c_{LZ}(x)\log_2 n}{n\log_2\alpha}\tag{4-4}$$

を得る．よって (4-2), (4-3) より，Lempel-Ziv の正規化情報量 (4-4) は，任意の列 x に対して，列 x に個有のすべての有限状態 IL 符号器に関して最良の圧縮比 $\rho(x)$ の下界となることがわかる．

一方，後述の Lempel-Ziv の incremental 算法を用いて，次の符号化定理を構成的に証明することができる．

定理 5 (符号化定理)

任意の列長 $n>0$ に対して，次の性質を持つ有限状態 ILF 符号器 I が存在する．実際，

Lempel-Ziv の incremental 算法はこのような ILF 符号器 I である.

(1) 任意の $x=x(1,n)\in A^n$ に対して,

$$\rho_I(x)\leqq\frac{c(x)+1}{n\log_2\alpha}\log_2(2\alpha(c(x)+1)).$$

ここで, $c(x)$ は定理4と同様, x の部分語への分解:

$$x=X_1\cdots X_m\quad(X_i\text{ は互いに異なる})$$

に現れる部分語の個数 m の最大数である. すなわち;

$$c(x)=\max\{m\mid x=X_1\cdots X_m,\ X_i\text{ は相異なる}\}.$$

(2) 任意の有限状態数 s と任意の $x=x(1,n)\in A^n$ に対して,

$$\rho_I(x)\leqq\rho_{E(s)}(x)+\delta_s(n).$$

ここで, $\lim_{n\to\infty}\delta_s(n)=0$ である.

(3) 任意の無限入力列 $x\in A^\omega$ に対して, 引き続く列長 n の入力ブロック $x(kn,(k+1)n)$, $k=0,1,2,\cdots$ を符号器 I に入力しながら, ブロックごとの I による圧縮で実現される圧縮比を $\rho_I(x,n)$ とするとき, 任意の $\varepsilon>0$ に対して,

$$\rho_I(x,n)\leqq\rho(x)+\delta_\varepsilon(x,n).$$

ここで, $\lim_{n\to\infty}\delta_\varepsilon(x,n)=\varepsilon$ である.

性質(1)は, incremental 算法 I が有限列 x に対して実現する圧縮比 $\rho_I(x)$ のかなり良い上界を与える. 性質(2)は, この $\rho_I(x)$ が任意の有限状態 IL 符号器 $E(s)$ で到達可能な最良の圧縮比 $\rho_{E(s)}(x)$ に一様に (すなわち, すべての列 x に対して) 近いことを示している. 性質(3)は, incremental 算法の漸近的最良性を示している.

incremental 算法 I は, Lempel-Ziv 算法 U を制限した算法であることから, 直観的に, 算法 U および算法 I が漸近的最良性を示すことは自明である.

いま, 観点を変えて, 任意の $w\in A^l$ と任意の $x=x(1,n)\in A^n$ に対して, Kronecker 積を, $0\leqq i\leqq n-l$ に対して,

$$\delta(x(i+1,i+l),w)=\begin{cases}1 & (x(i+1,i+l)=w\text{ のとき}),\\ 0 & (\text{その他のとき}).\end{cases}$$

とすると, 列 x に語 w が出現する相対頻度は統計的に,

$$P_r[x,w]=\frac{1}{n-l+1}\sum_{i=0}^{n-l}\delta(x(i+1,i+l),w)$$

である.

これを確率測度とすれば, 正規化したエントロピーは,

$$\hat{H}_l{}^n(x)=-\frac{1}{l\log_2\alpha}\sum_{w\in A^l}P_r[x,w]\log_2P_r[x,w]$$

となる. いま,

$$\hat{H}_l(x)=\limsup_{n\to\infty}\hat{H}_l{}^n(x),\qquad\hat{H}(x)=\lim_{l\to\infty}\hat{H}_l(x)$$

と定めるとき,

定理 6

任意の無限列 x に対して,

$$\rho(x)=\hat{H}(x).$$

定理 7

列 x をエントロピー H のエルゴート情報源からとると,

$$P_r[\rho(x)=H]=1.$$

系

列 x をエントロピー H の定常情報源からとると，

$$P_r[\rho(x)]=H$$

を得る．よって，$\rho(x), H(x)$ は共に Shannon のエントロピーと同等の役割を示し，さらに非定常的な場合，$\mathrm{Exp}[\rho(x)]$ と $\mathrm{Exp}[H(x)]$ は拡張したエントロピー概念を与える．

5. Lempel-Ziv 算法 U[6]

本算法は3章の生成過程を bounded delay で行う．算法は圧縮を行う符号算法と復元を行う復号算法よりなる．

5.1 符号算法

符号算法は，入力記号列 $x=x_1 x_2 \cdots, x_i \in A$ を入力順に**内符号語** $\{X_i\}$ へ符号化する**分解** (parsing)：

$$x=X_1 X_2 X_3 \cdots$$

の**内符号化**と，内符号 X_i を2進**外符号** c_i に符号化する **符号手順(外符号化)**

$$c=c_1 c_2 c_3 \cdots$$

よりなる．

5.1.1 分解 (parsing)

分解では3章の最長成分生成による最小成分の分解歴を生成する．すなわち，$A^* \ni x$ の真の語頭 $x(1, j)$ と正整数 $1 \leqq i \leqq j$ に対して，関係：

$$x(i, i+l-1)=x(j+1, j+l), \qquad (l \leqq l(x)-j)$$

を満たす最大非負整数 l を $L(i)$ とし，p を

$$L(p)=\max\{L(i) \mid 1 \leqq i \leqq j\}$$

となる $x(1, j)$ 内の位置とし，x の部分列 $x(j+1, j+L(p))$ を $x(1, j)$ の x への(最長)再生拡大として，p を列 $x(j+1, j+L(p))$ のポインタとする．

5.1.2 パラメタの設定

パラメタ L_s を定め，各内符号 X_i の語長が L_s 以下となるような分解を行う：

$$L_s \geqq l(X_i) \qquad (i=1, 2, 3, \cdots).$$

各外符号 c_i の語長は固定長 L_c とする．本算法では n 記号長の入力バッファ B を設け，B には n 記号からなる最新の入力記号列を蓄える．

符号算法では，関係：

$$L_c=1+\lceil \log (n-L_s) \rceil+\lceil \log L_s \rceil$$

が成り立つ．ここで log の底は $\alpha=|A|$ であり，$\lceil\ \rceil$ は ceil 関数である(5.1.5 項参照)．

5.1.3 初期設定

符号算法の開始時点において，入力列 x の語頭には，$n-L_s$ 個のゼロの列 $Z=0^{n-L_s}$ を付加する．そして1番目の内符号語 X_1 を決定しようとするステップ1のバッファ内容 B_1 を：

$$B_1=Zx(1, L_s)$$

とする．一般に i 番目の内符号語 X_i を決定するステップ i のバッファ内容を B_i と書く．

5.1.4 内符号化算法

次の再帰的手順で内符号化を行う．すなわち，ステップ $i \geqq 1$ において，バッファ内容 B_i を設定した後，

$$X_i=B_i(n-L_s+1, n-L_s+l(X_i))$$

であって，X_i の長さ $l(X_i)-1$ の語頭 X_id が $B_i(1, n-L_s)$ の $B_i(1, n-1)$ への最長再生拡大となる X_i を求める．

5.1.5 外符号化算法

ステップ $i\geqq 1$ において，X_i を決めるポインタを p_i とするとき，X_i の外符号語 c_i は，

$$c_i = c_{i1}\, c_{i2}\, c_{i3}$$

であり，

c_{i1} は長さ $\lceil \log(n-L_s) \rceil$ を持つ p_i-1 の α 進表現，

c_{i2} は長さ $\lceil \log(L_s) \rceil$ を持つ $l(X_i)-1$ の α 進表現，

c_{i3} は X_i の右端の記号，すなわち，B_i の $n-L_s+l(X_i)$ 番目の記号

である．

5.1.6 バッファの更新

バッファの内容 B_i を B_{i+1} に更新するには，B_i の最初の $l(X_i)$ 個の記号を左にシフトアウトして，列 x の次の $l_i=l(X_i)$ 個の記号を右からバッファに読み込む．すなわち，

$$B_{i+1} = B_i(l_i+1, n)\, x(h_i+1, h_i+l_i)$$

で，h_i は B_i の最右の記号 $B_i(n, n)$ が入る B_{i+1} 内の位置に等しい．

5.2 復 号 算 法

復号は符号算法の逆である．すなわち次の再帰的手順である．

5.2.1 バッファの設定

ステップ i で復号する内符号語 X_i を蓄える長さ $n-L_s$ のバッファ D を設定し，開始時(ステップ1)ではバッファに $n-L_s$ 個のゼロの列 $Z=0^{n-L_s}$ を入れておく．

5.2.2 内符号の復号算法

ステップ $i\geqq 1$ において，c_i の最初の $\lceil \log(n-L_s) \rceil$ 個の記号と続く $\lceil \log L_s \rceil$ 個の記号から，p_i, l_i を求める．

そして，次の手順を l_i-1 回反復する．

手順: D_i の位置 p_i にある記号 $D_i(p_i, p_i)$ をとりだし，D_i を左へ1記号シフトして，$D_i(p_i, p_i)$ を位置 $n-L_s$ に入れる．

l_i-1 回の反復が終了したならば，c_i の次の記号，すなわち，X_i の最終記号をとり，D_i を1記号左へシフトして位置 $n-L_s$ に入れる．

この結果，求められるバッファの内容はステップ $i+1$ 用の D_{i+1} であり，また，

$$X_i = D_{i+1}(n-L_s-l_i+1, n-L_s)$$

として，X_i が求められる． (算法終り)

この算法をこのまま実現するとき，最長再生拡大を求める pattern matching のための時間量は $O(n^2)$ となることを注意する．ここで，n は入力列の列長 $(n=l(x))$ である．

6. Lempel-Ziv の incremental 算法 *I*[14]

本算法は Lempel-Ziv 算法 U において再生拡大の pattern matching を既定義の内符号語の語頭に限定し，さらに再生拡大の最長性の条件を緩和して得られる算法である．

本算法も符号算法と復号算法よりなる．

6.1 符 号 算 法

符号算法は可変長 $l(x)$ の入力列 x を内符号語 $\{X_m\}$ に分解する incremental 分解：

$$x \mapsto X_1\, X_2\, X_3 \cdots .$$

と，各内符号語 X_m を2進外符号語 Y_m に符号化する外符号手順：

$$X_m \mapsto Y_m \qquad (m=1, 2, 3, \cdots)$$

よりなる.

6.1.1 incremental 分解

incremental 分解では 6.1.2 項に述べる分解基準を満たすように，可変長 $l(x)$ の入力列を $p+1$ 語の内符号語：

$$X_m=x(n_{m-1}+1,\ n_m) \qquad (m=1, 2, 3, \cdots, p+1)$$

へ分解する．すなわち

$$x \mapsto X_1 X_2 \cdots X_{p+1}.$$

である.

6.1.2 分解基準

1) 最終の X_{p+1} を除き，語 $X_1, \cdots, X_p$ は互いに相異なる.

2) すべての $m=2, 3, \cdots, p+1$ に対して

$i \in \{1, \cdots, m\}$ が存在して，

$$X_m d \prec X_i \cdots X_{p+1}=x(n_{i-1}+1, l(x)).$$

3) すべての $m=2, 3, \cdots, p+1$ に対して，

$$X_m \prec X_i \cdots X_{p+1}$$

となる $i \in \{1, \cdots, m\}$ は存在しない.

いま，最も単純に既定義の内符号語 $\{X_0, X_1, \cdots, X_{m-1}\}$, $X_0=\lambda$ が与えられているとき，

$X_{i_m} \prec x(n_{m-1}+1,\ l(x))$ となる**最長**の $X_{i_m} \in \{X_0, X_1, \cdots, X_{m-1}\}$

をとり，$X_m d=X_{i_m}$ とすれば 1)，2)，3) を満たす.

そこで，内符号化算法を行う.

6.1.3 内符号化算法

次の再帰的手順で内符号化を行う.

1) 開始時点では，$m=1$, $n_0=0$, $X_0=\lambda$ とする.

2) ステップ $m \geqq 1$ においては，ステップ m までの内符号語 $\{X_0, \cdots, X_{m-1}\}$ が用意されている．このとき，

$X_{i_m} \prec x(n_{m-1}+1,\ l(x))$ となる最長の $X_{i_m} \in \{X_0, \cdots, X_{m-1}\}$

を求めて，$n_m=n_{m-1}+1+l(X_{i_m})$, $X_m=x(n_{m-1}+1, n_m)$ とする.

6.1.4 外符号化算法

i_m から X_{i_m} を求める探索木データベースを用意すると，incremental 分解で得られる各内符号語 X_m は組 $\langle i_m, x_{n_m} \rangle$ によって次のように定まる.

$$X_m=X_{i_m} x_{n_m}.$$

そこで，最適化した修正 2 進語頭表現によって，符号化：

$$\langle i_m, x_{n_m} \rangle \mapsto Y_m$$

を行う．ここでは詳細は省略する.

6.2 復 号 算 法

再帰的に各繰返しステップ $m \geqq 1$ において，復号化：

$$Y_m \mapsto \langle i_m, x_{n_m} \rangle$$

によって，$X_m=X_{i_m} x_{n_m}$ を復元する．詳細は省略する.

7. 線形時間量の実現[17-23]

情報伝送系において，以上の算法を応用するためには少なくとも線形時間量(linear time

complexity) の実現を構成しなければならない．以上の算法の時間量は内符号化算法における pattern matching の時間量と考えてよい．線形時間の pattern matching 算法は 1970 年の Knuth-Morris-Pratt の研究以来急速に進み[21,22]，探索木のデータ構造を用いた Weiner の算法や McCreight の算法が知られている．本算法の実現においても，探索木データベースを採用することによって，線形時間量の実現を行っている．

8. データ圧縮の事例

日本ユニバック(株)と Sperry Research Center において，incremental 算法によるデータ圧縮プログラムを UNIVAC シリーズ 1100 で作成した．このデータ圧縮プログラム・ライブラリによるファイルの圧縮例のごく一部を以下に示す．この結果は日本ユニバック(株)の宗像清治による．

	ファイル	内 容	コード	圧縮比
1.	PRG	プログラム	FD, binary	0.82
2.	GTP	印書データ	FD,	0.09
3.	CFH	データ	FD, binary	0.28
4.	MDA	マスター	ASCII, binary	0.39
5.	MDG	マスター	ASCII, binary	0.40
6.	EMA	マスター	ASCII, binary	0.20

注) 圧縮比=圧縮データ/原データ

9. おわりに

本稿は，日本ユニバック(株)と Sperry Research Center との合意に基づき進められて来たデータ圧縮法の共同研究開発タスク・フォースの結果の一部を述べたものである．M. Cohn, W. L. Eastman, A. Lempel, J. Ziv ならびに J. Speroni, 宗像清治，森俊治の各氏に感謝する．とくに基礎研究では W. L. Eastman に，算法の実現と事例については W. L. Eastman と宗像の両氏によるところが大きい．

参考文献 [1] C. E. Shannon, and W. Weaver, *The Mathematical Theory of Communication*, Univ. of Illinois Press, 1949.

[2] D. A. Huffman, "A Method for the Construction of Minimum-Redundancy Codes," *Proc. IRE*, Vol. 40, 1952, pp. 1098-1101.

[3] A. N. Kolmogorov, "Three Approaches to The Quantitative Definition of Information," Probl. Inform. Transmission, Vol. 1, 1965, pp. 1-7.

[4] G. J. Chaitin, "Information-theoretic computational complexity," *IEEE Trans.*, Vol. IT-20, 1974, pp. 10-25.

[5] A. Lempel and J. Ziv, "On the Complexity of Finite Sequences," *IEEE Trans.*, Vol. IT-22, No. 1, 1976, pp. 75-81.

[6] J. Ziv and A. Lempel, "A Universal Algorithm for Sequential Data Compression," *IEEE Trans.*, Vol. IT-23, No. 3, 1977, pp. 337-343.

[7] L. D. Davisson, "Universal Noiseless Coding," *IEEE Trans.*, Vol. IT-19, No. 6, 1973, pp. 783-795.

[8] S. S. Ruth and P. J. Kreutzer, "Data Compression for Large Business Files," *Datamation*, Sept. 1972, pp. 62-66.

[9] J. Aronson, "Data Compression—A Comparison of Methods," *NBS Special Publication 500-12, U. S. Dept. of Commerce*, 1977.

[10] C. Holborrow, J. McNemer and P. Stoneburner, "A Review of Data Compression Algorithms," *U. S. Dept. of Commerce Technical Memorandum TM 122-76*, 1976.

[11] T. Berger, *Rate Distortion Theory—A Mathematical Basis for Data Compression*, Prentice-Hall, 1971.
[12] A. Rosenfeld and A. C. Kak, *Digital Picture Processing*, Academic Press, 1976.
[13] たとえば,
K. G. Gray, "Upper Bound on Compression Ratio for Run-Length Encoding," *Proc. IEEE*, Jan. 1972, p. 148.
[14] J. Ziv and A. Lempel, "Compression of Individual Sequences via Variable—Rate Coding," *IEEE Trans.*, Vol. IT-24, No. 5, 1978, pp. 530-536.
[15] P. Martin-Löf, "The Definition of Random Sequences," *Information and Control* 9, 1966, pp. 602-619.
[16] R. J. Solomonoff, "A formal theory of inductive inference. Part I.," *Information and Control 7*, 1964, pp. 1-22.
[17] S. Even and M. Rodeh, "Economical Encoding of Commas between Strings," TR-54, *Dept. Comp. Sci.*, Technion, Haifa, Israel, 1975, also *in Comm. of the ACM*, Vol. 21, No. 4, 1978, pp. 315-317.
[18] M. Rodeh, V. R. Pratt and S. Even, "A Linear Algorithm for Finding Repetitions and its Applications in Data Compression," TR-72, *Dept. Comp. Sci.*, Technion, Haifa, Israel 1976.
[19] P. Weiner, "Linear Pattern Matching Algorithms," *IEEE 14th Ann. Symp. on Switching and Automata Theory*, 1973, pp. 1-11.
[20] E. M. McCreight, "A Space-Economical Suffix Tree Construction Algorithm," *Journal of the ACM*, Vol. 23, No. 2, 1976, pp. 262-272.
[21] D. E. Knuth, J. H. Morris, and V. R. Pratt, "Fast pattern matchings in strings," *Comp. Sci. Rep.*, STAN-CS-74-440, Stanford Univ., 1974.
[22] J. H. Morris, and V. R. Pratt, "A linear patten-matching algorithm," TR-40, Computing Center, University of California at Berkeley, 1970.
[23] M. Rodeh, V. R. Pratt and S. Even, "Linear Algorithm for Data Compression via String Matching," *Journal of the ACM*, Vol. 28, No. 1, 1981, pp. 16-24.

執筆者紹介 山 田 真 市 (Shinichi Yamada)
昭和12年生, 34年東京大学理学部数学科卒業. 同年日本ユニバック(株)入社. 45年 Harvard 大学大学院修士課程修了. SM (応用数学). プログラミング等に従事. 現在に至る.
日本数学会, AMS, ASL, IEEE, ACM 会員.

論説

3次元磁場解析への有限要素法の適用 ——ベクトル・ポテンシャルの取扱い

An Application of The Finte Element Method to The Analysis of 3 ——Dimensional Magnetic Field

渡 部 義 維

要 約 構造力学の領域で著しい成果をあげた有限要素法は，電気・電子分野にも浸透しており，とくに磁気ヘッド，電動機，マグネトロン等，磁気回路設計の有力な道具となっている．ベクトル・ポテンシャルを導入して，定常電流に伴う磁場——静磁場——の変分原理を議論し，3次元静磁場問題への有限要素法の適用を試みた．

入力カードの処理，連立方程式の解法等は，汎用構造解析プログラム NASTRAN を利用した．

Abstract By making use of NASTRAN which has a wide variety of functions for analyzing heat conduction problems and an analogy of basic equations, trials of solving such subjects as diffusion of neutrons and an analysis of 2-dimensional magnetic field, have been reported.

Though a lot of applications of the finite element method to the problems concerning electromagnetic field have also been reported, most of them are restricted to particular cases where the vector potential can practically be treated as a scalar potential.

In Nippon UNIVAC Kaisha, Ltd., they have accomplished formulations by finite element method for vector potential with variational principle; in addition they have reported an advanced attempt of treating an arbitrary 3-dimensional static magnetic field, by using solid elements of NASTRAN.

Equations for the magnetic field with the given current density J and its distribution are:

$$\text{rot } \boldsymbol{H}=\boldsymbol{J} \qquad (\boldsymbol{H}\text{: Magnetic field}) \tag{1}$$

$$\text{div } \boldsymbol{B}=0 \qquad (\boldsymbol{B}\text{: Magnetic flux density}) \tag{2}$$

$$\boldsymbol{H}=\mu\boldsymbol{H} \qquad (\mu\text{: Magnetic permeability}) \tag{3}$$

Substituting a vector potential $\boldsymbol{A}$ which satisfies the Eq. of $\boldsymbol{B}=\text{rot } \boldsymbol{A}$ for $\boldsymbol{B}$, it follows that

$$\frac{1}{\mu}\text{ rot }\boldsymbol{B}=\frac{1}{\mu}\text{ rot rot }\boldsymbol{A}=\boldsymbol{J} \tag{4}$$

Regarding Eq. (4) as the basic equation, in order to obtain a functional Φ which has Eq. (4) as the Euler's, the follwing is to be expanded.

$$\delta\Phi=\int_V\left(\frac{1}{\mu}\text{ rot rot }\boldsymbol{A}-\boldsymbol{J}\right)\cdot\delta\boldsymbol{A}\,dV \tag{5}$$

$$\delta\Phi=\frac{1}{2\mu}\int_V(\nabla\times\boldsymbol{A})\cdot(\nabla\times\boldsymbol{A})dV-\int_V\boldsymbol{J}\cdot\boldsymbol{A}dV \tag{6}$$

Furthermore, by applying the tetrahedral element which is used in the structural analysis, and by formulating the stationary condition of the functional Φ, the following simultaneous equations expressing a superposition of all N elements are obtained.

$$\sum_N(K_N+F_N)=0 \tag{7}$$

Solving Eq. (7) gives vector potential (A_x, A_y, A_z), which can be transformed into magnetic flux density (B_x, B_y, B_z).

Maxwell's equation in the above formulation can not be solved by NASTRAN. However, they have elaborated a plan so that input data concerning solid elements (TETRA, WEDGE, HEXA 1, HEXA 2) of NASTRAN can directly be used as it is, and they have attached a routine for building a matrix to compute K_N, and another routine for the transformation $\boldsymbol{A}\rightarrow\boldsymbol{B}$ to NASTRAN.

The results of the analysis on the following cases are in good accordance with the theoretical ones. ① If a potential, of which magnitude is proportional to radius, is given in the circumferential direction, a uniform magnetic fild in the direction of Z-axis within the cylinder is produced. ② Being

given a current in the direction of Z-axis along the center line of the cylinder, the magnetic field within the cylindrical area is produced in the circumferential direction.

1. はじめに

本稿は6年前，静磁場解析へ有限要素法の適用を試み昭和51年6月ユニバック研究会技術計算分科会(第1回)で発表したものに若干の加筆をしたものである．その後，電気・電子分野で有限要素法の真価が認識されるにいたり，半導体デバイス内のミクロ挙動，電子機器の電場・磁場の解析等に欠かせないものとなった．磁場関係では磁気ヘッドの磁界解析を中心に数多くの成果が発表されているが，3次元磁場解析の実例はまだ数少ない．

本稿は，変分原理を利用してベクトル・ポテンシャルに対して有限要素法の定式化を行い，任意形状の3次元静磁場解析を可能にした一試行である．

上記のような環境下で，最近，ユーザからの磁場解析への要求がたかまってきている．そこで，日本ユニバック(株)では磁場解析専用の要素を NASTRAN に登録し，正式に提供する予定である．

2. 基礎方程式

電流密度 $\boldsymbol{J}$ とその分布が知られている磁場に対する方程式は，

$$\mathrm{rot}\ \boldsymbol{H}=\boldsymbol{J} \qquad (\boldsymbol{H}\ \text{は磁場}), \tag{2-1}$$

$$\mathrm{div}\ \boldsymbol{B}=0 \qquad (\boldsymbol{B}\ \text{は磁束密度}), \tag{2-2}$$

$$\boldsymbol{B}=\mu\boldsymbol{H} \qquad (\mu\ \text{は透磁率}). \tag{2-3}$$

$\boldsymbol{B}$ の代わりに $\boldsymbol{B}=\mathrm{rot}\,\boldsymbol{A}$ なるベクトル・ポテンシャルを導入すると，(2-1)は

$$\frac{1}{\mu}\mathrm{rot}\ \boldsymbol{B}=\frac{1}{\mu}\mathrm{rot\ rot}\ \boldsymbol{A}=\boldsymbol{J} \tag{2-4}$$

となる．(2-4)は

$$\frac{1}{\mu}\mathrm{rot\ rot}\ \boldsymbol{A}=\mathrm{grad\ div}\ \boldsymbol{A}-\nabla^2\boldsymbol{A}$$

であり，$\mathrm{div}\,\boldsymbol{A}$ の任意性によって，$\mathrm{div}\,\boldsymbol{A}=0$ とおくと，

$$\frac{1}{\mu}\nabla^2\boldsymbol{A}=-\boldsymbol{J}$$

となる．

$\boldsymbol{A}$ および $\boldsymbol{J}$ がスカラー量とみなせる場合は，この Poisson 方程式は熱伝導等の解法で容易に解きうる．

本稿では，(2-4)を基礎方程式とする．

3. 変分原理の適用

式(2-4)を Euler 方程式とする汎関数 Φ を求めるため，

$$\delta\Phi=\int_V\left(\frac{1}{\mu}\mathrm{rot\ rot}\ \boldsymbol{A}-\boldsymbol{J}\right)\cdot\delta\boldsymbol{A}\,dV \tag{3-1}$$

を展開する．

$\boldsymbol{n}$ を法線ベクトルとして，

$$\delta\Phi=\int_V\left(\frac{1}{\mu}\nabla\times(\nabla\times\boldsymbol{A})-\boldsymbol{J}\right)\cdot\delta\boldsymbol{A}dV$$

$$=\frac{1}{\mu}\int_\Gamma\langle(\nabla\times\boldsymbol{A})\times\delta\boldsymbol{A},\boldsymbol{n}\rangle d\Gamma+\frac{1}{\mu}\int_V\langle\nabla\times\boldsymbol{A},\delta(\nabla\times\boldsymbol{A})\rangle dV-\delta\int_V\boldsymbol{J}\cdot\boldsymbol{A}dV$$

$$=\frac{1}{\mu}\int_{\Gamma}\langle(\nabla\times\boldsymbol{A})\times\delta\boldsymbol{A},\boldsymbol{n}\rangle d\Gamma+\delta\frac{1}{2\mu}\int_{V}(\nabla\times\boldsymbol{A})\cdot(\nabla\times\boldsymbol{A})dV-\delta\int_{V}\boldsymbol{J}\cdot\boldsymbol{A}dV,$$

よって，汎関数は

$$\Phi=\frac{1}{2\mu}\int_{V}(\nabla\times\boldsymbol{A})\cdot(\nabla\times\boldsymbol{A})dV-\int_{V}\boldsymbol{J}\cdot\boldsymbol{A}dV. \tag{3-2}$$

となる．

4. 要素の形状関数

構造解析で使用される四面体要素 (tetrahedron) の形状関数を適用する．他の要素は NASTRAN 内部で TETRA に分割される．

以下，$A_x\rightarrow u$, $A_y\rightarrow v$, $A_z\rightarrow w$ と記述することとする．

四面体要素の形状関数

$$\begin{aligned}u&=q_1+q_2x+q_3y+q_4z\\ v&=q_5+q_6x+q_7y+q_8z\\ w&=q_9+q_{10}x+q_{11}y+q_{12}z\end{aligned} \tag{4-1}$$

のマトリックス表現は次のとおりである．

$$\boldsymbol{A}=(u\ v\ w)=(1\ x\ y\ z)\begin{pmatrix}q_1&q_5&q_9\\q_2&q_6&q_{10}\\q_3&q_7&q_{11}\\q_4&q_8&q_{12}\end{pmatrix}=\boldsymbol{D\alpha}. \tag{4-2}$$

式(4-2)に節点値を入れると，

$$\boldsymbol{u}^e=\begin{pmatrix}u_1&v_1&w_1\\u_2&v_2&w_2\\u_3&v_3&w_3\\u_4&v_4&w_4\end{pmatrix}=\begin{pmatrix}1&x_1&y_1&z_1\\1&x_2&y_2&z_2\\1&x_3&y_3&z_3\\1&x_4&y_4&z_4\end{pmatrix}\begin{pmatrix}q_1&q_5&q_9\\q_2&q_6&q_{10}\\q_3&q_7&q_{11}\\q_4&q_8&q_{12}\end{pmatrix}=\boldsymbol{C\alpha}. \tag{4-3}$$

式(4-2)，(4-3)より $\boldsymbol{\alpha}$ を消去すると，

$$\begin{aligned}\boldsymbol{A}=(u\ v\ w)&=\boldsymbol{DC}^{-1}\boldsymbol{u}^e\\&=(1\ x\ y\ z)\begin{pmatrix}h_{11}&h_{12}&h_{13}&h_{14}\\h_{21}&h_{22}&h_{23}&h_{24}\\h_{31}&h_{32}&h_{33}&h_{34}\\h_{41}&h_{42}&h_{43}&h_{44}\end{pmatrix}\begin{pmatrix}u_1&v_1&w_1\\u_2&v_2&w_2\\u_3&v_3&w_3\\u_4&v_4&w_4\end{pmatrix}.\end{aligned} \tag{4-4}$$

ここで，$a_i=h_{1i}, b_i=h_{2i}, c_i=h_{3i}, d_i=h_{4i}$ とし，$N_i=a_i+b_ix+c_iy+d_iz$ とすると，

$$\left.\begin{aligned}u&=\sum_{i=1}^{4}N_iu_i\\ v&=\sum_{i=1}^{4}N_iv_i\\ w&=\sum_{i=1}^{4}N_iw_i\end{aligned}\right\} \tag{4-5}$$

$$\left.\begin{aligned}\frac{\partial A_x}{\partial x}&=\frac{\partial u}{\partial x}=\sum_{i}^{4}b_iu_i=h_{21}u_1+h_{22}u_2+h_{23}u_3+h_{24}u_4=q_2\\ \frac{\partial A_x}{\partial y}&=\frac{\partial u}{\partial y}=\sum_{i}^{4}c_iu_i=h_{31}u_1+h_{32}u_2+h_{33}u_3+h_{34}u_4=q_3\\ \frac{\partial A_x}{\partial z}&=\frac{\partial u}{\partial z}=\sum_{i}^{4}d_iu_i=h_{41}u_1+h_{42}u_2+h_{43}u_3+h_{44}u_4=q_4\\ \frac{\partial A_y}{\partial x}&=\frac{\partial v}{\partial x}=\sum_{i}^{4}b_iv_i=h_{21}v_1+h_{22}v_2+h_{23}v_3+h_{24}v_4=q_6\end{aligned}\right|$$

$$\left.\begin{aligned}
&\frac{\partial A_y}{\partial y}=\frac{\partial v}{\partial y}=\sum_i^4 c_i v_i=h_{31}v_1+h_{32}v_2+h_{33}v_3+h_{34}v_4=q_7\\
&\frac{\partial A_y}{\partial z}=\frac{\partial v}{\partial z}=\sum_i^4 d_i v_i=h_{41}v_1+h_{42}v_2+h_{43}v_3+h_{44}v_4=q_8\\
&\frac{\partial A_z}{\partial x}=\frac{\partial w}{\partial x}=\sum_i^4 b_i w_i=h_{21}w_1+h_{22}w_2+h_{23}w_3+h_{24}w_4=q_{10}\\
&\frac{\partial A_z}{\partial y}=\frac{\partial w}{\partial y}=\sum_i^4 c_i w_i=h_{31}w_1+h_{32}w_2+h_{33}w_3+h_{34}w_4=q_{11}\\
&\frac{\partial A_z}{\partial z}=\frac{\partial w}{\partial z}=\sum_i^4 d_i w_i=h_{41}w_1+h_{42}w_2+h_{43}w_3+h_{44}w_4=q_{12}
\end{aligned}\right\} \qquad (4\text{-}6)$$

となる.

汎関数 Φ に必要な微分 q_2〜q_{12} は(4-1)での係数となっている.

5. 汎関数 Φ の停留条件

式(4-7)を使って Φ を展開し, Φ の停留条件

$$\left.\begin{aligned}
&\frac{\partial \Phi}{\partial u_i}=0\\
&\frac{\partial \Phi}{\partial v_i}=0\\
&\frac{\partial \Phi}{\partial w_i}=0
\end{aligned}\right\} \qquad (i=1\text{〜}4) \qquad (5\text{-}1)$$

を求める.

Φ の第1項 Φ_1 は,

$$\Phi_1=\frac{1}{2\mu}\int_V\left[\left(\frac{\partial w}{\partial y}-\frac{\partial v}{\partial z}\right)^2+\left(\frac{\partial u}{\partial z}-\frac{\partial w}{\partial x}\right)^2+\left(\frac{\partial v}{\partial x}-\frac{\partial u}{\partial y}\right)^2\right]dV,$$

$$=\frac{1}{2\mu}\int_V[q_{11}{}^2-2q_{11}q_8+q_8{}^2+q_4{}^2-2q_4q_{10}+q_{10}{}^2+q_6{}^2-2q_6q_3+q_3{}^2]dV$$

であるから, $\partial\Phi_1/\partial u_1$ は,

$$\frac{\partial \Phi_1}{\partial u_1}=\frac{1}{2\mu}\int_V\frac{\partial}{\partial u_1}(q_4{}^2-2q_4q_{10}-2q_6q_2+q_3{}^2)dV$$

となる. ここで Δ を体積とすると,

$$\frac{\partial \Phi_1}{\partial u_1}=\frac{1}{\mu}\left[\sum_i^4(d_1d_i+c_1c_i)u_i-d_1\sum_i^4 b_iw_i-c_1\sum_i^4 b_iv_i\right]\Delta$$

$$=\frac{\Delta}{\mu}[(h_{41}h_{41}+h_{31}h_{31})u_1+(h_{41}h_{42}+h_{31}h_{32})u_2+(h_{41}h_{43}+h_{31}h_{33})u_3$$

$$+(h_{41}h_{44}+h_{31}h_{34})u_4-h_{41}h_{21}w_1-h_{41}h_{22}w_2-h_{41}h_{23}w_3-h_{41}h_{24}w_4$$

$$-h_{31}h_{21}v_1-h_{31}h_{22}v_2-h_{31}h_{23}v_2-h_{31}h_{24}v_4]$$

$$\frac{\partial \Phi_1}{\partial u_2}=\frac{1}{2\mu}\int_V\frac{\partial}{\partial u_2}(q_4{}^2-2q_4q_{10}-2q_6q_3+q_3{}^2)dV$$

$$=\frac{1}{\mu}\left[\sum_i^4(d_2d_i+c_2c_i)u_i-d_2\sum_i^4 b_iw_i-c_2\sum_i^4 b_iv_i\right]\Delta$$

となる.

以上をまとめると,

$$\frac{\partial \Phi_1}{\partial u_j}=\frac{\Delta}{\mu}\left[\sum_i^4(d_jd_i+c_jc_i)u_i-c_j\sum_i^4 b_iv_i-d_j\sum_i^4 b_iw_i\right]$$

$$\left.\begin{aligned}\frac{\partial \Phi_1}{\partial v_j}&=\frac{\Delta}{\mu}\left[-b_j\sum_i^4 c_i u_i+\sum_i^4(d_j d_i+b_j b_i)v_i-d_j\sum_i^4 c_i w_i\right]\\ \frac{\partial \Phi_1}{\partial w_j}&=\frac{\Delta}{\mu}\left[-b_j\sum_i^4 d_i u_i-c_j\sum_i^4 d_i v_i+\sum_i^4(c_j c_i+b_j b_i)w_i\right]\end{aligned}\right\}\quad (j=1\sim4). \tag{5-2}$$

Φ の第2項 Φ_2 は,

$$\Phi_2=\int_V(J_x u+J_y v+J_z w)dV$$

$$u=\sum_i^4 N_i u_i,\quad v=\sum_i^4 N_i v_i,\quad w=\sum_i^4 N_i w_i$$

であることから,

$$\left.\begin{aligned}\frac{\partial \Phi_2}{\partial u_j}&=\int_V J_x N_j dV\\ \frac{\partial \Phi_2}{\partial v_j}&=\int_V J_y N_j dV\\ \frac{\partial \Phi_2}{\partial w_j}&=\int_V J_z N_j dV\end{aligned}\right\}\quad (j=1\sim4) \tag{5-3}$$

となる. (5-2) と (5-3) から得られる要素に関する方程式をまとめると,

$$\begin{bmatrix}[K_{xx}] & [K_{xy}] & [K_{xz}]\\ [K_{xy}^T] & [K_{yy}] & [K_{yz}]\\ [K_{xz}^T] & [K_{yz}^T] & [K_{zz}]\end{bmatrix}\begin{Bmatrix}\{u\}\\ \{v\}\\ \{w\}\end{Bmatrix}=\begin{Bmatrix}\{J_x\}\\ \{J_y\}\\ \{J_z\}\end{Bmatrix} \tag{5-4}$$

であり, 各々の要素は,

$$K_{xxij}=\int_V\frac{1}{\mu}\left(\frac{\partial N_i}{\partial y}\frac{\partial N_j}{\partial y}+\frac{\partial N_i}{\partial z}\frac{\partial N_j}{\partial z}\right)dV=\frac{\Delta}{\mu}(h_{3i}h_{3j}+h_{4i}h_{4j})$$

$$K_{yyij}=\int_V\frac{1}{\mu}\left(\frac{\partial N_i}{\partial x}\frac{\partial N_j}{\partial x}+\frac{\partial N_i}{\partial z}\frac{\partial N_j}{\partial z}\right)dV=\frac{\Delta}{\mu}(h_{2i}h_{2j}+h_{4i}h_{4j})$$

$$K_{zzij}=\int_V\frac{1}{\mu}\left(\frac{\partial N_i}{\partial x}\frac{\partial N_j}{\partial x}+\frac{\partial N_i}{\partial y}\frac{\partial N_j}{\partial y}\right)dV=\frac{\Delta}{\mu}(h_{2i}h_{2j}+h_{3i}h_{3j})$$

$$K_{xyij}=-\int_V\frac{1}{\mu}\left(\frac{\partial N_i}{\partial y}\frac{\partial N_j}{\partial x}\right)dV=-\frac{\Delta}{\mu}(h_{3i}h_{2j})$$

$$K_{xzij}=-\int_V\frac{1}{\mu}\left(\frac{\partial N_i}{\partial z}\frac{\partial N_j}{\partial x}\right)dV=-\frac{\Delta}{\mu}(h_{4i}h_{2j})$$

$$K_{yzij}=-\int_V\frac{1}{\mu}\left(\frac{\partial N_i}{\partial z}\frac{\partial N_j}{\partial y}\right)dV=-\frac{\Delta}{\mu}(h_{4i}h_{3j})$$

$$J_{xi}=\int_V J_x N_i dV$$

$$J_{yi}=\int_V J_y N_i dV$$

$$J_{zi}=\int_V J_z N_i dV$$

である. これをすべての要素について重ね合わせることにより,

$$\boldsymbol{KA}=\boldsymbol{J} \tag{5-5}$$

が得られる.

$\boldsymbol{K}$ が正の定符号であることは, 内部エネルギーの記述式(3-2)第1項より明らかである.

次に, 境界条件について考える. (3-1) に表れる積分 $\frac{1}{\mu}\int_\Gamma\langle(\nabla\times\boldsymbol{A})\times\delta\boldsymbol{A},\boldsymbol{n}\rangle d\Gamma$ を,

$$-\int_\Gamma \frac{1}{\mu}\langle(\nabla\times\boldsymbol{A})\times\delta\boldsymbol{A},\boldsymbol{n}\rangle d\Gamma=\int_\Gamma\langle(\boldsymbol{H}\times\boldsymbol{n}),\delta\boldsymbol{A}\rangle d\Gamma$$
$$=\int_\Gamma[(l_zH_y-l_yH_z)\delta A_x+(l_xH_z-l_zH_x)\delta A_y$$
$$+(l_yH_x-l_xH_y)\delta A_z]d\Gamma$$
$$=\int_\Gamma\boldsymbol{\sigma}\cdot\delta\boldsymbol{A}\,d\Gamma$$

とおくと．自然境界条件は，

$$\boldsymbol{\sigma}\cdot\delta\boldsymbol{A}=0 \qquad (5\text{-}6)$$

で表せる．

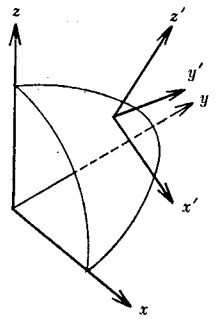


図 1 境界座標系
Fig. 1 Boundary coordinate system

境界座標系 (x', y', z') を図1のように x'-y' を境界面に z' を境界面に垂直に定めれば，$\boldsymbol{\sigma}\cdot\delta\boldsymbol{A}=\boldsymbol{\sigma}'\cdot\delta\boldsymbol{A}'$ は座標変換に対し不要であり，次のように表せる．

$$\boldsymbol{\sigma}'\cdot\delta\boldsymbol{A}'=(l_z'H_y'-l_y'H_z')\delta A_x'+(l_x'H_z'-l_z'H_x')\delta A_y'+(l_y'H_x'-l_x'H_y')\delta A_z' \qquad (5\text{-}7)$$

ここで，$l_x'=l_y'=0$，$l_z'=1$ であるから，

$$\boldsymbol{\sigma}'\cdot\delta\boldsymbol{A}'=H_y'\delta A_x'-H_x'\delta A_y'=\left(\frac{\partial A_x'}{\partial z'}-\frac{\partial A_z'}{\partial x'}\right)\delta A_x'+\left(\frac{\partial A_y'}{\partial z'}-\frac{\partial A_z'}{\partial y'}\right)\delta A_y'=0 \qquad (5\text{-}8)$$

となる．これは，

$$\frac{\partial A_n}{\partial\tau}-\frac{\partial A_\tau}{\partial n}=0 \qquad (5\text{-}9)$$

と表せる．

6. 磁束密度 $\boldsymbol{B}$ の算出

連立方程式(5-4)を解き，ベクトル・ポテンシャル A_x, A_y, A_z を得て，磁束密度 B_x, B_y, B_z に変換する．$\boldsymbol{A}\rightarrow\boldsymbol{B}$ の変換は以下のとおりである．

要素の形状関数は(4-1)であり，(4-2)より，

$$\left.\begin{aligned}A_x&=q_1+q_2x+q_3y+q_4z\\A_y&=q_5+q_6x+q_7y+q_8z\\A_z&=q_9+q_{10}x+q_{11}y+q_{12}z\end{aligned}\right\} \qquad (6\text{-}1)$$

である．また，$\boldsymbol{B}=\mathrm{rot}\,\boldsymbol{A}$ より，

$$\left.\begin{aligned}B_x&=\frac{\partial A_z}{\partial y}-\frac{\partial A_y}{\partial z}=q_{11}-q_8\\B_y&=\frac{\partial A_x}{\partial z}-\frac{\partial A_z}{\partial x}=q_4-q_{10}\\B_z&=\frac{\partial A_y}{\partial x}-\frac{\partial A_x}{\partial y}=q_6-q_3\end{aligned}\right\} \qquad (6\text{-}2)$$

であるから，要素頂点での A_x, A_y, A_z の値，座標値を (6-1) に代入して (4-3) $\boldsymbol{u}^e=\boldsymbol{A}^e=\boldsymbol{C}\boldsymbol{a}$ の変形式 $\boldsymbol{a}=\boldsymbol{C}^{-1}\boldsymbol{A}^e$ より q_1〜q_{12} を計算する．

$$\begin{pmatrix}q_1&q_5&q_9\\q_2&q_6&q_{10}\\q_3&q_7&q_{11}\\q_4&q_8&q_{12}\end{pmatrix}=\begin{pmatrix}1&x_1&y_1&z_1\\1&x_2&y_2&z_2\\1&x_3&y_3&z_3\\1&x_4&y_4&z_4\end{pmatrix}^{-1}\begin{pmatrix}A_{x1}&A_{y1}&A_{z1}\\A_{x2}&A_{y2}&A_{z2}\\A_{x3}&A_{y3}&A_{z3}\\A_{x4}&A_{y4}&A_{z4}\end{pmatrix} \qquad (6\text{-}3)$$

その結果 q_1〜q_{12} を用いて，(6-2)により各要素の $\boldsymbol{B}$ を得る．

7. NASTRAN データの利用

以上の定式化にみるような Maxwell 方程式を NASTRAN が解くというわけにはいかない．NASTRAN に準備されている立体要素 (TETRA, WEDGE, HEXA 1, HEXA 2) に関する入力データをそのままの形で使えるように考慮し，(5-2) によるマトリックス作成ルーチンと(5-4)，(6-2)による $\boldsymbol{A}\rightarrow\boldsymbol{B}$ の変換ルーチンを NASTRAN 内部に組み入れた．

7.1 入力データ

① Executive Control……

```
ID      MAXWEL, NASTRAN
APP     DISP
SOL     1, 0
TIME    5
CEND
```

② Case Control……

OUTPUT	磁束密度 $\boldsymbol{B}$ の出力指定
STRESS=ALL	ベクトル・ポテンシャル $\boldsymbol{A}$ の出力指定
DISP =ALL	
SPC =1	固定境界 $\boldsymbol{A}=\boldsymbol{A}_0$ の指定
LOAD =1	電流密度 $\boldsymbol{J}$ の指定

③ Bulk Data……

座標系データ	直交 CORD1R, CORD2R
	円柱 CORD1C, CORD2C
	球 CORD1S, CORD2S
節点データ	GRID
要素データ	四面体  CTETRA
	三角柱  CWEDGE
	六面体  CHEXA 1 CHEXA 2
要素定数データ	MAT1 (等方性)，MAT2 (異方性)
	透磁率 μ……ヤング率
	電流密度 J……材料密度
境界条件	ベクトル・ポテンシャル $\boldsymbol{A}$ 固定 SPC, SPC1
電流データ	電流密度による指定 GRAV および MAT1，電流 I=断面積 (電流方向)×電流密度となる．
	電流値による指定 FORCE，電流 $I=\dfrac{\text{入力値 (FORCE)}}{\text{長さ } L}$

7.2 磁場解析の出力

節点でのベクトル・ポテンシャル $\boldsymbol{A}$ は節点での変位ベクトルに対応する．次のように変位座標系で指定した任意の座標系で出力する．

DISPLACEMENT VECTOR

P id	T 1	T 2	T 3	R 1	R 2	R 3
1	×××	×××	×××			
2	×××	×××	×××			

磁束密度 $\boldsymbol{B}$ は要素応力に対応する．出力の概略は次(詳細は図5)のようになる．

STRESS IN SOLID ELEMENT

E id	SIGMA XX	SIGMA YY	SIGMA ZZ
1	××××	××××	××××
2	××××	××××	××××

8. 静磁場の解析例

プログラムの精度を確認するために，理論解が知られているケースの解析を試み，解の比較をした．

8.1 ケースA：Z方向の一様磁場

一様な磁場 $\boldsymbol{B}_0$（Z軸を $\boldsymbol{B}_0$ 方向にとる）とベクトル・ポテンシャルの関係は，

$$B_x=\frac{\partial A_z}{\partial y}-\frac{\partial A_y}{\partial z}=0$$

$$B_y=\frac{\partial A_x}{\partial z}-\frac{\partial A_z}{\partial x}=0$$

$$B_z=\frac{\partial A_y}{\partial x}-\frac{\partial A_x}{\partial z}=B_0$$

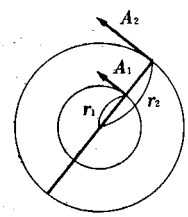


図 2 接線ポテンシャルと一様磁場
Fig. 2 Tangential potential and uniform magnetic field.

可能な解の1つは $\boldsymbol{A}=\boldsymbol{B}\times\boldsymbol{r}/2$ であり，$\boldsymbol{B}\cdot\boldsymbol{r}/2$ の大きさを持ち，Z軸まわりに回っている．

つまり，円周方向に $\boldsymbol{r}$ に比例した大きさのポテンシャルを与えると軸方向に一様な磁場が存在する．

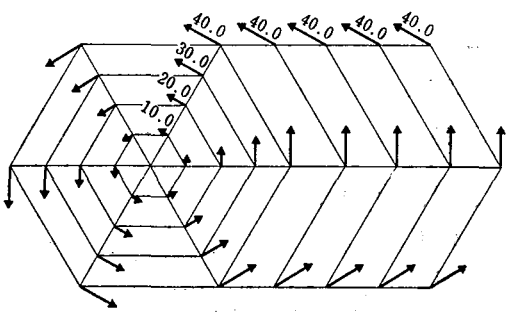


図 3 解析の境界条件——周辺でポテンシャル固定
Fig. 3 Boundary condition for analysis

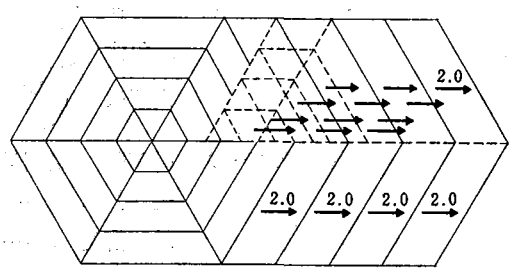


図 4 磁束分布の解析結果
Fig. 4 Solution of magnetic flux

たとえば半径 10.0, 20.0, 30.0, 40.0 で，円周方向に 10.0, 20.0, 30.0, 40.0 なるポテンシャルを与えたとき，$B_0=2.0$ となる．

図3の境界条件のもとで解析し，図4のような解析結果を得た．図5にその結果の数値の一部を示す．

8.2 ケースB：定常電流のつくる磁場

ケースAと同様な円筒領域で円筒中心 (CWEDGE 部分) に Z 軸方向に電流を与えたとき(図6参照)の磁場を解析した．

STRESSES IN SOLID WEDGE ELEMENTS (CWEDGE)

ELEMENT-ID	SIGMA-XX	SIGMA-YY	SIGMA-ZZ	TAU-YZ	TAU-XZ	TAU-XY	OCTAHEDRAL TAU-0	PRESSURE P
1	3.725290-09	3.725290-09	2.000000+00	.0	.0	.0	.0	.0
2	1.862645-09	7.450581-09	2.000000+00	.0	.0	.0	.0	.0
3	-3.725290-09	-2.797180-09	2.000000+00	.0	.0	.0	.0	.0
4	-5.587935-09	-3.725290-09	2.000000+00	.0	.0	.0	.0	.0
5	.0	-3.725290-09	2.000000+00	.0	.0	.0	.0	.0
6	3.725290-09	8.863885-10	2.000000+00	.0	.0	.0	.0	.0
25	-1.862645-09	.0	2.000000+00	.0	.0	.0	.0	.0
26	-1.862645-09	-3.725290-09	2.000000+00	.0	.0	.0	.0	.0
27	.0	-2.531288-09	2.000000+00	.0	.0	.0	.0	.0
28	.0	.0	2.000000+00	.0	.0	.0	.0	.0
29	1.862645-09	.0	2.000000+00	.0	.0	.0	.0	.0
30	.0	5.973250-10	2.000000+00	.0	.0	.0	.0	.0
49	-1.862645-09	.0	2.000000+00	.0	.0	.0	.0	.0
50	-1.862645-09	.0	2.000000+00	.0	.0	.0	.0	.0
51	.0	1.761100-09	2.000000+00	.0	.0	.0	.0	.0
52	.0	.0	2.000000+00	.0	.0	.0	.0	.0
53	.0	.0	2.000000+00	.0	.0	.0	.0	.0
54	.0	3.295455-11	2.000000+00	.0	.0	.0	.0	.0
73	-7.450581-09	-3.725290-09	2.000000+00	.0	.0	.0	.0	.0
74	.0	-3.725290-09	2.000000+00	.0	.0	.0	.0	.0
75	3.725290-09	-5.628449-09	2.000000+00	.0	.0	.0	.0	.0
76	1.862645-09	3.725290-09	2.000000+00	.0	.0	.0	.0	.0
77	1.862645-09	7.450581-09	2.000000+00	.0	.0	.0	.0	.0
78	-3.725290-09	.0	2.000000+00	.0	.0	.0	.0	.0

図 5 結 果 例
Fig. 5 An example of solution

$$\oint \boldsymbol{B}\cdot d\boldsymbol{s}=2\pi rB$$

$$B2\pi r=\mu I$$

理論式として,

$$\therefore\quad B=\frac{\mu I}{2\pi r}.$$

いま $\rho=4.0$, $R=10.0$, $\mu=1.0$ とすると, $I=\pi R^2\rho$ $=1256.64$A となり, 図7のグラフを得る.

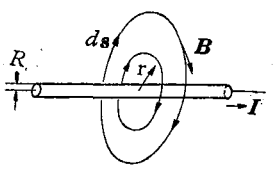
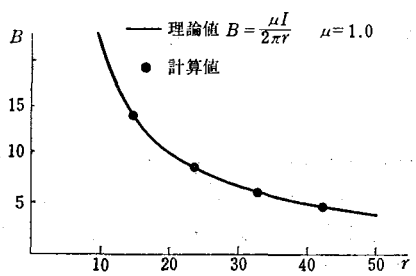


図 6 定常電流のつくる磁場
Fig. 6 Magnetic field by steady state current

一方, 図8の円筒モデルについて解析を試みた.

座標系……CORDIC 円筒
座標値……GRID
要 素……導線部分 CWEDGE
空間部分 CHEXA 2
透磁率 $\mu=1.0$−MAT 1
電流 { 電流密度 $J=4.0$−MAT 1 / 電流方向 −GRAV / 電流 I=断面積×電流密度=1131.37A }
境界条件……円筒外周面でポテンシャル 0-SPC



HEXA 2 要素重心位置での数値比較
(透磁率: 1.0)

半 径	14.607	23.554	32.664	41.830
理論値	13.692	8.490	6.123	4.781
計算値	13.543	8.398	5.975	4.452

図 7 理論値と計算値の比較
Fig. 7 Numerical comparison between theory and calculation

解析結果とその評価をそれぞれ図9, 7に示す.
他に, 円周方向6分割, FORCE による電流指定等を試み同様によい結果が得られた.

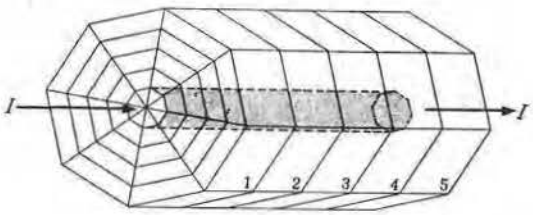


図 8 解析モデル
Fig. 8 Analysis model

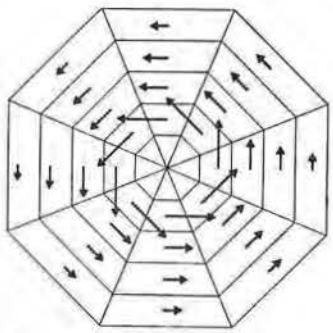


図 9 第3ブロックの磁束分布
Fig. 9 Magnetic flux distribution of the third block

8.3 ケースC：定常電流によるパーマロイ内の磁場

図10に示したようなパーマロイの中心上部に電流があるとき，パーマロイ内部の磁束分布を図11に示す．このとき，zy 面を対称面として 1/2 モデルを解析した．

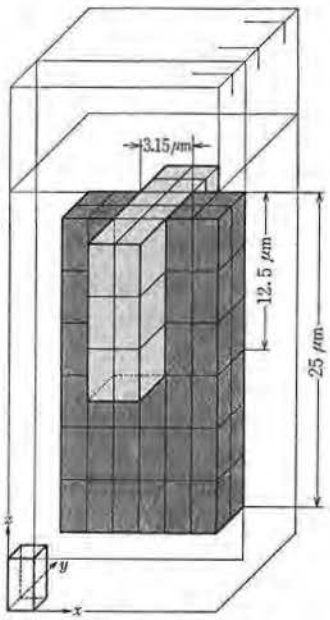


図 10 解析モデル
Fig. 10 Analysis model

図 11 解析結果
Fig. 11 Solution

9. おわりに

3次元磁場解析への有限要素法の適用に必要な定式化を行い．NASTRAN の持つ Solid Element に注目し，磁場解析のための NASTRAN の改良を試みた．基本モデルについては満足しうる解が得られたが，まだ実用には手を加えるべき部分が残されている．

また，磁場解析機能を NASTRAN に登録するに当たり，ご協力いただいた関係各位に感謝します．

参考文献 [1] E. Spreeuw, "Applications of NASTRAN to Nuclear Problems," *NASTRAN USER'S EXPERIENCES*, (NASA Langley Research Center, 1972), pp. 429-441.

[2] E. Spreeuw and R. J. B. Reefman, "Rigid Format Alter Packets for the Analysis of Electromagnetic Field Problem," *NASTRAN USER'S EXPERIENCES*, (NASA Langley Research Center, 1975), pp. 557-570.

[3] 大田充, 伊藤智之, "有限要素法による漏れ磁場の計算(2次元)," 核融合研究, Vol. 25, No. 1, 1970.

執筆者紹介 渡 部 義 維 (Yoshizumi Watanabe)

昭和14年生, 44年早稲田大学大学院理工学研究科(博士)修了. 同年日本ユニバック(株)入社, 現在に至る. 応用ソフトウェア部技術計算グループにおいて, 有限要素法プログラム・サービス等に従事. 物理学会会員.

データベース・マシン

原　　潔

1. データベース・マシンの概要

データベース・マシンとは，データベース・システム専用のアーキテクチャを持つ特別なコンピュータで，データベース・プロセッサやデータベース・コンピュータ等の類義語がある．従来のデータ処理は，70年代を通してデータベース・システムと呼ばれる形態に発展し，データ処理にまつわる各分野が盛んに研究されてきた．しかし，データベース・システムを von Neumann 型コンピュータ上で動作させることを想定すると，データ処理効率の問題や補助記憶と主記憶との間のデータ転送量が過大になりやすいこと，データ処理に並列処理がとりいれられていないこと等の欠点を被むらなければならない．データベース・マシンは，増大する多種多様な負荷を従来の汎用コンピュータ上のソフトウェアにかけるのではなく，データベース・システムの機能，さらにはデータベースそのものを専用のハードウェアに分担させて，情報システム全体の能率向上を図ろうとしている．データベース・マシンは，現在の商用コンピュータから脱却したデータ処理向きのコンピュータ・アーキテクチャの実現を目標としており，その動向は将来のデータ処理の全体像にかかわるだろうといわれている[1]．

2. データベース・マシンの背景

データ処理では，大量のデータを操作するために補助記憶装置が不可欠であり，主記憶と補助記憶間の隘路が大きな問題になってきている．また，データベース・システムの普及や研究を通してデータ独立や集合論的データ操作の導入等，質的な高水準化の要請が強くなってきた．これらに対してデータの関係モデルが提唱され，理論的には１つの解決の道を示した．一般に，このモデルに基づくデータベース・システムを von Neumann 型コンピュータ上に実現するのはむずかしく，その実現のために新しいアーキテクチャ開発の動きが現れてきた．

また，非常に複雑で規模の大きなデータベース・ソフトウェアの信頼性の向上が重要な課題となっている．従来に比べて高水準の外部インタフェースを与えるデータベース・マシンを利用し，ソフトウェアを容易に作成し，信頼性の高いものにしようという要求が出てきた．

データベース・マシンは，データ処理という非数値処理のためのアーキテクチャであるし，データベース言語を対象とする高級言語マシンでもある．また，LSI 新技術の格好の応用の場ともなっている．

データベース・マシンは，分散システムにおけるノードと考えることもでき，広域網分散システムとの関係での研究も進められている．

このような背景がデータベース・マシンの発想を支えている[1]．

3. 代表的なデータベース・マシン

システム全体の中でどのように位置づけられるかという観点から，これまでのデータベース・マシンは，集中論理方式，分散論理方式，広域網方式の３つの方式に分類される（図１）[1]．

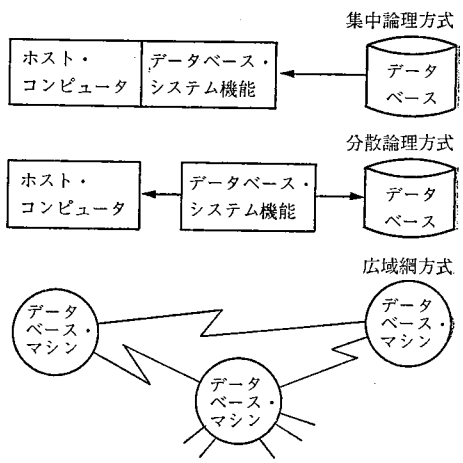


図１　データベース・マシンの概念

これまで提案されてきているデータベース・マシンは，なんらかの形で親となる汎用コンピュータと組み合わせて動作する方式（バックエンド，後置型）をとっている．データベース・システムは，大量のデータ操作を目的とし，補助記憶を利用せざるを得ないので，分散論理方式が研究の主流になっている．

データベース・マシンの研究は70年代初頭より始められ，現在，これらの研究結果が商業ベースで実現されようとしている．この研究を盛んにする１つのきっかけとなったのが Bell 研究所の XDMS である．これはミニコンをデータベース専用の後置コンピュータとして使用するシステムである．この方式では，ミニコンが普通のやり方で磁気ディスクを操

作しているが，これらのミニコンをもっと磁気ディスクに近づけ，ハードウェア的に専用化してホスト・コンピュータからみた補助記憶装置を高水準化（知能ディスク）することが考えられる．そのような例として ICL 社の CAFS がある．以後の方式は補助記憶そのものにデータベース機能を組み込み，なんらかの形で内容呼出しを実現する形態をとっている．固定ヘッド・ディスクのような回転記憶の各ヘッドに論理機構を付与し（論理セル），多数の論理機構に並列の内容呼出し処理を行わせるのが Tront 大学の RAP である．Ohio 大学の DBC は，論理セルと磁気ディスクによる部分的な内容呼出しを組み合わせる構成の大規模データベースのためのデータベース・マシンである．Sperry Univac 社は Ohio 大学の Hsiao 教授と共同研究を行っており，DBC の商品化を図っている．日本では，XDMS の改良案である日電中央研究所の GDS，論理セル方式を拡大して磁気バブル記憶とマイクロ・プロセッサとを組み合せた電子総合研究所の EDC 等がある[1],[2].

4. アーキテクチャ

最近のハードウェア技術，とくに記憶素子技術の著しい発達がデータベース・マシンの実現化の1つの刺激となっている．機械的運動をともなう記憶媒体（磁気ディスク等）に代わる電子的な記憶媒体（固体ディスク）の活用がデータベース・マシン研究開発の主要課題の1つになっている．CCD，磁気バブル記憶装置，電子ビーム記憶装置等がその代表であり，RAP，DBC，EDC がいずれもこの方向を目指している．データベース・マシンのアーキテクチャは多様で，研究者の数だけあるというのが現状である．分散論理方式によってデータベース・マシンが目指している方向に次の2つがある．

第1は，データベースをできるだけデータそのものが存在するところ，補助記憶あるいはその近くで操作することによって，主記憶と補助記憶間のデータ転送量を少なくすることである．実用に供しうる大容量のデータベースを格納し，しかもその上で高度の並列性を有する演算を可能にすることはしばらくの間不可能と思われる．このため，データベース格納に必要な2つの機能，大容量性と演算可能性とを分離して，前者をデータベース・マシン自身の補助記憶で，後者をデータベース・マシンの主記憶で受け持たせ，その間で必要に応じデータを移動させる方法が考えられる．データベース・マシンの主記憶に要求される機能は，高度の並列処理性と少なくともファイル数個を格納できる容量である．

第2は，並列処理を導入して，補助記憶の近いところで内容呼出し記憶の実現を図ることである．データベース・マシンの目的の1つが逐次的なデータ転送の制限から逃れることとすれば，データ転送をいかに並列に効率よく行えるかが，データベース・マシンの成功の大きな鍵を握るといえる．一般に複数のデータベース処理装置による補助記憶の並列処理を行い，補助記憶の水準での比較的低速で，しかも安価で大容量の内容呼出し記憶を実現させようとすることが多い．これは内容呼出し機能には，データ構造や検索条件の変化に強いという特徴があり，関係モデルを作成するためにはうってつけのハードウェアであるという背景による[1,2].

5. おわりに

並列マイクロ・プロセッサが効率を改善し，製造コストを削減するという研究結果を踏まえて，Sperry Univac 社はビットスライスド・マイクロプロセッサをベースとした最初の大規模汎用コンピュータである UNIVAC シリーズ 1100/60 システムを世に出した．マイクロプロセッサ，とくに並列に操作されるマイクロプロセッサの使用は，またデータベース・マシンに適用することができる．Sperry Univac 社は，この領域の研究を79年秋の IEEE で報告した[3]．内容呼出し記憶上でマイクロプロセッサ群によって大容量のデータが並列に転送されるというのがこの設計の鍵であり，DBC のアーキテクチャの拡張になっている．

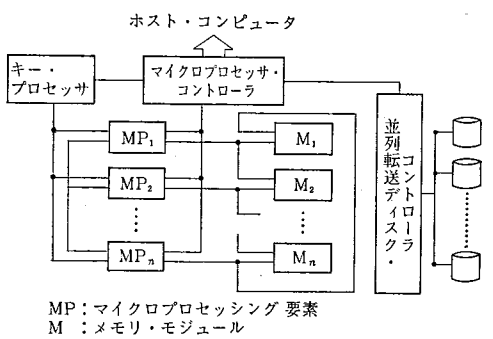


図 2 マイクロプロセッサ・ベースのデータベース・マシン

多くのデータベース・マシンは，関係モデルをベースとして問合せ処理に対して，最も効果が上がるように設計されている．しかし，現在多くのデータベースは網目型であり，言語プロセッサのファイルが大部分を占めている．このような領域でもデータベース・マシンが適用できるためには，データ・モデル間の変換や共通データ・モデルの導入が必要となる．Sperry Univac 社では共通データベース・アーキテクチャという概念を開発している[4].

データベース・マシンの守備範囲をどう考えるか，データベース・マシンの外部インタフェースをどう設定するか，一般のデータベース・システム機能をどう実現するか，等の解決しなければならない問題がまだ山積みされている．これらの問題を解決しつつ，80年代中期にはユーザ・システムの中にデータベース・マシンは定着していくであろう．

参考文献

[1] 植村俊亮，“データベース・マシン”，情報処理叢書 1，情報処理学会，1980.

[2] O. H. Bray, *et al.*, “Data Base Computers,” D. C. Heath and Company, 1979.

[3] H. A. Freeman, “A Microprocessor-Based Design of the Data Base Computer,” *Proc. COMPCOM '79 fall*, Washington, D. C., Sept. 1979, pp. 179–182.

[4] H. R. Johnson and J. A. Larson, “Data Management for Microcomputers,” *Proc. COMPCOM '79 fall*, Washington, D. C., Sept. 1979, pp. 191–194.

(システム・ソフトウェア開発部)

ローカル・ネットワーク

守田洋一

1. 情報通信ネットワークの現状

LSI 革命の到来により，情報処理技術分野と同様に，電気通信技術分野においても，一層の技術革新が予想されている．

通信技術は，アナログ通信からディジタル通信へ進み，高速通信，パケット交換通信，画像通信等が実現している．日本電信電話公社 (NTT) が提供するサービスについても，従来のアナログ方式の通信回線サービスからディジタル通信回線サービスの提供，通信処理機能を付加したパケット交換サービスの提供を行い，VAN (value added network : 付加価値ネットワーク) の分野に広がっている．

諸外国のサービスも同様に，通信回線サービスという基本サービスから，通信処理サービスという高度サービスへと拡大し，高度な公衆通信サービスを行う VAN 事業者が出現してきている．とくに米国においては，この種のサービスに関しても，自由競争理念をとり入れ，民間業者の参入を認めている．また，AT & T (American Telephone and Telegraph) は，米国最大の通信事業者であり，付加価値通信サービスの ACS (Advanced Communication Service) を計画し，FCC (Federal Communication Commission : 連邦通信委員会) に対し認可申請を行ったが，他の通信事業者とのバランスのとれた自由競争に反するとして認可されず，延び延びになっていた．ようやく昨年 (1980年) 末に，FCC は組織を別として運営することを条件に，ACS のサービスを認めることになった．このように，各国ともディジタル通信時代に突入するとともに，反面，付加価値サービスの機能をめぐり，公衆通信事業者の役割と情報通信サービス業者の役割との不明確さが目立ってきている．

一方，日本でも通信政策の大幅な改革が，この 2～3 年以内に起こり，米国と同様，高度サービスについては，規制外の自由競争という考え方を導入するものと予想される．このような状況において公衆網については，将来パケット交換技術をとりいれた付加価値ネットワークが民間も含めて構築されていくものと思う．

また，企業内ネットワークも，先の公衆網を用いた高度サービスを実施する方向であるとともに，自営サブネットワーク化も目立つようになった．コンピュータ・ネットワークは，サブネットワークを中心とした複数ホスト・コンピュータの資源共有利用であり，分散処理に向けて，一層発展していくものと思う．

すなわち，分散処理時代に対応できるネットワークが，情報通信ネットワークとして期待されるものである．

ローカル・ネットワークは，この分散処理時代に出現した，企業内を中心とする局所的なネットワークとして，最近，とくに注目をあびている．

2. ローカル・ネットワークの定義

ローカル・ネットワークの定義は，まだ定まったものはないが，ここでは，次のような定義を試みた．

- 1 組織の所有するネットワーク
- 10 km 以内のネットワーク
- 交換要素技術を用いたネットワーク

すなわち，自営通信設備を用い，かつ新技術の導入が図られた付加価値ネットワークといえる．

3. ローカル・ネットワークの必要性

分散処理システムの構築が実現したことにより，あらゆる地域での情報処理ニーズが高まり，コンピュータの分散配置，インテリジェント・ターミナルの設置，ミニコン，オフコン，パソコン等の配置が相続いで行われている．また，製鉄会社，自動車製造会社，化学・薬品会社等においては，生産管理システムのオンライン・ネットワークを構内設備によって構築する必要から，いち早く (昭和42年頃) ローカル・

ネットワークを構築してきた．データハイウェイ・ネットワークは，この時期に生れたローカル・ネットワークの代表的なものであるといえる．しかし，通信技術の高度化に伴って，伝送速度の高速化，通信方式の高水準化，通信施設（ケーブル等）の高性能化，送受信装置のインテリジェント化等の発展により，ローカル・ネットワークが注目されてきた．また，事務合理化を目的としたオフィス・オートメーションの急速な発展によって，同一ビル，建物内の通信設備の要求が増加し，一層ローカル・ネットワークの構築が必要となってきている．

このような状況下において，近距離のコンピュータ間の通信のための分散処理型ネットワークとして，1977年 Xerox 社が “ETHERNET”（イーサネット）を開発し，発表した．この ETHERNET は，Xerox 社が全面的なネットワーク・デザインを，DEC 社が送受信装置関連を，Intel 社がインタフェースのチップ関連をそれぞれ担当した．その他，Ungermann-Bass 社の “Net/One”，Zilog 社の “Z-NET”等，続々とローカル・ネットワークが出現している．Sperry Univac 社においても，“Shinpads” と呼ぶネットワークを開発中である（表1参照）．

4. ローカル・ネットワークの構成要素

通信施設，送受信装置，インタフェース装置，制御装置，の4つが，一般的構成要素である．

4.1 通信施設

ローカル・ネットワークで用いられる通信媒体として，マイクロウエーブ (line of sight microwave)，光ファイバ・ケーブル (fiber optic cable)，同軸ケーブル (coaxial cable-CATV)，電話線 (twisted-pair cable) 等がある．先の ETHERNET, Net/One 等の主要なローカル・ネットワークは，CATV 用のタップとコネクターを持つ低損失の同軸ケーブルを用いている．

同軸ケーブルは，次の長所を持つ．

- 高速通信（およそ 10 M ビット/秒）が可能である．
- ノード（送受信装置）間の距離が長くとれる．
- CATV の技術が利用できる．
- 比較的安価である．
- 比較的雑音（ノイズ）に強い．

反面，短所としては，セキュリティ，ケーブルの太さや重量等に問題がある．

次に，今後の主力設備として用いられる光ファイバ・ケーブルについて，その長所をあげてみる．

- 高速通信が可能（100 M ビット/秒）である．
- 雑音に非常に強い．
- 伝送損失が低い．
- 物理的に細く軽量である．

一方，短所としては，統一化への方向，現在での費用，標準化への欠如等の課題がある．

4.2 送受信装置

同軸ケーブルに接続するためには，受動的なベース・バンド・トランシーバと非破壊的なタップを経由する．主機能は同軸ケーブル内を数 M ビット/秒で送受信する（ETHERNET では 3 M ビット/秒）ものの他に，送信データ，受信データ，衝突の検出，電力の供給制御等がある．

4.3 インタフェース装置

ETHERNET のインタフェース装置は，処理装置で使われる並列データと通信路上の直列データとの間の直並列変換を行う．また，送受信される直列データに対する16ビットの CRC（誤りチェック）回路を持っている．すなわちリンク・レベルのプロトコル制御を行っている．

4.4 制御装置

Net/One 等では制御装置をネットワーク・ノード（プロセッサ）と呼んでいる．この装置の主な機能は，処理装置（ユーザ・データ）からきたデータをパケット化し，送受信制御とコード変換，フォーマット交換等のネットワーク・レベルとそれ以上の高位レベルのプロトコル制御を行う．

ETHERNET では，トラフィック制御，衝突管理等を行っている．

5. ネットワーク・トポロジー

ローカル・ネットワークを設計するに当たって，最初に決定すべきことは，ネットワーク構成のトポロジーであり，スター型，リング/ループ型，データバス/マルチドロップ型，メッシュ型のどれを選ぶかである．以下，そのポイントについて記述する．

1) スター・トポロジー：すべての通信が管理できる中央ノードと全利用者とを接続する方法で，中央装置の限界と信頼性，ケーブルが長くなるための経費高等の問題点がある．
2) リング・トポロジー：アーキテクチャが単純で，そのためプロセッサと他のネットワーク要素を接続する費用が少ない点，中央の装置によるボトルネックが解消できる点等の長所を持つ．一方，ネットワーク全体の信頼性を各々の接続に依存する点，メッセージが一方向にだけ回っているので，リングのいずれかの地点の故障が全体に影響する等の欠点がある．そのため，各端末とネットワークを接続するハードウェアは，その個所のノードが故障した場合にも

表 1　ローカル・ネットワークの各社の状況 ("3 Com Local Computer Network Vender List" より)

会社名	製品名	設備	ステーション	速度 [ビット/秒]	距離
Data General	MCA	マルチコンダクタ・ケーブル	15 CPU	約 1 M	150 ft
Data Point	ARC system	同軸ケーブル	255	2.5 M	4 mil
DCC	CAP	同軸ケーブル	100	1 M	CATV
GTE/Telenet	TP 4000 S	バーレッド (RS-232C)	256	56 k	無制限
Hewlett-Packard	Fiber-Optic HB-1B Link	optical fiber	2 HP-1 Bs	160 k	100 m
ITT	Optical-Fiber Digital Modules	optical fiber	2	5 M	1.5 km
Network Resources Corp.	LOCALNET	Broadband 同軸ケーブル	24,000	9.6 M	40 mil
Network Systems Corp.	HYPER Channel	同軸ケーブル	256	50 M	最大 3,000 ft
Network Systems Corp.	HYPER Bus	同軸ケーブル optical fiber	—	6 M	—
Prime	Primnet Node Controller	同軸ケーブル	200	8 M	750 ft (ノード間)
Sperry Univac	Shinpads	triaxial cable	256	8 M	300 m
Ungermann-Bass	Net/One	同軸ケーブル	250	4 M	4,000 ft
Valmet	Dataway	同軸ケーブル	120	250 k	2,000 m
Xerox	ETHERNET	ベースバンド, 同軸ケーブル	—	10 M	500 m
Zilog	Z-Net	同軸ケーブル	255	800 k	2,000 m

ネットワーク機能を維持しなければならないため，複雑なものとなる．

3) バス・トポロジー(方式)：単純さという点では，リング型とほぼ同じである．この方式では，受動的なコネクションを採用することによって信頼性が失われるのを防いでおり，さらにバスへのアクセスに使用する伝送ハードウェアは，より単純である．

6. おわりに

ローカル・ネットワークの基礎的な事項について触れたが，今後，プロトコルの標準化も進められ，異社，異機種間接続等も容易に実現されていくものと期待される．一方，通信媒体も同軸ケーブルに替わり光ファイバ・ケーブルも利用されるであろう．

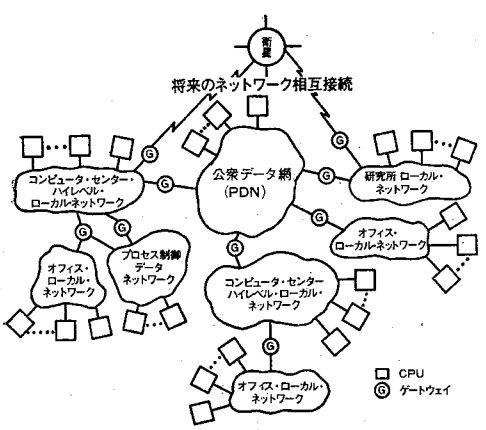


図 1　接続構成の将来

また，適用システムについても音声メール，ワードプロセッサ，画像システム等の広帯域アプリケーションにも適用され，オフィス・オートメーションに革命をもたらすものと思う．

(コミュニケーション・システム部)

プロトコルの形式的記述技法

鈴木　直

1. 仕様としてのプロトコル

プロトコルは，その性格上，コンピュータのアーキテクチャ，システム・コンセプト，用語等が異なるシステム間で共通に適用される．ところが，その仕様に関する表記法，前提知識（仕様書に記述されない仕様）等についての世界的な共通基盤がいまだに確立されていない．しかし，データ・コミュニケーションの発達にともない，最近ではハードウェア，ソフトウェア等の個々の製品の機能仕様とは独立に，プロトコルのみを純粋に記述することの必要性が一般に認められるようになってきた．ISO では1980年1月より，CCITT では1981年4月より，それぞれプロトコルの形式的記述技法 (Formal Description Technique : FDT) に関する検討を行っている．

2. プロトコル・マシン

プロトコルは“通信等の協調動作を行う複数の何らかの‘物’(エンティティ：entity) の間の約束ごと”と説明される．その仕様の記述は，エンティティ間の情報交換を矛盾なく実行できるよう，

・交換される情報の種類と形式

・情報交換にかかわる内部処理と動作

の2つにわけて行う．とくに，協調動作を行うエンティティ間では，処理手順や動作はデータの内容のみから一意に規定できない．これらは，自分および相手エンティティのその時々の状態，および非同期的に発生する要因とによって左右される．したがって，この制御構造を仕様として明確に規定する必要がある．

このために，各エンティティの状態（の集合）は互いに既知で，その要素の個数は有限であり，状態の遷移と遷移の要因（入力通知や出力指示等）についても既知であるような系を想定する．この系において，各エンティティを1種の機械，すなわち

$M=(S, F, P, \delta, \omega)$

S：状態集合，F：要因集合，P：動作集合

δ：状態遷移関数，ω：動作決定関数

として抽象化し，さらに $s\in S$, $f\in F$, $p\in P$ で

$$s^{i+1}=\delta(s^i, f^i),\quad p^{i+1}=\omega(s^i, f^i)$$

とする．すなわち，エンティティの状態遷移や動作は要因とその要因の発生した時点の状態によってのみ決定され，それ以前の状態には左右されないもの（あるいはそのように状態を分割する）と仮定する．一般に，プロトコルの制御構造はこの仮定に基づき表現される．このような仮想の機械をプロトコル・マシンという．

3. 仕様検証と合成プロトコル・マシン

一般にプロトコルの設計段階では，定められたプロトコル自身に誤りがないか否かを検証する必要がある．誤りとは，たとえば協調動作を行う複数のエンティティ全体を1つの系とみなしたとき，動作上のデッドロックやループ状態に陥ることを意味する．この種の検証すなわち仕様検証のためには，影響を及ぼし合う個々のプロトコル・マシンを合成して，1個のプロトコル・マシンとして扱う必要がある．2個のエンティティにより構成される系の場合を考えてみよう．合成プロトコル・マシンの状態数は個々のエンティティの状態数の積となり，通信路（入出力チャネル，通信回線，共通ストレージ等）に誤りがないと仮定すれば，一方の出力動作を他方の入力要因として結合することができる．この仮定には実用上無理がある（5章参照）．しかしながら，出力された情報が通信路上で完全に紛失する場合を考慮した合成プロトコル・マシンの研究も行われている．

3個以上のエンティティからなる系については，P. M. Merlin がトポロジーの概念の導入を試みたようであるが，彼の急死（1979年7月）以後，この議論の進展についての情報は得ていない．

4. 代表的なプロトコル記述技法

プロトコルの制御構造を表現するための多くの形式的記述技法（FDT）が提案されている．それらはおよそ以下の6つに大別される．

4.1 ペトリ・ネット

制御と情報の移動を，一種の有向グラフ上のトークンの移動で表現する（図1）．

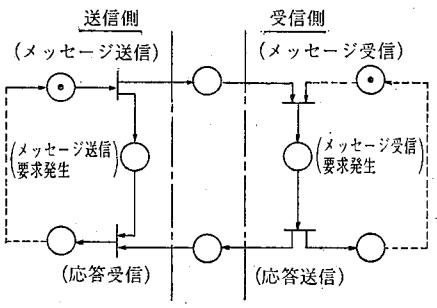


図1 ペトリ・ネットの記述例（メッセージ送受信）

時間や論理演算，論理判断等の概念を導入した種種の拡張技法も提案されている．

4.2 状態遷移図

有向グラフの1個のノード s_i を1個の状態に対応させ，個々の要因に対応するアーク f_i が状態間の遷移を表現する（図2）．

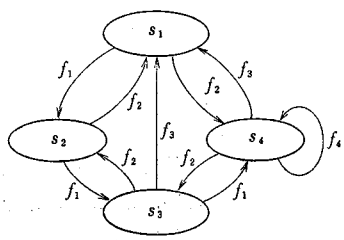


図2 状態遷移図の例

4.3 状態遷移表

状態遷移図を表形式にしたものであり，行と列をそれぞれ，要因（f_i）と状態（s_i）に対応させて，個々の行と列の交点で次の遷移状態（s_p）と処理および動作（p_l）を示す（図3）．

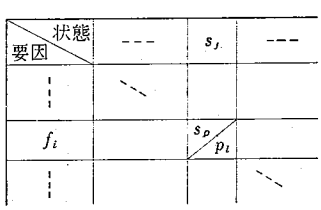


図3 状態遷移表の形式

4.4 経路行列

P. Zafiropulo が提案したもので，基準となる状態から出発して，状態の遷移をたどり，元の状態にもどる1個の経路を1個の行列で表す．このようにしてプロトコルをすべての異なる経路に対応する行列の組で表現する．

4.5 プログラム言語

G. V. Bochman や S. Schindler 等が行っている

技法で，非同期的に発生する要因の処理順序に一定の制約を課し，処理そのものを定められた構文で表現する．見かけは異なるがフローチャート風に図形記号を用いて見やすくしたものもある．ただし，合成プロトコル・マシンの表現には用いられない．

4.6 その他の技法

プロトコルの一般的な性質を取り扱うための通信路上の事象の系列を，代数的に表現する方法（T. Kimura）がある．また形式的技法ではないが，自然言語による方法，局所的なインタラクションの表現に便利なタイミングチャート（図4）もよく使われている．

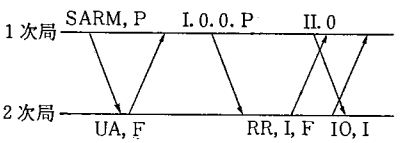


図 4 タイミングチャートの例

5. 実用的な記述技法

現実には，誤りのない通信路を期待することはできず，一方誤りの発生の仕方について，何らかの仮定を置くことも許されない．したがって，実際に扱うプロトコルのモデルとして合成プロトコル・マシンを用いることはなく，体系的な仕様検証は行われていないのが実情である．ただしデッドロックやループに関しては，種々のタイマやカウンタを設けて，個々のエンティティの動作を自分自身で監視するようなプロトコルとしている．

また，実際に扱うプロトコルの複雑なものでは状態数が60を超え，要因数もまた60に近いものもある．現実に60個のノードを持ち1,000本を超えるアーク（状態により無視されたり，状態遷移を引き起こさない要因がある）が入り乱れているようなグラフは，プロトコルの設計，保守，インプリメンテーションまた製品検証のいずれの段階に対しても有効でありえない．また，プロトコルの制御構造の表現に適したプログラム言語仕様についても，まだ試行錯誤の段階である．このため，現在状態遷移表が最も一般に用いられている．プロトコルのその他の側面，たとえば情報の種類や形式については日本ユニバック（株）をはじめ一般に図，表および自然言語で規定しているが，Sperry Univac 社では文法記述用ツール Backus-Naur Form (BNF 記法) を用いて規定している．

6. 今後の動向

エンドユーザが直接期待するものは，よいプロトコルではなく，良質なサービスである．プロトコルとは，そのサービスを実現するために用意されるべき方法であり，目的ではない．提供するサービスの内容が複雑になるに従って，一般の認識も変化し，エンティティの個々の動作そのものの規定の他に，動作やメッセージが持つ意味や意図を正確に定義する必要性が認識されつつある．この流れに沿ってサービスの形式的記述も試みられるようになった．

曖昧さのない (unambiguous)，完全 (complete) な仕様は，ハードウェア，ファームウェア，オペレーティング・システム等との機能分担を限定することによって，プログラムの自動作成を可能にする．プロトコルの形式的記述技法を基にしたオートメーションについても，サービスの正確な記述，他との機能分担の明確化等と相まって着実に進歩すると思われる．

すでに，プロトコル仕様の記述に関する論文は，数多くある．1980年1月に開催された ISO/TC 97/SC 16/WG 1 の FDT 関連国際会議資料として提出された J. Day と C. Sunshine の “Revised Bibliography on the Formal Specification and Verification of Computer Network Protocols”（旧版は 1978 年）には約 160 件の論文が列挙されている．このうち，約130件は1976年以降の4年間に表されたものである．この種の技法が，世界的な合意のもとに定着するまでには，まだかなりの時間を必要とするであろう．

（コミニュケーション・システム部）

RONALD P. UHLING 他著

"The Office of the Future"
ICCC Monograph Series: Vol. 1

North-Holland, 15.3×23.0cm, xiv+380 pp., 1979.

ICCC (International Council for Computer Communications) は国際会議の開催や会報の発行だけでなく，各種出版活動を通じてこの分野の知識の普及に努めている．その1つに，今日的テーマを選び，掘り下げた叢書の刊行がある．

本書はその叢書 (Monograph Series of the International Council for Computer Communications) の第1巻目で，Ronald P. Uhling, David J. Farber および James H. Bair の3名による共著である．

Carl Hammer 博士は，序において次のように述べている．

「人間の知的能力を広げる科学技術の歴史は始まったばかりで，その将来を予見するのは容易ではない．しかし，コンピュータおよびコミュニケーションの発達は勢いの赴くところオフィスの大々的な変革とならざるをえない．現在のところ，オフィスの将来像は薄明かりの中にぼんやりとしか見えない．オフィスは何のためにあり，そこに働く者にとって意義ある仕事とは何か．」

本書はこのような視点に立ち，技術面に深入りしすぎることなく，まとめられたものである．したがって，事務系・技術系を問わず，オフィスの変革に興味を持つ企業人にとり，問題の諸様相を一通り知るために，一読の価値があるものと思われる．

本書の構成は大きく3部に分かれていて，

PART I : Use of Computers in the Office of the Future

PART II : Technological Imperatives

PART III : The Impact of Office Automation

からなる．各部分の内容を見てみよう．

情報の量，とくにその流通量が急速に増大している．オフィスは，この情報が交錯し集中する1つの重要な焦点である．そしてそこで働く者は，各種のツールを縦横に使いこなし，情報の受渡し・分析・変更・作成などを行うこととなる．PART I では，このような未来のオフィスの様相と各種のツール，すなわち文書作成用ツールを記述している．さらには，会話型のオフィス支援システムとその音声との融和についても言及している．

未来のオフィスの情報処理能力が大きく向上するためには，それを可能とするだけの技術基盤の充実が必須である．PART II は，ディジタル通信をはじめとする通信技術の革命的進展，ソフトウェアに関する考察，通信技術の上にうちたてられる分散処理，分散データベース等につき，例を織りまぜて説明している．

技術の進歩につれ，オフィス風景は一変するはずである．しかし，機器をはじめとする無機的物体だけがオフィスにあるわけではなく，主体は人間とその組織であることはいうまでもない．したがってオフィスの進化は常に人間およびその組織との共存，整合性の確保を念頭に置いて図られねばならない．このような文脈から，PART III では

①オフィス・オートメーション実現のための方策
②オフィス・オートメーションの影響の評価方法
③影響のうち既知のもの
④組織の生産性への影響

について，8章にわたって述べている．

坂井 秀寿 著

"日本語の文法と論理"

勁草書房，A5判，vi+266 pp., 1979年，3,500円．

本書は，Richard Montague の semiotic program に沿って，日本語を論理的に考察し，日本語のごく小さい断片を用いて，自然言語である日本語の厳密な semiotic study の基本方針を述べたわが国で最初の成書である．ちなみに，序文冒頭に，「本書の狙いは，アメリカの論理学者 R. Montague が開発した文法理論を日本語にも適用し，それを通じて日本文法の論理構造を解明する」ことにあると述べている．

ところで，R. Montague は自然言語の semiotic study も論理のような"formal"な言語の semiotic と同様に数学的に厳密に行いうるという thesis に立ち，1966年から死の前年1970年にかけて，自然言語の syntax, semantics, および pragmatics の研究を厳密に論理的に行うためのフレームワークを高階の内包論理を用いて開発した．これが Montague

の semiotic program であり，開発されたフレームワークは，Montague の文法理論とも，Montague 文法ともいわれる．

R. Montague の semiotic program の全貌は，彼の選集である

"Formal Philosophy—Selected Papers of Richard Montague", Edited and with an introduction by Richmond H. Thomason, Yale University Press, 1974, 369 pp.

の3章から8章に収められた6編の論文：

3. Pragmatics
4. Pragmatics and Intensional Logic
5. On the Nature of Certain Philosophical Entities
6. English as a Formal Language
7. Universal Grammer
8. The Proper Treatment of Quantification in Ordinary English

に示されている．6章の English as a Formal Language で R. Montague は彼の thesis を述べ，7章の Universal Grammer では，syntax と semantics 記述の一般理論を開発している．また，R. Montague は8章 The Proper Treatment of Quantification in Ordinary English において小さい英語の断片を用いて高階内包論理への翻訳を行い，Montague のフレームワークによる自然言語の semiotic study の基本指針を示している．

なお，英語の断片については，この Montague の thesis を継承する M. J. Cresswell の著書の解説つき翻訳である；

M. J. Cresswell 著，石本新・池谷彰訳，言語と論理，紀伊国屋書店，1978, pp. 353

がある．

本書は，この Montague の thesis に沿って Montague のフレームワークを用い，日本語の semiotic study をきわめてていねいに試みた著書である．著者が序文に述べる意図を引用すると，「私は読者に論理学の知識まして Montague の文法理論の知識を一切要請しなかったかわり，それらの紹介をも併せ行わざるをえなかった．決して私の本意ではなかったが，本書は高階の述語論理，高階の様相述語論理，そして Montague 文法理論への入門書としても奉仕している」と．はじめに述べたように，わが国での最初の成書としての本書の位置と，Montague の semiotic program の有効性を考えるならば，「決して著者の本意ではない」本書の有用性は歴然としている．この分野に関心のあるすべての人に本書を勧めたい．

さて，本書の前半の 1, 2, 3 章は，ハイライトをなす後半の4～7章のためのていねいな序章となっている．前半ではいわゆる orthodox な論理である extensional な higherorder predicate Logic L_P とその解釈を導入し，日本語の断片 J の L_P への翻訳を行う．本書の後半の 4, 5, 6, 7 章では，Montague の thesis の説明につづいて高階内包論理 L_I が導入され，possible worlds による S. Kripke の解釈が与えられる．

そして，まず断片 J の L_I への翻訳が実行される．日本語の断片 J を拡張した J′ を考えて，次に J′ の L_I への翻訳を論ずる．終章の7章では「前途瞥見」と題して L_I の L_{IP} への拡張やその他の残された今後の研究課題とその方向を概論している．章立てを示すと，

1. 高階述語論理を骨子とする人工言語 L_P
2. 日本語の断片 J と，その「翻訳」が意味するもの
3. J の L_P への翻訳
4. L_P の拡張：内包論理を骨子とする言語 L_I
5. 日本語の断片 J′ J の統語規則の再構成
6. J′ の新たな統語規則と，その L_I への翻訳
7. 前途瞥見

わが国の情報処理において，自然言語としての日本語を使用する情報処理が実用上重要であることはいうまでもない．同時に，それは魅力的な研究開発のテーマでもある．

周知のように，日本語 Query による情報検索，機械翻訳，日本語理解システムや対話システム，Q/A システム，あるいは日本語による仕様記述やプログラム合成等の多様な応用人工知能ないしは知識工学の応用システムの研究開発が進められている．情報処理の世界では，機械処理ができるような論理的枠組によって日本語——それが domain specific であり，また controlled language であろうとも，——応用領域で求められる日本語の論理構造を解明することは，これらの日本語情報処理に不可欠な基礎的ステップであろう．

幸いに，近年，Montague の文法理論への関心も高まってきている．この意味で，本書のはたすべき役割は，著者の本意とかかわりなくきわめて大きい．

CALENDAR

●第11回画像工学コンファレンス——東京，1980年12月4日～5日

延べ750名の参加を得て，8編の招待講演と15件のポスタ・セッション，26件の一般講演が行われた．招待講演では，穂坂衛教授（東大）の「コンピュータ・グラフィックスと幾何モデル」，深浪良治氏（横須賀通研）の「画像通信技術の国際標準化」等が発表された．このほか，ポスタ・セッションではホログラム画像メモリ，光学的座標計測方式とその応用，曲面物体の形状計測と記述，ステレオ像の自動解析，地形図の色分離ファイル化法，魚眼レンズ画像のディジタル処理，ミニコン・ベースによる顔の特徴抽出，流れの可視化の計測処理等が話題となった．

●26 th IEDM (International Electron Devices Meeting)——Washington D.C. 1980年12月8日～10日

総数200でを越える論文（1/3が米国外のもの）が発表された．今回の話題は，サブミクロンの線幅を可能にするリソグラフィとマルチプル・セルフ・アライメント技術，三重レベル多重シリコンEE (Electrically Erasable)—PROM (Mostek)，多結晶シリコンによるバイポーラ素子および3次元素子，非晶質シリコンによる集積化インバータ，GaAs半導体によるバイポーラ素子と絶縁ゲートFETカウンタ，バルク・シリコンあるいはサファイア上のCMOS，バイポーラ技術と接合形FETとI^2Lの混用，シリコン・マイクロトランスデューサやマイクロ・メカニカル加速度計等のセンサー，フォト(light-activated)・サイリスタ等であった．

●ソフトウェア・シンポジウム '80——東京，1980年12月10日～11日

ソフトウェア産業振興協会の主催および同協会の会員の発表により実施された．招待講演は，「ソフトウェア・メトリックス：高信頼性ソフトウェアへの新しいアプローチ」と題してT. Gilb氏により行われた．また，一般講演はツール，開発支援システム，テスト，生産性と管理，標準化，保守，設計，方法論の各部門に分かれ，29編の論文が発表された．

●第22回プログラミング・シンポジウム——箱根，1981年1月12日～14日

招待講演は，田辺和夫氏(トヨタ自工)により「トヨタ自工におけるボデー開発工程のCAD-CAM」と題して行われた．また，一般講演の主なテーマは，自動的にハイフネーションする英文清書システム，日本語文書処理用入力設計と国語辞典の分析，マイクロコンピュータによる漢字処理システム，教育用FORTRANコンパイラの試作，TAO LISP開発，SNOBOL 4の制御構造を持つLISP，パケット・プログラミング，ポータブルLISPエディタPLET，LISPプログラミング・ツールHLISP，目的コード生成過程記述用ツール，会話型ソフトウェア開発支援ツールSOFT，ハッシング技法における探索効率の改良等である．

●1981 ACM Computer Science Conference——St. Louis, 1981年2月24日～26日

招待講演は，P. Wengner教授 (Brown大学)，B. W. Boehm氏 (TRW)，H. D. Mills氏 (IBM)により，それぞれ「ADAの概要」，「ソフトウェア工学における研究課題」，「ソフトウェア開発における例外的な効率」と題して行われた．また，今回のForsythe記念講演はG. G. Dahlquist教授 (Royal Institute of Technology, Stockholm)により行われた．同講演はACMのSIGNUMにより1972年に制定されたもので，数値解析に顕著な貢献のあった人に与えられる名誉である．Dahlquist教授は「常微分方程式の数値解法の安定性」の研究における功績により同講演者に選ばれた．

●COMPCON '81 Spring——San Francisco, 1981年2月23日～26日

基調テーマは，「研究所・オフィス・工場・家庭におけるVLSI」．基調講演と特別講演は，C. Mead教授 (California大学) とH. A. Simon教授 (Carnegie-Mellon大学) により，それぞれVLSI技術のコンピュータ・アーキテクチャへの影響，オートメーションの社会に及ぼす影響について述べた．また，パネル討論は，VLSI技術の将来，UNIXのマイクロコンピュータへの移植，可処分インテリジェンスとしての手持用コンピュータ (hand-held computer)，ソフトウェアのマスマーケティング，ソフトウェアの所有権・価値・法的義務等であった．

●第5回国際ソフトウェア工学会議——San Diego, 1981年3月9日～12日

基調講演は，PASCALの生みの親であるN. Wirth氏による「PASCALのつぎに何が登場するか」と，泡箱の発明者として知られるNovel物理学賞受賞者のD. Glaser氏による「自然から学ぶ設計の教訓」等．また，パネル討論のテーマは，ソフトウェア工学手法の実用性，ソフトウェア管理

の成功および失敗例，ソフトウェア品質尺度の有効性，ミニ/マクロ・コンピュータにおけるソフトウェア工学等が取りあげられた．

●昭和56年度 OR 学会春季研究発表会——大阪，1981年3月12日～13日

物流問題を特別テーマとして開かれ，約110編の論文が発表された．特別講演は，永山幸次氏（久保田鉄工）による「久保田鉄工におけるロジスティックス・システム」と永峯昭氏（ニチイ）による「量販店の物流について」であった．特別セッション「物流とOR課題」では，大学関係者から，ストックレス生産と最適化概念，物流と情報システム，物流と計画技法，また民間企業からは，物流マネジメントとOR，倉庫業の変遷と80年代の志向，販売と物流の現状と課題が報告されている．

●第22回情報処理学会開催——東京，1981年3月24日～26日

特別講演および招待講演は，林雄二郎氏（未来工学研究所）の「社会の成熟化と情報化社会」と菅野卓雄教授（東大）の「Josephson 効果とその応用素子」であった，また，2つのパネル討論が，「ソフトウェア工学80年代の課題」（司会＝大野豊教授，東大），「オフィス・オートメーションの課題」（司会＝三浦大亮氏，東レ）と題して行われた．一般講演では，548件の研究論文が，言語，エディタ，性能評価，OS，ソフトウェア工学，言語処理系，画像処理，ネットワーク・アーキテクチャ，ネットワーク・プロトコル，数値計算，基礎理論，情報処理教育，教育システム，回路解析シミュレーション，CAD と論理設計，社会公共システム，対話型編集システム，自由曲面とアニメーション，設計・予測・支援，マイクロコンピュータ，交通システム，日本語文書処理，日本語入出力装置，文字認識，オフィス・システム，プロセッサ・アーキテクチャ，データフロー・マシン，端末・入出力システム，ファームウェアとリスト処理，マルチプロセッサ，検査・信頼性，論理回路・LSI 設計，データベース，情報検索，自然言語，日本語処理・辞書，音声，人工知能，知識工学・知識ベース等の各セッションに分かれて発表された．

① 基礎理論……履歴データを持つデータベースの内包論理による意味論プログラム合成法，プロセス・ネットワーク・モデルに基づくプログラムの合成，複数グループ間の暗号鍵共有法と保護強度解析，GF(2) における基本演算の複雑性，高次代数方程式の判別式の数式処理による計算等．

② 言語と言語処理系……述語型言語 DURAL，アルゴリズム記述用言語機能，PROLOG/F とその処理系，ベクトル演算高速化率の測定，PASCAL の heap 領域の拡張，BCPL によるオウン・コーディング，入力されたプログラムを知識として活用するプログラミング・システム，ALGOL 68 の3パス・コンパイラ，YACC をモデルとしたコンパイラ・コンパイラ CC 1，データフロー解析技術を用いたポインタ・エラー検出等．

③ OS ……データフロー・マシン TOPSTAR-Ⅱにおける LISP と PROLOG，データフロー・マシン用言語 Valid，分散モニタ機構 DIALOG-M 等．

④ ソフトウェア工学……大規模 OS の品質管理手法，階層的仕様記述システム HISP，LISP エディタによる PASCAL プログラムの編集，複数画面を用いるエディタ，未テスト・パスの実行条件自動抽出によるテスト・データ作成，プログラム修正時の影響範囲の自動抽出，メンテナンス・サポート・システム，保守用ドキュメントの機械作成等．

⑤ 画像処理……ディジタル画像の選択符号化，構文解析の法を用いた部分図形の検出，連続画像の処理，画像の構造化蓄積と特徴パラメータによる検索，眼底地図の作成，文書画入の再成法，天気図認識プログラム WERP，画像解釈言語 PIL-1，アニメーションのための情景記述，アニメーション記述言語 IMAGE-2 等．

⑥ ネットワーク関係……無線回線による計算機網，スプール制御による計算機通信システム，網向きプロセス間通信プロセッサのソフトウェアとハードウェア，分散処理システム記述言語，通信プロトコルの記述法と設計法，待ち行列網シミュレータ HASSQN，機能分散型計算機におけるシステム管理サブシステム等．

⑦ 日本語文書処理とオフィス・システム関係……オンライン手書き文字認識システム JOLIS-O，言語 CLU を用いたマクロ方式の仮名漢字変換，漢字情報処理における文字セット管理システム等．

⑧ コンピュータ・アーキテクチャ……連想処理機能を持つ MSIMD システム，連想プロセッサ DREAM-Ⅱ，並列マルチ画像処理計算機 PIPE，推論マシン・アーキテクチャ，関数型データフロー・マシン，データフロー・マシン TOPSTAR-Ⅱ，データフロー・プロセッサ・アレイ，抽象データ構造のファームウェア化，標準 LISP 用マシ

ン，計算機複合体 POPS におけるメモリ競合等.

⑨ データベース……データベースの代数的意味論，ファジーな性質を持つデータベースにおける整合性，高速集合演算用のデータ構造，分配分割法による関係演算の高速化，関係データベース専用 OS，個人用 DBMS "Micro-Period"，マルチ・データ・モデル・アーキテクチャ DBMS，アルゴリズム情報の知識構造，分散型データベースにおける異種 DBMS の質問プロセッサ，質問作成・改良を支援する利用者インタフェース，関係モデル・データベースの問合せ用言語，サーチ・エンジンとソート・エンジン，可変構造多重処理データベース・マシン等.

なお，日本ユニバック(株)からは，田中康仁が「専門用語の抽出」と題して発表を行った.

●昭和56年度電子通信学会総合全国大会——東京，1981年4月1日～3日

特別講演は，森政弘教授（東工大）による「やわらかい頭」と井深大氏（ソニー）による「教育について」であった．シンポジウムでは，拡散スペクトル通信方式における諸問題，仕様記述・設計検証の理論と実際，電子計算機網と分散処理，LSI プロセス・シミュレーション，センサ・デバイスとその応用，高密度磁気ディスク技術，遮蔽の理論と実際，ギガビット回路の現状と将来，生体情報システム，ディジタル動画像処理とその応用，認知科学，可視光半導体レーザとその応用，エレクトロ・メカニックスの諸問題，衛星の姿勢制御と軌道制御等が討議された.

●NCC '81——Chicago, 1981年5月4日～7日

基調テーマは，「生産性向上への鍵 (Keys to Productivity)」であり，イタリアの Italtel SA の M. Bellisario 氏が講演を行った．同氏は NCC 史上初の女性基調講演者となった．本大会では，110 を越えるテクニカル・セッションが開かれた．その主なテーマは，ハードウェアおよびアーキテクチャ関連では，高水準マイクロプログラミング言語とその最適化，適応型アーキテクチャ，フォルト・トレラント・コンピューティング，アレイ・プロセッサの応用等であった．ネットワーク技術とコンピュータ能力計画の関連では，開放システム・インターコネクションの ISO 参照モデル，ローカル・ネットワーク，能力計画 (capacity planning) 等がとりあげられた.

① ソフトウェア分野……プログラム品質の定量的尺度，ソフトウェア信頼性，ソフトウェア開発用ツールおよび設備 (facility)，辞書システム，ソフトウェア保守，汎用ユーザ・インタフェース，PASCAL の標準化と機能拡張，Smalltalk-80 等.

② 情報処理システム管理の分野……80年代のソフトウェア生産工程，ユーザ要求分析，データベースの監査と管理，セキュリティと災害回復，コンピュータ要員管理，ソフトウェア開発方式の技術移転等.

③ 教育および社会関連……産学協同による技術者育成計画と研究計画，1985年における国民のコンピュータ使用能力 (computer literacy)，国民健康情報システムの構想，個人生活におけるコンピュータの影響 (Loving Grace Cybernetics Group による地域記憶 (community memory) の実験，官僚による国民監視）等.

④ 自動化オフィスおよびコンピュータ応用関連……電子郵便の現状，連邦政府におけるオフィス・オートメーションの実験，自然システムのシミュレーション（類人猿の言語行動，結晶の成長，脳における神経の成長と接続）等.

⑤ データベース・システムおよびコンピュータ応用……分散型データベース，データベース・マシン，医療用データベース，80年代における身体障害者に対するコンピュータの影響，核融合エネルギー研究におけるコンピュータ利用等.

⑥ 最も注目を集めた画像・自然言語処理および人工知能分野……画像データベース・モデルと問合せ言語，自然言語によるコンピュータ・インタフェース，産業界における人工知能の応用（油井データ解析，VLSI 設計プログラミング環境等)，学習モデルに基づく知的 CAI (Intelligent CAI)，エキスパート・システムと知識工学等.

なお，同会議のパイオニア・デーには，1951年に米国統計局にリリースされた UNIVAC I の30周年を記念する特別プログラムが行われた．また，コンピュータの歴史を専攻する3人の博士の誕生を記念して，コンピュータの歴史のセッションも開かれた.

●第8回コンピュータ・アーキテクチャ・シンポジウム——Minneapolis, 1981年5月11日～14日

基調講演者として S. Jerritts 氏 (HIS) を迎え，2つのチュートリアル・セミナー（分散処理，データ・フロー・アーキテクチャ）と12のペーパ・セッションが開かれ，40件の論文（日本からは4件）が発表された.

なお，Sperry Univac 社からは，K. E. Mackenzie が，「コンピュータ・アーキテクチャの創造説を否定する」と題して招待講演を行った.

MEMORANDUM

〈UNIVAC 関係の論文・講演等の要約〉

●プログラムとプログラム図式の公理的分析手法——ここで述べる公理的分析は，まず，プログラムから抽象プログラム図式を作成し，つづいて，この抽象プログラム図式の正当性を証明するというステップを踏む．用いられる公理は，計算公理，論理公理，領域公理の3つである．(T. J. Ostrand, "Axiomatic Analysis of Programs and Program Schemes," 13th Hawaii Int. Conf. on System Sciences, 1980)

●テスティング理論の動向——W. H. Howden の代数的プログラム・テスト，L. J. White と E. I. Cohen の領域テスト，E. J. Weyuker と T. J. Ostrand の暴露サブドメイン (revealing subdomain)，R. A. Demillo のプログラム変種作成，T. S. Chow や J. A. Bauer 他の有限状態機械モデル等をとりあげ，それらの比較を行っており，さらに有効な尺度も紹介されている．(T. J. Ostrand, et al., "Current Directions in the Theory of Testing," COMPSAC '80)

●プログラム・テストの本質的むずかしさ——テスト結果の正当性を正確にチェックできるのは，きわめてまれである．このことは，多くの決定問題が部分的可解 (partially solvable) であり，帰納的可解 (recursively solvable) でないことに対応している．神託仮説 (oracle assumption) が成立しない場合にも適用されるテスト法としては，①異なる2つのアルゴリズムによるプログラム出力の比較，②入力データの範囲を狭めてチェック，③出力の取りうる範囲を狭めるように入力を限定する，等の方法がある．テスト網羅率等の尺度の前提は，一般的に非決定 (nondecidable) で，本質的な欠点を持っている．(E. J. Weyuker, "The Oracle Assumption of Program Testing," 13th Hawaii Int. Conf. on System Sciences, 1980)

●ユーザの視点から見た制御言語の設計法について——制御言語に求められる特性として，①無矛盾性 ②自然性，③合理性，④完全性，⑤健全性があげられる．制御言語の設計に当たっては，ユーザの能力とシステムの使用の推移に関するモデルと，トランザクションの処理段階によるモデルが有用である．

また，制御言語の設計に当たっては，各洗練レベルに固有な制御言語の構造・機能の決定，無理なく洗練レベルが引き上げられるような機構の構築が必要である．このほか，人間工学的 (human factors) な諸技法の有効性にも言及している．(M. L. Schneider, "Designing Control Languages from Users Perspective," *Command Language Directions*, D. Beech ed., North-Holland, 1980)

●最初に修得したプログラミング言語が後のプログラミング能力に与える影響——プログラミング言語の場合でも，言語学の Sapir-Whort の仮説（言語が人間の思考法や世界観に大きな影響を与える）が当てはまると考え，それを検証するために，アンケートを収集した．ここでは，その結果のまとめ，さらに，J. Nichols 氏や A. Perllis 氏らのコメントが載せられている．(R. L. Wexelblat, "The Consequences of One's First Programming Language," 1980 ACM Joint Technical Symposium on Small and Personal Computer Systems)

●自然言語を用いたデータベース・インタフェースにおける不整入力の解決法——ここで述べる方法は，通常処理における規則違反は不整として表示され，通常処理の規則を変更する上位ルール (meta-rule) が動作し，誤りの同定と回復が行われる．現在，Sperry Univac のデータベースの英語フロント・エンドと Delaware 大学の質問応答システムに組込み評価中である．(N. K. Sondheimer, *et al.*, "A Problem with English Input to Data Bases," MICRO-DELCON '80)

●自然言語によるデータベース・インタフェース——データベースの自然言語（英語）インタフェースの進歩は著しく，SRI International の LIFTER, IBM San Jose の RENDEZVOUS, Dartmouth 大学の ROBOT (Artificial Intelligence Corp. により商品化) 等が知られている．ここでは，エンド・ユーザの観点から，これらの自然言語インタフェースの利点・機能等を評価するとともに会話例を示している．(M. H. Fogel, "Natural Language Data Base Interface: The User View," MICRO-DELCON '80)

●データベース・マシンのためのデータ構造と索引づけの手法——大容量のデータベースをサーチする場合，データベースをいくつかのブロックに分け，それと同数のマイクロプロセッサを用いて同時に並行・検索し，その結果をまとめるのがよい方法である．ここでは，PCAM (partitioned content-addressable memory) で採用したデータ構造，インデ

ックス編成法，クラスタリング法について述べている．(J. Banerjee, "Data Structuring and Indexing for Data Base Machines," 5th Workshop on Computer Architecture for Nonnumeric Processing, 1980)

●データベースの動的かつタイムリな再編成法——動的なインプレイス (in place) 再編成プログラムは，新旧のスキーマと，その間の写像 (mapping) 情報を包含するトランジション・スキーマ (transition schema) を用いることによりスムーズな移行を可能にするシステムである．そして，新旧スキーマ間の写像の記述には，現在のデータ記述言語 (DDL) が，さらに，再編成を制御するために再編成制御言語 (RCL), が用いられる．RCL としては，DMS 1100 の問合せ言語 QLP 1100 に類似した言語と，FORTRAN をホストとするデータ操作言語を考えており，現在，後者を入力としてインプレイス再編成プログラムを生成する RCL プロセッサのプロトタイプが試作されている．(T. B. Wilson, "The Description and Usage of Evolving Schemas," COMPSAC '80)

●分散型データベースのためのデータ・アーキテクチャ——分散型データベースでは，地域的データ独立性 (geographical data independence) でなければならない．地域的データ独立性を保持するため，ユーザ・プログラム中ではデータベース名を直接に指定できない．そこで，ユーザ・プログラムとは独立に，実行時のデータベース名を割り当てる手順を分散仕様として定義する．この分散仕様を，必要な分散度に合わせて，データ・アーキテクチャ(ユーザ・プログラムからサブスキーマ，スキーマ，蓄積スキーマを経てデータベースに至る階層) の適切な場所に挿入する．いま，分散仕様をユーザ・プログラムに近づけるほど各データベースの自律性 (autonomy) は高くなる．(J. Larson, "Data Architectures for Distributed Data Bases," COMPCON '80 Fall)

●分散型処理システムのためのシステム設計法——データ処理システムの開発工程は，①情報分析，②実現性調査，③システム分析・設計，④プログラム開発，⑤データ処理機器およびプログラムの購入，⑥システムの実現，⑦運用と保守，に分けられる．そして，システム分析・設計は，①ニーズの評価，②候補システム (candidate system) の設計，③候補システムの現実のプロダクトとの対応づけ，④候補システムの性能評価，⑤候補システムの採否の決定に分けて行う．なお，今後は分散システムのベンダーは，適切な費用と時間内に信頼性と詳細性をかね備えたシステム設計案の提供を迫られよう．(R. C. Heinselman, "System Design Selection for Distributed Data Systems," COMPCON '80 Fall)

●分散システムへの人間工学の適用——コンピュータ人間工学 (computers human factors) は，認識心理学 (cognitive psychology) と人間工学 (ergonomics) から発展した学問である．初期のコンピュータ人間工学は，単一コンピュータ・システムを対象としていた．最近は急速にソフトウェア・ユーザ・インタフェースの研究が進められており，コマンド，対話，メニュ等の入力形式や自然言語の利用等が研究されている．また，分散システムでは，ISO の開放システム間接続 (OSI) モデルの第 7 層 (アプリケーション層) における統一的インタフェースの研究が行われている．(M. L. Schneider, "The Application of Human Factors to Distributed Systems," COMPCON '80 Fall)

●ローカル・ネットワークの現状——ローカル・ネットワーク (LN) (本誌 pp. 102を参照) 形態ごとに分類すると，まず，開発の背景の相違により新規システムと既存システムを拡張したものに大別される．さらに，コミュニケーション技術の見地からパケット交換，回線交換，バス結合，入出力チャネル結合に細分される．(K. J. Thurber, "Perspective on Local Networks," 13 th Hawaii Int. Conf. on System Sciences, 1980)

●ISO の開放システム間接続 (OSI) 参照モデルの中間 2 層の機能——セッション層は，エンド・ユーザ・アプリケーション・プロセスを代表するプレゼンテーション・エンティティ間の構造的対話を支援するセッション機能を提供するものであり，トランスポート層は送信側の開放システムにおけるセッション・エンティティから受信側の開放システムのセッション・エンティティへと，高信頼性のエンド・ツー・エンドのデータ転送を行うもの，との国際的合意が1980年の ISO/TC79/SC16/NG 6 の会議で得られた．また，サービス機能の仕様定義は，インタフェースの実現と独立なサービス・プリミティブ (service primitive) を用いている．(J. P. McGoven, "Middle Layers of Open Systems Interconnection ——Session and Transport," COMPCON '80 Fall)

EDITORS' NOTE

渡部の3次元磁場解析への有限要素法の適用は，変分原理を用いて，ベクトル・ポテンシャルに対する有限要素法による定式化を与え，任意形状の3次元静磁場解析への有限要素法の適用を可能にした一結果を述べている．渡部は既に静磁場解析への有限要素法の適用を試みており，その結果は昭和51年のユニバック研究会技術計算分科会 (SYSTEMS・1976・TECHNICAL REPORT-1) で発表している．本稿はその延長線上にある．有限要素法は従来構造力学の領域において利用され，著しい成果をあげたが，近年，電気・電子分野においても有限要素法の真価が認識されるようになった．しかし，磁場解析の分野では，磁気ヘッドの磁界の解析を中心に多数の成果が発表されているものの，3次元磁場解析の実例は数少ない．本稿の結果は最近高まりつつある磁場解析の要求に答えるものである．なお，日本ユニバック（株）では磁場解析専用の要素を NASTRAN に登録して提供する予定である．

山田の **Kolmogorov-Chaitin の計算論的情報量と Lempel-Ziv 万能データ圧縮算法**は，日本ユニバック(株)と Sperry Research Center との合意に基づいて進められてきた新しいデータ圧縮法の共同研究開発タスク・フォースの結果の一端を述べている．

記号列のランダム性の測度としては Shannon の確率論的情報量が広く知られており，従来の無雑音圧縮技術はこの Shannon の情報理論に立脚している．一方，記号列のランダム性を，記号列を生成する2進プログラムの極小サイズで計量する Kolmogorov-Chaitin の計算論的情報量がある．Lempel と Ziv は記号の複製過程によって，この計算論的情報量の構成的定義を与え，この斬新なアイデアによって無雑音圧縮算法を考案した．タスク・フォースではいくつかの万能圧縮算法とその実現を行ったが，本稿ではこのアイデアと基礎理論を紹介する目的で Kolmogorov-Chaitin の情報量，Lempel-Ziv の情報量および万能データ圧縮算法を述べる．また算法の実現と圧縮プログラムによる圧縮の事例にもふれる．作成された無雑音圧縮ソフトウェアは万能性，漸近的最良性を特性とし，さらにオンラインや実時間の処理に利用できる．

R. M. Sedmak と H. L. Liebergot の **VLSI で実現した汎用コンピュータの耐故障性**は VLSI で実現した汎用コンピュータの耐故障設計法を評価している．ここに述べる設計法は VLSI チップ内の故障を検出するすべてのロジックをチップ自体の中に実装し，さらに，故障状態でチップが誤動作したり故障の発生が検出不能になるおそれを最小にするようにこれらのロジックを分割設計する方法である．この方法によって故障検出に用いるチップ数の増加を従来の耐故障設計に比べ，約 1/10 に押えることができる評価を得ている．

L. S. Baumann と R. D. Coop の **OA のためのワークフロー・コントロール・モデル**は Sperry Univac 社の電子化オフィス・リサーチ・プロジェクトで行ったワークフロー・コントロールの研究結果を述べている．オフィスの生産性向上のために，ワークフロー・コントロールの機械化が求められている．本稿は M. D. Zisman の拡張 Petri net によるワークフロー・モデルを用いて，同一作業域内および複数の作業域間のワークフロー・コントロールが分離可能かつ機械化可能なプロセスであることを示している．

S. R. Greenwood の **MACRO：プログラム言語**は UNIVAC シリーズ 1100 の汎用マクロ・システム MACRO について報告している．言語処理系の支援ツールとして，MACRO は TELCON システムのネットワーク制御文の構文解析や DMS 1100 システムの FDML プログラムの構文解析，PL/I 言語の表層変換等に利用されている．

J. J. Hart の**高水準会話型デバッギング・システム AIDS**は，万能コンパイル系 UCS の作業環境で任意の高水準言語で記述された原始プログラムを会話型式でデバッグするシステム AIDS の機能を報告している．日本ユニバック(株)では，AIDS を PADS (Pragrammers Advanced Debugging System) という名前で提供する予定である．

T. N. Turba の**汎用構文解析プログラム GSA**は，万能コンパイル系 UCS の機械独立なフロント・エンド生産に使用している汎用構文解析プログラムの言語仕様を中心に報告している．GSA は，各種の言語プロセッサやその他アプリケーションの入力テキストの構文解析部を生成する際の強力なツールとして有効に活用できよう．

EDITORS' NOTE

技　報
UNIVAC TECHNOLOGY REVIEW
No. 1

発行日	昭和 56 年 8 月 31 日
発行人兼編集人	富田和夫
発行所	日本ユニバック株式会社 東京都港区赤坂 2-17-51 〒107 TEL (03) 585-4111 (代表)
頒布価格	1,500 円
印刷所	三美印刷株式会社