ISSN 0289-62

技報

UNIVAC TECHNOLOGY REVIEW

1985年2月　第8号

日本ユニバック

還元 (reduction) とデータ・フローを混合したモデルを使って併行処理プログラムを支援するようにした多重プロセッサ系のクラス Rediflow を導入する Rediflow のデータフロー・プロセスの中には，G. Kahn が明らかにしたアプローチに従って von Neumann 型のプロセスが入っている．R. M. Keller 達の**還元規則を動作原理とする多重プロセッサの性能シミュレーション**では，Rediflow における自動負荷バランス法とシミュレーションによるその妥当性の検証を述べている．また，予備的な性能評価のシミュレーション結果にもふれている．

従来の住宅 CAD システムの大半は，簡単な間取図を入力して平面図と立面図を自動的に描かせる自動製図システムであり，システム内にもつ情報も製図に必要な図形情報に限られていた．しかしながら，住宅 CAD システムを製図作業の省力化だけでなく，営業戦略のツールとして用いようとする傾向がある．これに応えて日本ユニバックでは，住宅の属性情報を忠実に表現した家モデルを3次元でもち，この家モデルをもとにした住宅設計一貫システム HCAD (Housing CAD System) を開発した．篠田の**3次元家モデルに基づく住宅設計一貫システム**は，HCAD の概要と家モデルの構造について述べている．

従来の Sperry 社製の端末装置は，すべて文字発生機構と文字テーブルによってテキスト・データを生成し，一度に一種類の書体しか表示できなかった．また，図形表示の機能にも制約があった．そこで，これらの限界を解決する新しいアプローチが試みられた．R. G. Bond 達の**ラスタ図形処理ワークステーション**は，執筆者達が開発したマウスのついたビットマップ型ワークステーション FROG (Fast, Raster-Oriented Graphics project) と呼ばれる汎用ワークステーションによって，新しいアプローチを評価し，報告したものである．

テスト・データを作るには，テスト・データを抽象的に記述したテスト・ケースをいくつか設定する．テスト・ケースの設定には，外部仕様に基づいて創出するブラック・ボックス法と，できあがったプログラムに基づいて見つけるホワイト・ボックス法の二つの考え方がある．長谷川の**経路述語によるテスト・データの作成**は，ホワイト・ボックス法の一つである Huang の方法に基づいたテスト・データの作成を試みた手順を述べたものである．Huang の方法は，プログラム作成者側のモジュール・テストや単体テストの段階で有効であるが，人手で行うにはあまりにも労の多い作業であるので，コンピュータによる支援をあわせて提案している．

桑野の **LISP を用いて UNIVAC シリーズ 1100 上に作った4種の PROLOG 処理系**は，自然言語処理のための実験を UNIVAC シリーズ 1100 システムで行うために作った PROLOG の処理系の報告である．また応用事例として，英文から日本文への単純な翻訳の事例を示している．これは，コンパイルした PROLOG を用いて構文解析の結果得た英文の中間表現を日本文の中間表現に変換し，日本文の生成を行ったものである．

CMOS セミカスタム・ライブラリの能力を論証するために設計された PROTEUS は，他のマイクロプロセッサをエミュレートする機能もあり，潜在的な応用性をもつ 32 ビット CMOS マイクロプロセッサである．K. LeClair 達の**セミカスタム・セル・ライブラリを使った 32 ビット CMOS マイクロプロセッサ**は，この PROTEUS のアーキテクチャ，マイクロプログラマブル・フィーチャ，エミュレーション機能等について述べたものである．

差分法は，直交直線格子系法，任意節点配置法，座標変換法の三つに大別できる．直交直線格子系差分法は，計算手続きが簡単で内部精度が良好である．しかし，外部精度が不良で実用的な工学問題の処理に不適当である．任意節点配置差分法では，この点は改良されているが，計算量の大型化が懸念される。物理平面における任意形状の領域を解析平面に写像する座標変換差分法は，内・外精度ともに改善される．藤野の**非直交曲線格子系による座標変換差分法**は，非直交曲線座標系を導入し，解析平面内に格子定数の直交直線格子系を設定する方法を解説している．

さらに，藤野の**連続体解析のための新しい差分法について**は，筆者が本誌上で 1981 年から論じてきた差分法の論文を概観したものである．

☆

論文 還元規則を動作原理とする多重プロセッサの性能シミュレーション

Simulated Performance of a Reduction-Based Multiprocessor

R. M. Keller, F. C. H. Lin

要 約 還元 (reduction) とデータフローを混合したモデルを使って併行処理プログラムを支援するように設計した多重プロセッサ系のクラス，Rediflow を導入する Rediflow のデータフロー・プロセスの中には，G. Kahn が明らかにしたアプローチに従って von Neumann 型のプロセスが入っている．本稿では，Rediflow における自動負荷バランス法とシミュレーションによるその妥当性の検証を述べる．また，予備的な性能評価のシミュレーション結果にもふれる．

Abstract We introduce Rediflow, a class of multiprocessing system being designed to support concurrent programming through a hybrid model of reduction and dataflow. Our particular notion of dataflow includes a disciplined form of von Neumann processes, following an approach articulated by G. Kahn. We describe a technique for automatic load-balancing in Rediflow, and its validation via simulation. Some preliminary simulated performance measurements are included.

1. はじめに

多重プロセッサ (multiprocessor) 系を使用すれば，現存の素子技術で可能なシステムの処理速度をはるかに超えて速度を向上することが可能である．しかし，これらの系では各プロセッサへ作業を割り付け，プロセッサから正しい結果を取り出す手段が不可欠であるし，大多数の応用では，処理の再現可能性 (repeatability)，すなわち，決定性 (determinacy) や速度からの処理の独立 (speed-independence) あるいは“きわどい競走 (critical races)”の解消等を保証しなければならない．たとえば，もし共通変数 x を共有 (share) する $x_i = x+1$ や if $x=0$ then… else… のような文を別々の併行 (concurrent) プロセスで許すと，決定性が損われることがある．この場合，さらに（プロセッサの）実行順序に依存して，幾とおりもの出力が出て，（プロセッサ間で）きわどい競争が起こるかもしれない．しかし，関数型言語 (functional language) にもとづいて多重プロセッサ系を構成すれば，このような危険が回避できる．

本稿では論理的に透明に (transparent) プログラムが書けるように多重プロセッサ系を構成することを考える．論理的に透明であるとは，特定の構成について性能を最適化するという場合を除いて多重プロセッサと単一プロセッサの区別をプログラマから隠すということであり，プログラマが陽にプロセスを併行処理するように設定したり，プロセス間の同期を取ることがないようにすることを意味する．

2. 言語と併行処理 (concurrency)

関数型言語で書いたプログラムには，かなりの併行性 (concurrency) がある．関数型プログラムの実行は概念的には純粋に式の評価 (evaluation) のみであり，記憶セルへの値の

代入を基本とはしない．したがって，一つの関数から他の関数への“副作用 (side-effects)”はありえず，決定性が保証される．すなわち，実行中のプロセッサ間の通信やプロセッサ数に無関係に，プログラムは同一の結果を出す．だから，これらの関数型言語は多重プロセッサと単一プロセッサ (uniprocessor) との区別をしたくないときには，多重プロセッサのプログラムに理想的な言語であると考えられる．関数型言語はそのほかに，すでに論じられているような概念上の利点をもっている[7-9]．

関数型言語でどのように併行実行を行うかを示すために，次の式を考えよう：

max［部分式１，部分式２］

ここで，max は普通の最大値を返す関数（あるいは２引数の任意の関数）である．

併行実行モデル (concurrent execution model) では次の三つの段階を実行する．

1) 二つの部分式を併行評価するタスクの生成
2) 両方の評価の完了を定めるための同期
3) 2) が完了したときの最大値の評価

明らかに３番目の段階だけが逐次 (sequential) 処理で実現されるが，上の２段階には関数の併行評価がある．これらの仕様は，普通プロセス指向言語では対照的に陽に要求される．適当にデータを構造化すると，併行処理可能なタスク数は増加する．たとえば，多くの関数型言語では列を値とする式を扱うが，列はリスト，配列あるいは木で表される．記号\\で示す apply-to-all のような作用素を使うと，同様の表現を二つの列の中の成分の対に順に作用する max のような関数に適用できる：たとえば，

max \\ ([1, 3, 5], [6, 4, 2]) = [6, 4, 5]

そこで，長さ n の列の対に max を適用すると併行実行できる n 個の独立したタスクが生成されよう．さらに，これらの列の中に評価されていない部分式が含まれていれば，部分式を評価するタスクが追加して生成されよう．もちろん，ここで関数 max はもっと複雑な関数に置き換えてもよい．このような併行性は関数型言語では利用できるが，PASCAL のような言語では利用できない．なぜなら，PASCAL のような言語では，関数の引数を評価するとパラメタや大域的データを修正する副作用が起こりうるからである．したがって，実行順序によって副作用が起こるために，処理は一般に反復不能である．

カプセル化 (encapusalation) しても，局所的副作用のために多数の分散した局所的状態を扱うことになる．実際，全体をカプセル化したときは，局所的副作用のみをもつ逐次型プログラムは制限した形の関数型プログラムに対する意味論上の省略形であると考えられる[10]．

適当な関数型の枠組を作ると，非決定性の構成を組入れられる．たとえば，われわれは非決定性の“merge”を許す単純な拡張を使って関数型言語を補足し，併行更新を含む分散データベースの処理の直列化を保証する方法を示した[11]．

関数型の枠組では，さらに部分プログラムをプロセッサとメモリに割り当てることもできる．たとえば，“site pragma”は特定のサイトにあるデータベースの特定の部分トランザクションを強制的に実行させることができる．

3. 併行実行モデル

タスクをプロセッサに割り付け，正しく結果を取り出す実行モデルはその機能に応じて次の四つのカテゴリに分けられる．

3.1 共有メモリをもつ多重プロセス

“プロセス (process)” の概念は von Neumann 型計算機でのプログラムの実行を抽象したものである．プロセスはレジスタへの代入やテスト等を指示する命令の列に従う．

最初の併行計算モデルでは，一つの共通なメモリ空間内にこのようないくつかのプロセスを生成することを基本としていた[13]．プロセス間の通信は他のプロセスが値を代入したレジスタを調べる方法をもった．こうした通信を矛盾なく行うためにいろいろな同期の機構が考案されている[14]．

3.2 メッセージの受渡し (message-passing) による多重プロセス

非決定性の原因を解消し，その効果を分離するために，記憶場所の一般的な共有を禁止する系がある．これらの系では通信の基本手段としてメッセージの受渡し (message-passing) を採用している．この方式ではプロセスで他のプロセスを名指すか共通リンク・チャンネルを名指して他のプロセスへ送るメッセージを指定する．決定性を保証するために，ある基準をメッセージ受渡し系に設定することができる．たとえば，もし，

1) プロセスが入力メッセージ・バッファを調べるときには，プロセスはバッファにメッセージが入るのを待ち，

2) かつ，どの二つのプロセスも共通の入力バッファを共有せず，

3) どの二つのプロセスも共通の出力バッファを共有しないならば，

動作の決定性は保証される[6,10]．

いいかえれば，上の基準が満たされるならば，プロセスの集まりは全体として関数として作用する．この関数としての作用は制限した形で UNIX のプロセスをパイプ (pipe) で結合するのに利用されている[15]．しかし，このパイプ結合は関数の合成の特別の場合と見なせる．その他の関数型言語ではこの現象をより一般的な形式で利用しようとしている．

3.3 データフロー (dataflow)

データフロー計算機[16,17]では，メッセージの受渡しを小さく分解したタスク単位で行っている．結局，各基本演算は実際には値のストリームに時々刻々同じ操作を行う“プロセス”である．それによって，データフロー機械では逐次的処理に伴うオーバヘッドを消去しようとしている．

データフロー機械のプログラムは一般に有向グラフで表現される．グラフで各節点 (node) は演算子を表し，弧 (arc) はメッセージ・キュー (message queue) を表す．メッセージ・キューは1メッセージの容量をもつと考える．非同期性を増して，プロセスがさまざまの速度でメッセージを生成する場合に最大限の併行処理を実現しようとするときには，プロセスの処理速度をバランスさせるために恒等 (identity) 演算子を導入できる*．これらの恒等演算子は多くのバッファ・ステーションを作るが，これを通してグラフ内の一つの節点から他の節点へメッセージが渡される．残念ながら，データに依存して反復回数を変更するサイクリックなプログラムをバランスさせるアルゴリズムは存在しない．

3.4 グラフ還元による評価

最大限の非同期性を実現するには，できるかぎり値の生成を値の消費から分離する．

グラフ還元モデル (graph reduction model) にはこの分離手段がある．とくに，このモデルでは限定バッファ (bounded buffer) 方式の代わりに，大域ストレージ・プールから引き出したセルを結合するリストを用いる．この還元モデルによってこのような効果をエレガントに実現できる．

---

*(訳注) 恒等演算子は単に遅延作用のみを起こす．

還元モデルではプロセスより細かい単位でタスクを生成する．すなわち，このタスクの単位は従来の言語での関数呼出しに対応している．そしてタスクは少数の算術演算や論理演算，あるいはデータ構造の一部にストレージを割り当てる操作等の構造化操作を行う．しかし一般にタスクは長時間にわたる逐次型の動作を行うのではなく，むしろ他の同様のタスクを生成することで処理を継続する．したがって，タスクが他の二つのタスクを生成するという方法で高速で2分化する計算が表現できるが，逐次的計算では一つのタスクが仕事を終るとき，もう一つのタスクを生成してから終了するという仕方で表現される．

タスクの生成を Burge[7] と同様の関数型記法で表現したプログラムで説明しよう．階乗関数を次のように“分割統治 (divide-and-conquer)”によって計算することができる：

$$\text{Factorial}(x) = \text{DAC}(1, x)$$
$$\textbf{where } \text{DAC}(m, n) =$$
$$\textbf{if } m = n$$
$$\textbf{then } m$$
$$\textbf{else } \text{DAC}(m, \text{med}) * \text{DAC}(\text{med}+1, n)$$
$$\textbf{where } \text{med} = (m+n)/2$$

還元モデルでは，DAC の例 (instance) が要求されるたびに，一つのタスクとそのためのストレージが配置される．たとえば，Factorial (100) は例 DAC (1, 100) を要求するが，例 DAC (m, n) が与えられたとき，$m=n$ ならば (m を返して) 停止し，あるいは $m \neq n$ ならば，定義に従って二つの例を生成する．関数の例が計算されたとき，この例は関数の値に置き換わる．関数型言語では一般性を失うことなくある引数をとる関数を常にその関数値に置き換えられるから，こうしてもよい*．したがって，この例を共有するならば，値を1回計算するだけですむ．

2番目の例で還元モデルの中のプロセス様の動作を示す．いま，プロセスに列；

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

を計算させるとしよう．ここでは後続関係を表し，“あとにくる”と呼ぶ．

次の関数 NUMS-FROM はその引数 n 以降の数を計算するプロセスを表している：

$$\text{NUMS-FROM}(n) = n\ \text{NUMS-FROM}(n+1)$$

還元モデルでは，NUMS-FROM(n) への要求は図1に示すように n のあとに NUMS-FROM$(n+1)$ の例がくるデータ構造を生成する．

しかし，この例は要求がくるまで suspend[18] されたままであり，producer プロセスに

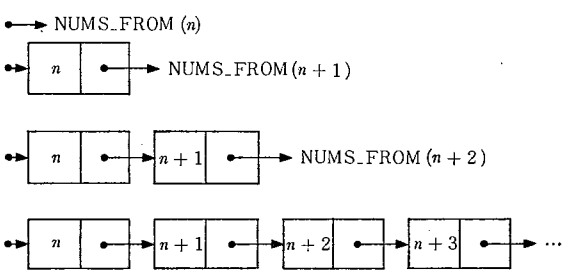


図 1　還元による列の計算（矢印は von Neumann メモリの中のポインタを表す.）

Fig. 1　Computing a sequence by reduction. Arrows are conventional pointers in a von Neumann memory

*(訳注)　論理的には，call-by-value rule でよい．たとえば Kleene の full substitution rule を考えればよい．

似ている．列が要求されるたびにこの構造は生成されていく．もし多数の要求がきわめて速く生成されると，評価系 (evaluator) は単に計算された結果の値を受け取る構造だけを設ける．同様に transducer プロセスも定義できる．たとえば，数列の各要素を平方する SQUARER は HEAD(ay)≜a と TAIL(ay)≜y を使って定義される[8,18-20]：

$$\mathrm{SQUARER}(x)\triangleq \mathrm{HEAD}(x)**2\mathrm{SQUARER}(\mathrm{TAIL}(x))$$

下の関数 PARTIAL_SUMS は入力列の関数であり，結果的にはその入力列の各要素に依存して出力を出す：

$$\mathrm{PARTIAL\_SUMS}(x)=\mathrm{AUX}(x,0)$$
$$\textbf{where } \mathrm{AUX}(x,ac)=b\mathrm{AUX}(\mathrm{TAIL}(x),b)$$
$$\textbf{and } b=ac+\mathrm{HEAD}(x)$$

この場合の“状態 (state)”は補助関数 AUX の第2引数（“アキュムレータ (accumulator)”として働く引数）で表される．

たまたま，この例は関数型プログラムでは“状態 (state)”あるいは履歴に依存する (history-sensitive)”操作をモデル化できないという神話に対する反例となっている．

プロセスのパイプ結合が関数の合成であることを想起して，図2に示すように三つの関数 NUMS_FROM, SQUARER, PARTIAL_SUMS を”パイプライン (pipeline)”に結合すると，関数の合成 (composition) の意味が直観的な動作で表されることがわかる．パイプラインの出力は $n**2$ で始まる平方の列の部分和の列である．

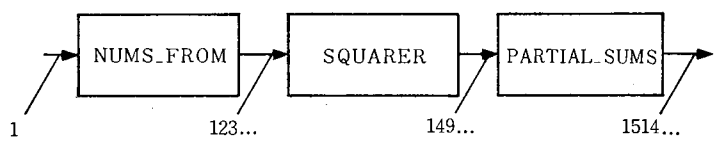


図 2 プロセスのパイプライン結合

Fig. 2 Pipeline interconection of processes

還元モデルとデータフロー・モデルを比べるために，関数 SQUARER の出力を第2の関数 POLY に入力しよう．POLY は高次の多項式のように各入力のある複雑な関数を計算する．ここで PARTIAL_SUMS と POLY は同じ速度でストリーム (stream) を消費するとは限らない．図3は上の関数のある還元計算で配置されるストレージ・セルを示している．明らかに，SQUARER の出力にどれだけのバッファ領域を配置すべきかをあらかじめ予測するのは困難であるから，還元モデルで用いる動的配置方式 (dynamic allocation

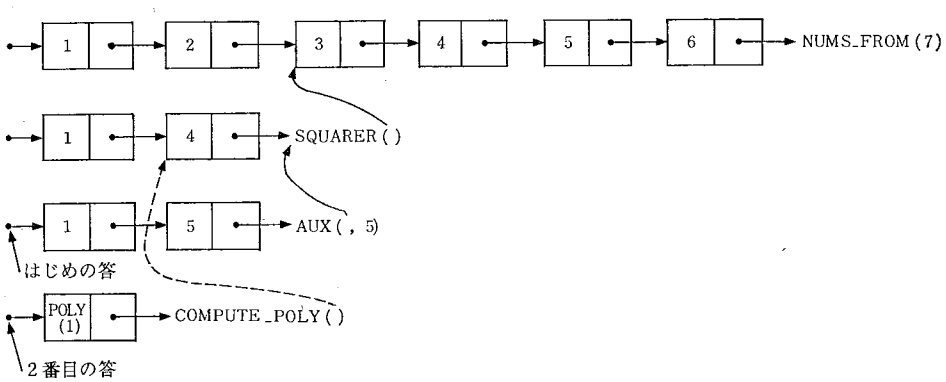


図 3 パイプラインの還元実現

Fig. 3 Reduction implementation of pipelining

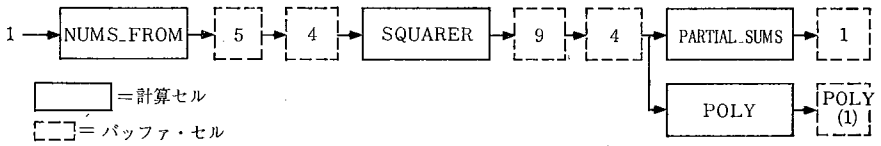


図 4　パイプラインのデータフロー実現

Fig. 4　Dataflow implementation of pipelining

scheme) は図4に示す限定バッファ (bounded buffer) 方式でのデータフローの実現に課せられる制約を緩和するのに役立つ.

3.5　還元とデータフローの結合

還元モデルとデータフロー・モデル (それぞれ“構造 (structure)”モデルと“トークン (token)”モデルとも呼ぶ[21]) はそれぞれ欠点をもっている. すでに見たように, 還元モデルは, 非同期性に予測不能なバッファリングが必要であるときや帰納 (recursion) を含む迅速なタスクの分割のときに役に立つ. 一方, ストリームによる通信が高度の非同期性を要しないときには, データフロー方式にはストレージの配置が要らないという長所がある. U-interpreter[22] のように“unfold”するデータフロー・モデルはデータフロー・モデルの枠内で還元モデルのループ展開機能 (unwinding) に近づいている.

還元モデルの一つの欠点は本質的に逐次的な計算の実現の効率が悪いことである. この場合, 還元モデルは上の関数 SQUARER のように“テール帰納 (tail recursion)”を使いループの各反復ごとにタスクを配置する. もし SQUARER の出力の消費が十分に遅ければ, 還元で用意される展開作用はいらない. 逐次的に SQUARER を計算する方が効率がよいが, メッセージ受渡し法を使えば還元モデルの枠内にこのような逐次的計算を組入れることができる. この手法によって, von Neumann コードをストリーム処理関数の節点の中にカプセル化できる[10]. この方法が次節で述べる Rediflow 系の中にデータフローの動作を導入するために採用したアプローチである. Rediflow では, 還元モデルの構造の中のポインタを使ってトークンが流れる論理チャンネルを設けることができる. ポインタを使うと動的に生成されるデータフロー・グラフを設定するのに便利である. このような関数を随意に使ってより複雑な系を構築したり, 純粋 (pure) の還元で実現される系とインタフェースをとることができる. 実際, 2個の単純な関数を使えば, 還元で実現した関数とデータフローで実現した関数のインタコースをとることができる. すなわち, 一つは構造からトークンのストリームを作り出す関数であり, もう一つはトークンのストリームから構造を作る関数である*. 次のシミュレーションは評価系の純粋に還元だけの部分に依っている. 大きな, 本質的に逐次的成分を含む例題の性能は, 提案する Rediflow 統合によって改良されよう.

4.　Rediflow 系の構成

“Rediflow”は多重プロセッサ設計とそれに付随するソフトウェアに対するわれわれの関数による概念構成に与えた名前であり,“還元 (reduction)”と“データフロー (dataflow)”の合成語である. われわれのモデルには von Neumann 型評価モデルも含まれる. 関数型のアプローチについては述べたから, つぎにその物理的構成の問題を論じる.

---

* 詳細と, この構成の使用については Tanaka[23] を参照. 本稿も対応する言語構成への全体的アプローチを述べている[24].

4.1 ハードウェアの問題

プロセッサを組み合わせて多重プロセッサを構成する際の主要問題は過大な通信のオーバーヘッドを回避しながら効果的にタスクを（プロセッサ間に）分配することである．一般的に言えば，プロセッサおよびメモリ間のリンクを設け，適当な計算機言語で書いたプログラムをでき上った系の上に写像（map）しなければならない．素子技術を不変とすれば，多重プロセス系の成否を決定するのはこの写像の容易さである．

写像の容易さは応用のクラスや使用言語，コンパイラおよび基本となるハードウェア構成に依存する．特定の応用を固定し，専用機械を設計すれば明らかに，その応用に対して他のすべての機械を上回るようにできる．しかし，われわれの関心はそのような専用機械にあるのではなく，広範囲な応用に対して多重プロセス能力が発揮できるような技術にある．この応用範囲を限定するには，応用を規則性（regularity）やサイズ・スパン（size span）や細分単位（granularity）によってグループ分けするのがよい．

高い規則性をもつ応用では類似した計算要求をもつ多数の非常に似た操作が現れる．このような場合には，ベクトル，プロセッサとか細胞アレイ（cellular array）とか静的データフロー（static dataflow）といったアプローチが適当であろう[16]．一群の応用のサイズ・スパン特性は系のライフ・タイム中に問題のサイズが変化する程度に関係する．単一のアレイ・ロードに含められるような問題には特定のアレイ・プロセッサが理想的であるが，大きな問題をプロセッサ構成にマッチするようにまた受け入れられる性能を維持するように展開したり分解するのには困難が伴う．このような規則性はないが，非常に大きなサイズ・スパンをもつ問題に適合するように Rediflow を設計した．

ビット操作や算術演算のレベルの操作単位は非常に細かい．一方，プロセスやジョブ全体の細分単位は大きい．Rediflow では中間のあるいは関数レベルの細分単位以上の細分単位が適当な応用を目的とする．知識ベース・システムのように不規則に構成された問題がこのレベルの細分単位を使うのに適している．これらの問題では，逐次的に実行しなければならないために単純な算術演算よりは粗い細分単位となるが，大多数のプロセスよりは細かい細分単位となるような操作列が一般的である．この中間レベルの細分単位をもつ他の応用としては適応性（adaptive）の数値計算やあるタイプの信号処理がある．Rediflow はさらに予測不能な相互作用を行ういろいろな応用領域をも目的としている．

4.2 細 分 単 位

細分単位の二つのトレードオフに，通信のオーバーヘッドと負荷バランス(load balancing)の柔軟性がある．細分対象（granule）間に過度のデータ通信があれば，システムはそのピーク能力を発揮できなくなる．この形式の通信は同等の純粋の逐次的計算には存在しないが，複数個のプロセッサがあるときには無視できない．細分対象が非常に小さければ，通信による遅延が操作自体の遅延を超えてしまう．このために，規則性が十分に高く，必要な通信路が比較的短くかつ静的であるか，または細分対象間の通信が少ないときに限って，小さい細分単位を使用できる．小さい細分対象を広く見境いなく分散させれば，通信のオーバーヘッドが増大することになる．Rediflow ではいくつかの細かい操作をまとめて一つの関数本体の中に組入れて，分散を避けるようにしている[20]．このようにして，逐次的な操作列の実行速度を von Neumann 型計算機での速度に近づけることができる．細分単位の選択に影響するもう一つの要因は負荷バランスである．負荷バランスとは細分対象を処理装置たちに分配することである．理想的な状況とはすべての処理装置に同サイズの細分対象を最初に 1 回分配すればすむ状況である．

しかし，多くの応用では仕事の負荷がデータに依存し，静的に解析できないので，あらかじめこのような決定を下せる状況は極めてまれである．そのため，使える多重プロセス資源を十分に活用して最高の速度に達するようにするには負荷を動的に分配する必要がある．この場合，均等にバランスできる小さな細分対象と全体として分配の手間を最小にする大きな細分対象の間のトレードオフに注意しなければならない．還元モデルでは，広く仕事の負荷を系全体に広げるのに十分なほど細かい細分ではあるが，不当な分配コストが掛かるほどは細かくない細分が得られるように思える．このことを最初のシミュレーションで検証した．

本研究では還元評価で現れる中間レベルの細分タスクのバランスに注目する．4.1節に述べた埋め込んだデータフロー・プロセスは大きな細分プロセスであるが，これらのプロセスの長所は負荷バランスに与える余計な複雑さと勘案しなければならない．この問題は十分に調べたわけではないが，大きい細分プロセスはガーベージ・コレクションの間に再びバランスさせることができる．しかし，このアプローチの得失はまだ分析していない．

4.3 相互連絡 (interconnection) の問題

多重プロセス系は，応用の細分単位による分類と同様にプロセッサとメモリの構造によっても分類できる．その一方の極にあるのが“共有メモリ (shared memory)”構成である（図5）．図はスイッチング・ネットワークで分離されたプロセッサ(P)とメモリ(M)とを示す．

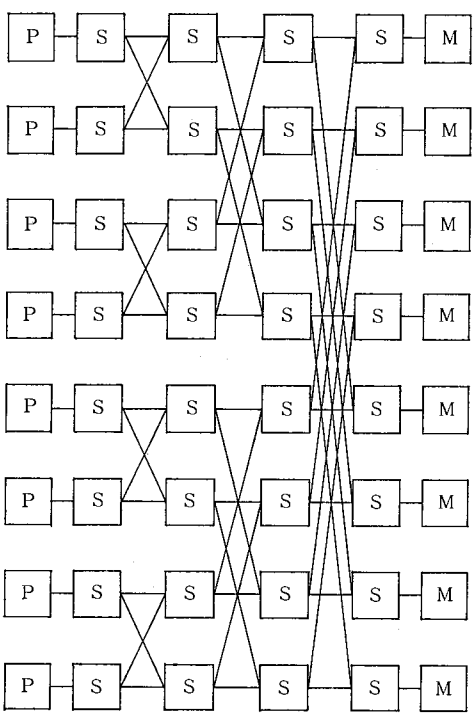


図 5 共有メモリ多重プロセッサ構成

Fig. 5 Shared-memory multiprocessor configuration

図によると，任意のプロセッサは任意のメモリに一様 (uniform) なアクセス時間でアクセスできるようにみえるが，個々のスイッチで無視できない競争があれば，一様性は保たれない．遺憾ながら，この一様な遅延はさらにプロセッサとメモリの個数が増加するに従い一様に遅くなる．プロセッサに密接に (closely) 連結したキャッシュ (cashes) を導入して

局所的情報に速くアクセスできるようにすることは可能である．しかし，キャッシュもまた“無矛盾性 (coherence)”についてのむずかしい問題を起こす[25,26]．たとえば，あるプロセッサが他のプロセッサによってキャッシュに入れられた情報を更新すると，あとのプロセッサに対してこの情報を無効にしなければならず，結局，余分なプロセッサ間通信がいることになる．この問題を解決する機構を導入すれば，システムの有効な能力を低下させるだけである．

さて，もう一方の極では，Rediflow 構成等で，各プロセッサを密に一つのメモリと対にし，パケット・スイッチを使ってこれらの対の間の通信を行う方法がある．

まれに“軟結合 (loosely coupled)”系と呼ばれるが，各プロセッサとメモリの対に関する限り，この結合は現実には非常に硬い (tight)．

たしかに，(プロセッサとメモリの) n 対からなる系のピーク帯域幅は大きなスイッチで分離された n プロセッサと n メモリの系より非常に高い．対からなる構成については次の諸点があげられる：

1) 各プロセッサ・メモリの参照では最悪の場合の遅延 (worst-case delay) は起こらない．多くの場合遅延はたかだか1回の局所アクセス時間であり，平均するとその遅延は最悪の場合より小さい．
2) この構成では“局所性 (locality)”を利用できる．局所性とは，論理的に関連する操作が行うデータの部分への参照を一つにまとめることができることを意味する．Rediflow では，関数レベルの細分でこのようなまとめがある程度可能である．
3) 各プロセッサは対をなす自分自身のメモリを排他的 (exclusive) に制御するから，無矛盾性の問題は起こらない．この排他的制御によりさらに test-and-set 等の多重プロセッサのメモリ・アクセス用の特殊な命令を使う必要がなくなる[29]．
4) 一つのスイッチ用ハードウェア成分を各プロセッサに付けるから，全体としてのスイッチ・ハードウェアのコストは ($O(n \log n)$ 以上というより) プロセッサ数 n に関して線形に増大する．

Rediflow では，プロセッサ・メモリ対と情報移送用のパケット・スイッチの組合せを Xputer と呼んでいる．この名前は INMOS 社の“トランスピュータ (transputer)”チップとの類似性から付けられた．Rediflow の Xputer の情報の流れのスケッチを図6に示す．

Rediflow* のシステム・レベルの構成では，図7に示すシャッフル交換，図8の格子，

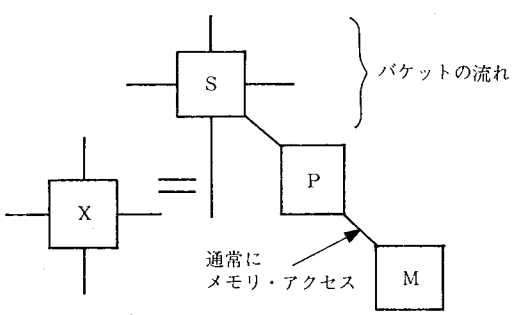


図 6 Xputer のスケッチ

Fig. 6 Sketch of an Xputer

*(訳注) Xputer を素元としている Rediflow である．

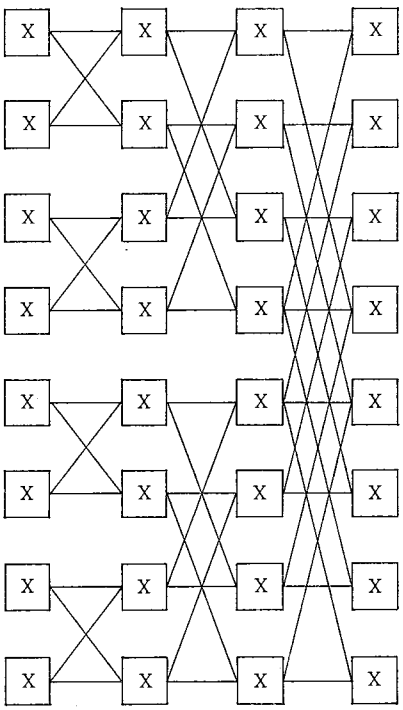


図 7 Xputer のシャッフル交換ネットワーク
Fig. 7 Shuffle-exchange Xputer network

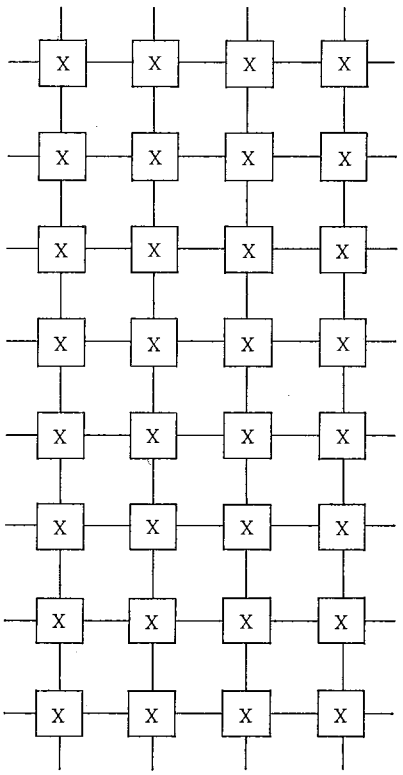


図 8 Xputer の格子状ネットワーク
Fig. 8 Grid Xputer network

その他，多様な立方体の構成を含むいろいろの相互連絡ネットワークが組める．

効果的なオペレーションに必要な最小の仮定は以下のとおりである：

1) 番地付けできること (addressability)

異なる Xputer で併行に実行される関数間のリンクを動的に設けられるように，全システム内の任意のメモリ場所に一意的に番地付けを行う方法が存在すること．

2) ルートで結べること (routability)

指定した番地のメモリからデータを取り出したり (fetch)，そのメモリに蓄えたり (store) する要求が与えられたとき，あらかじめ設定されているリンクをたどってその要求をどこへ回すかをスイッチは決定できなければならない．

リンクは純粋に仮想的であり，ポインタで作られ，リンク専用の論理パスというものはない．

しかし，図9に示すようにスイッチ，プロセッサ，メモリを論理的に並列な層に構成することができる．この場合 Xputer 間の相互連絡はスイッチの層だけにあり，メモリ層内のメモリは結び合わされて一つの大域的アドレス空間をなす．ある Xputer が他の Xputer のメモリにアクセスするときには，アクセスしたいアドレスを含んだ要求パケットを作る．このパケットはスイッチ層内をたどり，アクセスしたいアドレスをもっている Xputer に送られる．そして結果のパケットが作られ，それが要求を出した Xputer に回送される．この要求 (request)/回送 (return) 機構は還元評価のデマンド (demand)/ドライブ (drive) 機構と統合され，関数の遠隔呼出しが行われる[20]．

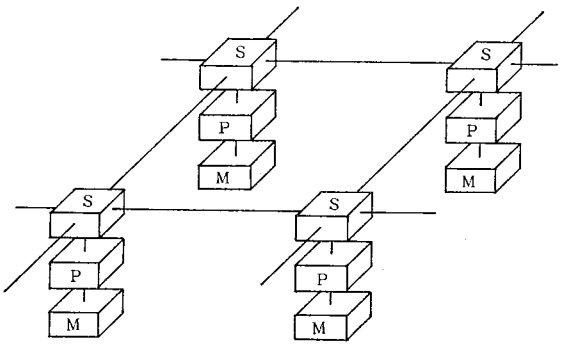


図 9 層状の格子状ネットワーク
Fig. 9 Layered view of grid network

入出力装置は図に示されていないが，任意の節点に付けることができる．前述のように順編成 (sequential) の入出力装置とのインタフェースは von Neumann 型プロセスを通してとる．

多数の二次記憶装置を付ける場合にも何ら概念上のむずかしさはない．アドレス可能な装置を使えば，システム全体の広がるアドレス空間を作る仮想記憶機構を実現することもできる．

図 10 はいくつかの Xputer にまたがるデータ構造の様子を示している．ポインタは従来どおり参照に使う他に，Rediflow 内のデータフローを実現する論理チャネルを定めるためにも用いる．

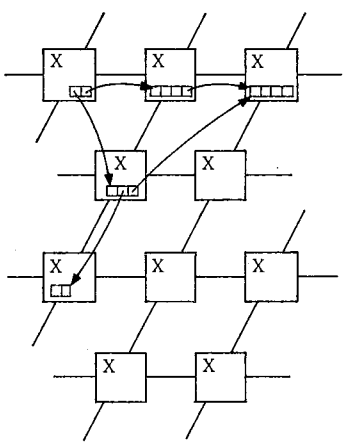


図 10 Xputer ネットワークにまたがる一つのデータ構造の広がり
Fig. 10 Spreading of a data structure over an Xputer network

4.4 負荷の分配とバランス法

システムの負荷を均衡させるときのボトル・ネックを避けるために，空きのプロセッサにタスクを分配する集中化したキューは使わない．その代わりに，タスクの移送(migration)法自体を分配する方法を採用する．また通信の遅延が無視できないほど細かくタスクを細分しないように，タスクの移送単位としては中間レベルの関数呼出しタスクを使う．たとえば，SQUARER の定義で示したように，プロセスに相当する動作をこのような呼出し

で実現できる．したがって，その結果のプロセスは移動可能 (mobile) である．負荷の分配を論じるために，まずタスクの移送に関係する機構を示す．

4.4.1 連結 (linkage) の機構

大多数の多重プロセッサでは，タスクのバックログが1本以上のキューに保存されるが，Rediflow においても各 Xputer ごとに4本のキューがある．すなわち；

1) ロケーションの内容に対する FETCH 要求の発生と (FORWARD と呼ぶ) これらを受理する IN キュー．
2) 移送可能な (migrable) 関数を作用させるタスクを保存する APPLY キュー．関数を大域アドレス空間に写像することによって，このキューにある関数を自由に他の Xputer へ移動することができる．
3) 移動不能の細かい細分タスクを保存する LOCAL キュー．
4) 他の Xputer に対する FETCH や FORWARD 要求を受け取る，あるいはこのような要求を受理する OUT キュー．

これらのキューの項目は四つのタイプのパケットで表す．すなわち；

1) 値を取り出すロケーションの番地と取り出した値を蓄えるロケーションの番地をもつ FETCH パケット．
2) 値とその値を蓄えるロケーションの番地をもつ FORWARD パケット．このロケーションには蓄えられる値を待っている (wait) suspend されたタスクへ通知する情報が入る．FORWARD パケットは前に出された FETCH パケットに対応して生成される．
3) APPLY パケットは，次の(a)(b)をもつ．
   (a) クロージャ (closure)
   作用させる関数を評価するコードへのポインタとその関数の大域的引数の値の組へのポインタをもつレコード．(引数の値の組へのポインタはラムダ算法の意味での自由変数の値に対応する．)
   (b) 作用させる関数に対する引数．この引数は関数が多変数のときには，値の組へのポインタとなる．
4) LOCAL キュー上のパケットは評価すべき操作節点の局所メモリ内の単なるアドレスである．これらは従来の計算機における“命令ポインタ”に似た働きをする．

結果のデータは，すべて大域的にアドレス可能なロケーションにあらかじめ配置されるから，データを宛先の Xputer に渡すためにトークンの照合を行う必要はない[17]．高速の von Neumann 型の番地付けを各 Xputer 内でも，すべての Xputer 間でも利用できる．この番地付けを使ってスイッチに最短ルートを指定することができる．このルート指定法は AMPS の初期の着想から直接採用したものである[21]．

ルート指定の手法を“トークン照合”の一つの形式と考えるのは正しくない．2次記憶あるいは常駐記憶からの最初の取り出しに続いて，Xputer 内の関数コードの常駐 (pure) コピーをキャッシュに入れるためには，最終的には内容を使う番地付け (content addressing) を限定した形で採用する必要がある．しかし，この番地付けは一つのラン (run) の中にある異なる関数コードの数が直接にインデックスが付けられるキャッシュに比べて大きすぎるときに限って必要である．論理コードと特定の Xputer との間には何の対応も存在しない．したがって，同一の関数を同時に多数の異なる Xputer で実行できるし，また一つの Xputer で多数の関数の実行を複合 (multiplex) できる．たとえば，ある関数がネットワー

クを通して渡されるデータを待っている間それを保留することもできる.

4.4.2 負荷の機構

移送可能なタスクをスイッチ層上に注がれた流体の分子とみなすことにしよう. そうすると, 移送の機構は, Xputer 間に流体を移動させる圧力 (pressure) の概念で簡単に記述できる. 一つの Xputer 内の内部圧力はその Xputer がどの程度忙しいかを示す. いいかえれば, 追加の仕事を処理する余裕 (availability) を表す. 現在のモデルでは, この内部圧力を Xputer の LOCAL と APPLY キューにあるパケット数と使用中のメモリ量だけで表す. 還元モデルでは基本的に動的メモリ配置系に依存しているから, この使用中のメモリ量は重要である. 目下使っている関数は次のように定めている:

$$\text{内部圧力} = \text{キューの長さ} + c\left(\frac{1}{1-\text{使用中のメモリ量の比}}\right)$$

ここで, c は定数である.

この関数はメモリがほとんど一様に使用されるまで, メモリの使用量の寄与を最小にしている.

われわれの負荷分配のバランス法では, Xputer が相互に圧力情報を見て過大なバックログをどこへ回すかを決定する方法を採る. しかし, この場合, ある Xputer がその内部圧力だけを他の Xputer に提供するだけでは十分でない. かりにこのデータで十分であるとすれば, 重く負荷がかかった Xputer がわずかな名目的な負荷がかかっている Xputer の壁によって囲まれ, この壁の外側に余分の負荷を受けられるような Xputer たちが存在することに気付かなくなるであろう. したがって, 各 Xputer が隣り合う Xputer に伝達する圧力の伝播 (propagate) の概念を導入することにする.

Xputer の伝播する圧力とはそれ自体の内部圧力と外部 (external) 圧力との関数であり, それはその隣りの Xputer の伝播圧力の関数となる.

ある Xputer の内部圧力が外部圧力を上回ると, その APPLY キューのあるパケットが相互連絡ネットワークの中に渡され, そこでより低い圧力の Xputer へ分配される. Rediflow はこのように低いポイントを見つけるように圧力の勾配に沿ってパケットを送ることができるスイッチを採用している. パケットが局所的に極小の圧力をもつ Xputer に到達すると, パケットはその Xputer の APPLY キューに吸収 (absorb) される. この吸収によってその Xputer の圧力は上昇し, タスクの完了によってその内部圧力を下げるまではパケットを受け付ける尤度 (likelihood) を低下させる. Xputer の圧力は明らかに変化し続けるから, それに応じて各 Xputer の外部圧力の認識をひんぱんに更新する必要がある. この更新では Xputer に十分な圧力の変化が起こったならば, その伝播圧力についてのデータを含んだパケットをその Xputer が送り出す必要がある. このために割合良好に働く一つの発見的関数では伝播圧力を次のような方程式で定める.

$$PP(X) = \textbf{if } PI(X) < \text{threshold } \textbf{then } 0$$
$$\textbf{else } \min[1 + PE(X), \text{ceiling}]$$

ここで, PP, PI, PE はそれぞれ伝播, 内部, 外部圧力であり, threshold は設定可能ないき値のパラメタである. ceiling としては, 1+ネットワークの直径 (同じ節点を2度以上たどらないような最長のパスの長さ) を用いる. PE としては,

$$PE(X) = \min\{PP(Y) \mid Y \text{ は } x \text{ と隣り合う近傍節点}\}$$

を使う.

上の関数を使うとパケットを最小の負荷の節点に向けて流すことができる. 実際,

$PE(X)$ がそのような節点に到達するためにたどるリンク数を与えることを示すことができる．各 x について $PP(X)$ を“連続的に”計算する方法は一つの緩和法であり，負荷は常に変化するから，必ずしも正確である必要はない．この計算の手間は十分に小さく，知的なパケット・スイッチに組み込むことができる．

4.5 飽和 (saturation) の効用

すべての Xputer が忙しく APPLY パケットを移送する試みがすべて無効に終ると，内部と外部の圧力差にもかかわらず飽和 (saturation) 現象が発生する．Rediflow の負荷バランス法の機構では，飽和を検出するために伝播圧力の値を上から押さえる ceiling 値を用いる．一つの Xputer の外部圧力がこの ceiling 値に達すると，移送は停止する．局所的配置の割合が増加すれば局所メモリ参照の割合は増大するから，この機構は局所性を高める．

前述のように，計算の還元モデルによって併行に実行できるタスクを容易に他のプロセッサへ移送するように生成できる利点が得られる．実際に，“(計算の) 張る木 (spanning tree)”は生長するが，この木は単一の伸びたり縮んだりする式に対応する．したがって，実行中のプログラムの“出力”は連続的に取り出すことができる．各 Xputer の APPLY キューの省略時のサービス・モードは FIFO であり，この木を横型に生成するから，併行実行可能な節点に速く到達する．そこで，飽和中に余分のタスクを生成しないようにするには，パケット生成の速度を抑えるために Xputer を縦型優先にするように LIFO にスイッチする．この制約によってキューのオーバーフロー（の度合）は軽減できるし，ある種のデッド・ロックで発生するメモリの過大な割当てを低減できる（この手法は Burton と Sleep[31] も述べている．）飽和モードでは，通常併行的に引数の組を要求する作用素では引数を逐次的に要求するように変更しなければならないが，この変更はわれわれの還元モデルの実現では容易である．また，この実現で資源の過大割当ての可能性を低下させるプログラムのコントロール機構を導入することもできる[33]．

4.6 パケットの一般的な流れ

一つの Xputer の図式的構成を図 11 に示す．この図では，圧力のサンプリング関数がスイッチ層を通して圧力パケットと他のタイプのパケット (APPLY, FETCH, FORWARD) と混ざった形で送られると仮定している．この仮定は，ガーベージ・コレクションを除い

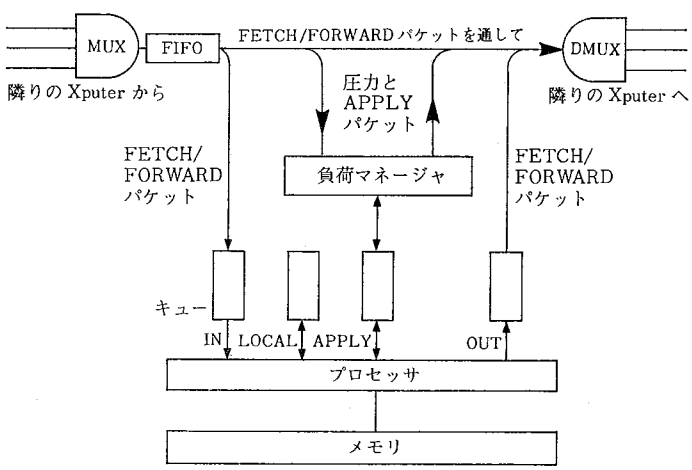


図 11 1台の Rediflow Xputer の中のパケットの流れ

Fig. 11 Packet flow within a Rediflow Xputer

てわれわれのシミュレーションの結果でも使っている．ガーベージ・コレクションについてはまだパケット・ベースでのシミュレーションをしていない．

5. 性能の評価

Rediflow アーキテクチャの性能を現在シミュレーションで評価している．目下同一の技術要素，アーキテクチャ，および評価モデルを仮定して単一プロセッサに対比した速度の向上を測定している．さらにモデルを改良して，現存の機械と対比した性能をシミュレーションで計測したいと思っている．

5.1 シミュレーションの一般的技法

Rediflow のシミュレータは埋め込んだ von Neumann 型プロセスを含むグラフ還元モデルの分散型インタープリタを実現している．スイッチ層に内在する遅延は，プロセッサの遅延に相対的にパラメタ化してある．Xputer に対して述べたバッファはすべてシミュレートしている．また，スイッチでの競争を解消するために各スイッチに一つずつの FIFO の入力バッファを入れてある．各グラフ還元ステップと基本作用素の計算には1単時間を要すると仮定し，単位時間は評価した平均値に設定してある．

実際には各 Xputer のメモリ内に各関数の常駐コードのコピーが存在すると仮定しており，まだコードの分配のシミュレーションは実施していない．コードのキャッシュ化を考えており，また異なるコード・ブロックの数は通常ブロックを例示する (instantiation) 回数に比べて少ないので，コードの分配のシミュレーションでは，たかだか小さい付加的定数だけ性能を変化させるだけであろう．

ガーベージ・コレクションは各 Xputer の中で併行的に（ストレージの）コンパクション (compaction) を用いて行う[34]．最終的にはメッセージをベースとする大域的コンパクション系でストレージ・コンパクションを行うつもりである．いまのところシミュレーションに使えるメモリ量が小さくコレクションには非現実的な時間がかかるので，まだ分散型の併行ガーベージ・コレクションをシミュレートしていない．

シミュレーションでは同期方式とイベント駆動 (event-driven) 方式を組み合せている．プロセッサとスイッチのサイクル（時間）はそれぞれの相対的速度に対応して設定できる比で定めている．シミュレーションでは通信に関与しない一定数の操作で書いた決定性のプログラムをほとんど使用し，各操作はほぼ一様の操作時間であるから，“on the fly” の速度向上を次の式で計算できる：

$$\frac{\text{主要操作の全所要時間}}{\text{シミュレートした時間}}$$

シミュレートした時間と異なり，主要操作は通信の遅延を含まないから，これには正しい負荷をかけているはずである．もし正しくなければ，測定した速度向上は過大であろう．

われわれのシミュレータを使うと，通信のオーバーヘッドを含まない無限に多くのプロセッサの併行性も計測できる．シミュレータでは併行性を “ply” の概念を使って定義している．ply 数が1のときには一つのパケットが存在し，このパケットは最初に要求した操作に対応する．ply n が与えられたとき，ply $n+1$ は ply n のパケットが生成するすべてのパケットの合併集合である．定義によって，ply 内のすべてのパケットは併行に実行できる．

したがって ply，内の併行性はその中のパケット数によって定義できる．併行性の平均計測値はある応用で到達可能な速度向上の粗い上限を定めている．したがって，この値は

多重プロセッサを上手に利用するための必要条件を与える．現在のシミュレーションで使っている Xputer ネットワークは長方形の格子 (grid) であり，$n=100$ 程度の小数の節点に対しては，この相互連絡図式の最悪の場合の遅延 sqrt(n) は最小の遅延を示す構成での値 log(n) と大きく違わないから，この図式で十分である．

シリコン・ウエハー (silicon wafer) 同志の相互連絡では異なった構成を採用するとしても，単一のウエハー上で節点の部分集合を実現しようとするときには，おそらく同様にこの格子構成を使用できよう．

5.2 ベンチ・マーク

われわれは2種類のベンチ・マークを走らせてみた．一つは純粋に分割統治で定義される関数や比較的独立したプロセスのような単純な種類のアクティビティとなるトイ・プログラムを集めたものである．部分的には，これらを使うことによって，パラメタや負荷バランスの機構を微調整し，システムの有効性を証明できた．たとえば，n 個の Xputer が一つのプログラムを均等にネットワーク内で n 個の独立したプロセスへ広げるようにするにはかなりの労力が必要であった．同様に，たとえば $2n$ 個の独立したプロセスが逐次的に生成されるときに，ネットワークではじめの n 個を走らせ，一つのプロセスが完了して空きが生れたときに $n+1$ 番目のプロセスを入れるようにする，等である．

さらに，トイ・プログラムの例として（前出の）分割統治で定義した階乗関数がある．図12の上のカーブは平方格子構成で Xputer の個数を変化させたときの 2^{10} の階乗の計算における速度の向上を示している．

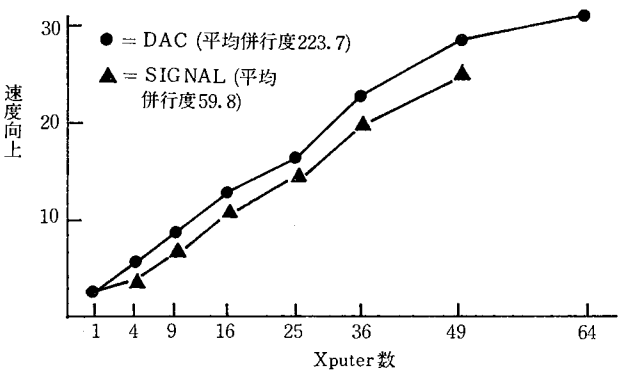


図 12 二つの応用プログラムでの速度向上

Fig. 12 Speed up in two applications programs

もう一つのクラスのベンチ・マークは，単純なデータベース探索やトランザクションの更新や信号処理の分野の多数のアクティビティを組み合せた現実的な応用からなる．図12の下のカーブはこのような一つの応用，すなわち二つの複素数値のデータのストリームの移動ウインドウ相関を求める信号処理問題における速度の向上を示している．一つのウインドウについての計算では，一つの信号の現在の n 個の値と他方の信号の現在の n 個の値の共役複素数の重み付きの内積を求める．図のランでは $n=20$ とし，50個のウインドウを併行に計算した．パラメタを変えたランでも同様の結果が得られた．シミュレーションでは，さらにデータと APPLY パケットについて Xputer 間のメッセージ距離の分布を計算して局所性の効果を定量的に求めるのに役立った．

図13は階乗と信号処理の例におけるそれらの平均距離を示している．プロセッサ当たり 10 μs の還元操作と 40 Mbps のスイッチ・スループットを仮定しているが，この仮定

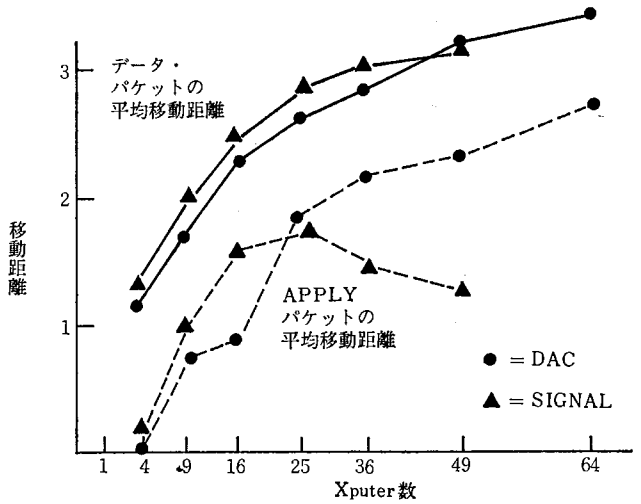


図 13 二つの応用におけるパケットの移動距離

Fig. 13 Packet transmission distance in two applications

値は，ほぼ通常のスイッチをもつマイクロプロセッサの性能に相当する．上述の例での局所性をみると，一つのパケットがたどる平均距離は最悪の場合の距離より，かなり小さいことがわかる．移送可能関数と局所関数を区別したときには，データ参照の5ないし20パーセントが一般に Xputer に局所的でないことがわかる．

6. おわりに

6.1 今後の作業

資源の制約から，すべての問題を撤底的に考えることはできなかったので，評価モデルから負荷分配技法，性能の測定，および言語の問題へ目を向けてきた．

われわれは評価モデル，とくに現在のグラフ還元評価系によるメモリの消費をさらに最適化しようとしている．シミュレータのメモリの制約から調べたい問題のサイズは限られていた．そこで，応用の中の併行性の度合が非常に高いときに限って現在の Rediflow 評価系は，(これはインタープリタであるが) 逐次型プロセッサと競合できると考えられる．もし，このインタープリタをコンパイル・コードに置き換えれば，良好な改善が得られるかもしれない．

基本の Rediflow システムの評価や改良に加えて，応用領域も広げている．たとえば，目下論理プログラムの評価についての研究をしている[36,37]．また，信頼性の研究も行うつもりである．

関数評価の基礎になる数学的モデルに，データがアクセスできなくなるまではデータを破壊しないようにする機能を付け加えた．そこで，このようなモデルは回復 (recovery) の機械化を表現するのに向いていると考えられる[12,38]．この可能性と円滑な機能低下 (graceful degradation) を支援する構成の方が望ましいというわれわれの論点によって Rediflow は信頼性の研究にも魅力的な系となっている．

6.2 関連する仕事について

Rediflow は Utah 大学での作用型多重処理システム (Applicative Multiprocessing System) についての初期の仕事から生れた．Rediflow は，そのトポロジーと負荷バランス法へのアプローチで AMPS と相異している．Rediflow では階層的アプローチを棄てて，システム

の各物理節点で有効に仕事が行えるような方式を選んだ．また von Neumann 型プロセスの保持 (retention) も新しく，強度に逐次処理に向いた部分プログラムに対しては von Neumann 型アーキテクチャが依然，最も速い実行モデルであるというわれわれの考え方を反映している．最後に，Rediflow のストレージ配置と再利用 (reclamation) 技法は AMPS の技法とは，まったく異なっている．

多数の研究者が Rediflow のゴールと同様のゴールを追究している．たとえば，Alice の提案はおそらく評価モデルの点で最もよく似ている．Alice はグラフ還元を使っているが，細かい細分の局所操作と粗い細分の移送可能操作とを区別していないので，局所性を高める利点を利用していない．第2の相異点はグラフ還元中の節点のリンク機構にある[20]．Hewitt の Apiary ネットワーク[40]も，また Rediflow と同様のゴールと機械化を行っていると思われる．Rediflow では，またもっと細かく細分した MIT の“結合 (connection)”機械[41]と同様にシステム全体にわたるデータ構造を設定するのにポインタを利用している．

データフロー機械も同様に研究されているが[16,17,42]，その研究では Rediflow よりも細かい細分単位の併行性を利用しようとしている．これについては以下の2点が観察される：

1) Rediflow 内の Xputer の中のパケットは併行して，あるいは順序に独立した仕方で処理できる．データフロー・アーキテクチャは多分1個の Xputer の内部性能を向上できるだろう．逆に，われわれが述べた負荷バランスの概念を使うと，データフロー向きのシステムの中に有効な第2レベルの仕事の実現手段を与えるだろう．
2) データ構造を扱う必要から，いくつかのデータフローの研究では還元モデルと要求駆動実行が有効であると認めているようにみえる[43,44]．還元の概念は I-構造[45]の概念の基礎ともなっているが，I-構造は本質的に要求駆動型の組ないし配列である[46]．

言語に関する仕事の中では，Shapiroの Bagel[47] のようにPROLOG関係のアプローチがある．Bagel 用のConcurrent PROLOG言語[48]は，おおざっぱにいえば単一化 (unification pattern matching) とガード付表現 (guarded expression) の形の不定の素関数 (primitives) をもつ関数型言語の機能をもっている．Shapiro はプログラマが陽に関数を systolic 配列* のように実行するサイトを指定すべきであるといっているが，このような指定は Rediflow ではオプションとなっている．しかし，シミュレーション結果からはこのような指定は不要であると考える．PROLOG全体を支援しようとする別の提案もテスト中である[50]．これらの提案はさらに複雑な逐次実行モデルを含んでいるので，広範囲の問題に有効か否か，あるいはより汎用のプロセッサの中で“バック・エンド (back-end)”に位下げになるかどうかはまだわからない．

最後に，言語主導でない汎用多重プロセッサの研究もされている[51]．その性能評価結果は有意義な情報であり，最終的には多重プロセッサの利用を単純化する言語主導のアプローチとの比較におけるベンチ・マークとして役立つに違いない．

Rediflow 多重プロセッサ系ではパケット・スイッチング・ネットワークを採用して高水準のプログラムの抽象化を実現している．その応用領域は中位の細分あるいは関数レベルの併行性を利用して，本質的には分散システムとして負荷のバランスをとる．予備的な性能評価の結果によれば，Rediflow のアプローチは有望である．

（技術企画部　山田真市　訳）

* (訳注) Carnegie-Mellon 大学で開発されている行列算法用アーキテクチャである．たとえば，H. T. Kung, “Why Systolic Architectures?” COMPUTER, Vol. 15, No. 1, January 1982, pp. 37-46 を参照．

参考文献 [1] D.E. Muller, W.S. Bartky, "A Theory of Asynchronous Circuits", *Proc. Int'l Symp. Theory Switching*, 1959, pp. 204-243.

[2] R.M. Karp, R.E. Miller, "Properties of a Model for Parallel Computations: Determinacy, Termination, Queueing", *SIAM. J Appl. Math*, Vol. 14, No. 6, Nov. 1966, pp. 1390-1411.

[3] R.M. Kap, R.E. Miller, "Parallel Program Schemata", *J. Computing and Systems Sciences*, Vol. 3, No. 2, May 1969, pp. 147-195.

[4] D.A. Adams, *A Computation Model With Data Flow sequencing*, Stanford University, Computer Science Dept., tech. report CS117, 1968.

[5] S. Patil, "Closure Properties of Interconnections of Determinate Systems", *Proc. Project MAC Conf. Concurrent Systems and Parallel Computation*, June 1970,pp. 107-116.

[6] R.M. Keller, "A Fundamental Theorem of Asynchronous Parallel Computation", T-y, Feng, ed., *Parallel Processing*, Springer-Verlag, New York, 1975, pp. 102-112.

[7] W.H. Burge, *Recursive Programming Techniques*, Addison-Wesley, Reading, Mass., 1975.

[8] P. Henderson, *Functional Programming*, Prentice-Hall, Englewood Cliffs, N. J., 1980.

[9] D.A. Turner, "The Semantic Elegance of Applicative Languages", *Functional Programming Languages and Computer Architecture*, Oct. 1981, pp. 85-93.

[10] G. Kahn, "The Semantics of a Simple Language for Parallel Programming", *Information Processing*, IFIP, North Holland, 1974, pp. 471-475.

[11] R.M. Keller, "Denotational Models for Parallel Programs with Indeterminate Operators", *Formal Description of Programming Language Concepts*, E. Newhold, ed., Elsevier, North-Holland, 1978, pp. 337-366.

[12] R.M. Keller, G. Lindstrom, *Toward Function-Based Distributed Database Systems*, University of Utah Computer Science Dept., tech. report UUCS-82-100, Jan. 1982.

[13] M. Conway, "A Multiprocessor System Design", *AFIPS Conference Proc.*, 1963, pp. 139-148.

[14] G.R. Andrews, F.B. Schneider, "Concepts and Notations for Concurrent Programming", *Computing Surveys*, Vol. 15, No. 1, Mar. 1983, pp. 3-44.

[15] D.M. Ritchie, K. Thompson, "The Unix Time-Sharing System", *Comm. ACM*, Vol. 17, No. 7, July 1974, pp. 365-381.

[16] J.B. Dennis, "Data Flow Supercomputers", *Computer*, Vol. 13, No. 11, Nov. 1980, pp. 48-56.

[17] I. Watson, J. Gurd, "A Practical Data Flow Computer", *Computer*, Vol. 15, No.2, Feb. 1982, pp. 51-57.

[18] D.P. Friedman, D.S., Wise, "CONS Should Not Evaluate Its Arguments", Michaelson and Milner, eds., *Automata, Languages, and Programming*, Edinburgh University Press, Edinburgh, Scotland, 1976, pp. 257-284.

[19] C.P. Wadsworth, *Semanics and Pragmatics of the Lambda-Calculus*, Oxford University, PhD thesis, 1971.

[20] R.M. Keller, G. Lindstrom, S. Patil, "A Loosely-Coupled Applicative Multi-Processing System", *AFIPS Conf. Proc.* Vol. 24, June 1979, pp. 613-622.

[21] A.L. Davis, R.M. Keller, "Dataflow Program Graphs", *Computer*, Vol. 15, No. 2, Feb. 1982, pp. 26-41.

[22] Arvind, K.P. Gostelow, "The U-interpreter", *Computer*, Vol. 15, No. 2, Feb. 1982, pp. 42-49.

[23] J. Tanaka, *Optimized Concurrent Execution of an Applicative Language*, University of Utah, PhD thesis, Mar. 1984.

[24] R. Keller, F.C.H. Lin, J. Tanaka, "Rediflow Multiprocessing", *Proc. Compcon Spring 84*, Feb. 1984, pp. 410-417.

[25] M. DuBois, F.A. Briggs, "Effects of Cache Coherency in Multiprocessors", *IEEE Trans. Computers*, C-31, No. 11, Nov. 1982, pp. 1083-1099.

[26] C.V. Ravishankar, J.R. Goodman, "Cache Implementation for Multiple Microprocessors", *Proc. Compcon Spring 83*, Mar. 1983, pp. 346-350.

[27] G.C. Fox, "Concurrent Processing for Scientific Calculations", *Proc. Compcon Spring 84*, Feb. 1984, pp. 70-73.

[28] E. G. Coffman, P. J. Denning, *Operating Systems Theory*, Prentice-Hall, Englewood Cliffs, N. J., 1973.

[29] A. Gottlieb, et al., "The NYU Ultracomputer—Designing an MIMD Shared Memory Parallel Computer", *IEEE Trans. Computers*, C-32, No. 2, Feb. 1983, pp. 175-189.

[30] I. Baron, et al., "Transputer Does 10 or More MIPS Even When Not Used in Parallel," *Electronics*, Nov. 1983, pp. 109-115.

[31] F. W. Burton, M. R. Sleep, "Executing Functional Programs on a Virtual Tree of Processors", *Proc. ACM Symp. Functinal Programming Languages and Computer Architecture*, Oct. 1981, pp. 187-195.

[32] P. C. Treleaven, D. R. Brownbridge, R. P. Hopkins, "Data-Driven and Demand-Driven Computer Architecture". *Computing Surveys*, Vol. 14, No. 1, Mar. 1982, pp. 93-143.

[33] R. M. Keller, G. Lindstrom, "Applications of a Feedback in Functional Programming", *Conf. Functional Languages and Computer Architecture*, Oct. 1981, pp. 123-130.

[34] H. G. Baker, Jr., "List Processing in Real Time on a Serial Computer", *Comm. ACM*, Vol. 21, No. 4, Apr. 1978, pp. 280-293.

[35] P. Hudak, R. M. Keller, "Garbage Collection and Task Deletion in Distributed Applicative Processing Systems", *Proc. ACM Symp. Lisp and Functional Programming*, 1982, pp. 168-178.

[36] G. Lindstrom, P. Panangaden, "Stream-Based Execution of Logic Programs", *Proc. Int'l Symp. Logic Programming*, Feb. 1984, pp. 168-176.

[37] U. S. Reddy, "Transforming Logic Programs into Functional Programs", *Proc. Int'l Symp. Logic Programming*, Feb. 1984, pp. 187-196.

[38] F. C. H. Lin, "A Distributed Load Balancing Mechanism for Applicative Systems", Department of Computer Science, University of Utah, PhD thesis proposal, Dec. 1983.

[39] J. Darlington, M. Reeve, "A Multi-Processor Reduction Machine for the Parallel Evaluation of Applicative Languages", *Symp. Functional Programming Languages and Computer Architecture*, Oct. 1981, pp. 65-77.

[40] C. Hewitt, H. Lieberman, "Design Issues in Parallel Architectures for Artificial Intelligence", *Proc. Compcon Spring 84*, Feb. 1984, pp. 418-423.

[41] W. D. Hillis, *The Connection Machine*, Massachusetts Institute of Technology AI Laboratory, tech. report 646, Sept. 1981.

[42] K. P. Gostelow, R. E. Thomas, "Performance of a Simulated Dataflow Computer", *IEEE Trans. Computers*, C-29, No. 10, Oct. 1980, pp. 905-919.

[43] E. A. Ashcroft, unpublished presentation on a Lucid machine, Lawrence Livermore National Laboratories, Oct. 1983.

[44] I. Watson, unpublished paper on dataflow research. Lawrence Livermore National Laboratories, Oct. 1983.

[45] Arvind, R. E. Thomas, *I-structures: An Efficient Data Type for Functional Languages*, MIT Laboratory for Computer Science, tech. report MIT-LCS-TM-178, Sept. 1980.

[46] R. M. Keller, "Divide and CONCer : Data Structuring for Applicative Multiprocessing", *Proc. Lisp Conference*, Aug. 1980, pp.196-202.

[47] E. Y. Shapiro, Presentation on Bagel and Concurrent Prolog, *ACM Symp. Prin Programming Languages*, Jan. 1984.

[48] E. Y. Shapiro, *A Subset of Concurrent Prolog and Its Interpreter*, Institute for New Generation Computer Technology, tech. report TR-003, Jan. 1983.

[49] E. W. Dijkstra, "Guarded Commands, Non-Determinacy, and a Calculus for the Derivation of Programs", *Comm. ACM*, Vol. 18, No. 8, Aug. 1975, pp. 453-457.

[50] D. deGroot, ed., *IEEE Int'l Logic Programming Symp.*, 1984, entire publication.

[51] E. F. Gehringer, A. K. Jones, Z. Z. Segall, "The Cm* Testbed", *Computer*, Vol. 15, No. 10, Oct. 1982, pp. 40-49.

執筆者紹介 Robert M. Keller

Washington 大学において B.S. および M.S.E.E を，また California 大学 Berkeley 校において Ph.D. を取得．1970 年から 1976 年まで Princeton 大学で電気工学の助教授を務め，また Stanford 大学と Lawrence Livemore 国立研究所の客員教授も兼務する．IEEE Transaction on Computers と Journal of Parallel and Distributed Computing の編集者でもある．

併行処理，並列コンピュータ・アーキテクチャおよび機能言語の具体化の研究に貢献し，理在は，高水準言語の多重プロセッサによるインプリメンテーションに関する多くのトピックス，とくに還元規則とデータフローの計算モデルの使用法に関心をもつ．現在 Utah 大学の電子計算機科学の教授である．

Frank C. H. Lin

1951 年台湾生まれ．1973 年国立台湾大学において電気工学で B.S. を，また 1978 年に Utah 州立大学において電気工学で M.S. を取得．

1976 年から 1977 年まで Cal-Comp Electronics 社のエンジニアであった．1978 年 Sperry 社に入社，現在の研究対象はコンピュータ・アーキテクチャ，ネットワーキング，応用システムおよび耐障害形計算である．

報告 3次元家モデルに基づく住宅設計一貫システム

Total Housing CAD System

篠田博水

要約 HCAD (Housing CAD System) は，従来の住宅 CAD システムが製図作業の省力化を目的とした自動製図システムであったのに対して，住宅のもつ情報を忠実に表現した家モデルを，それも3次元でデータベースにもち，自動製図（平面図，立面図，外観透視図，各種伏図，設備図，建具表）や見積りから，部材拾い，工場やメーカへの発注までを行う一貫した住宅 CAD システムである．

データベースに3次元モデルを実現することにより，今後の新しい機能の追加はもちろんのこと，将来住宅 CAD システムに要望されるであろう住宅シミュレーションにも対応できると確信する．

Abstract The former housing CAD system was an automatic drafting system developed for the purposes of reducing drafting manpower. However, HCAD is a total housing CAD system, it is possible automatic drafting, estimation, development of housing parts and ordering from a factory and maker by realizing 3 dimentional housing model on data base.

HCAD will be able to dramatically reduce manpower in the drafting, estimation, development of housing parts and ordering, and prevent anything mistakes that a human causes.

The realizing 3 dimentional housing model on data base enables to extend new functions and provide for future use such as housing simulation.

1. はじめに

従来の住宅 CAD システムの大半は，簡単な間取図を入力して平面図と立面図を自動的に描かせる自動製図システムであり，システム内にもつ情報も製図に必要な図形情報に限定されていた．また，図形情報も平面図と立面図だけであれば2次元で十分である．なかには，この図形情報をもとに，必要な情報を入力して見積りをさせることのできるシステムもある．

しかしながら，現在では住宅 CAD システムを製図作業の省力化だけではなく，営業戦略の道具として用いようとしている．すなわち顧客に対してタイムリに図面（それも外観透視図を含む図面）や正確な見積りを提出することにより，顧客の目を他メーカに向けさせないようにする，あるいは入力されている情報をもとに自動的に部材の展開を行って，工場や部材メーカへの発注までを行う一貫システムが望まれている．

日本ユニバック(株)では，こうした要望を実現するために，コンピュータ内に家のもつ情報を忠実に表現した家モデルを3次元でもち，この家モデルをもとにした住宅設計一貫システム HCAD (Housing CAD System) を開発した．

本稿では，HCAD の概要と家モデルの構造について述べる．

2. システムの概要

本システム（図1参照）は，設計から施工までを包含する一貫した住宅 CAD システムである．営業担当者と顧客との折衝の結果描かれたラフスケッチをもとに，住宅の外形，屋根，部屋内部(床，壁，天井)，収納，建具，造作，備品，設備，基礎等を「家構成デー

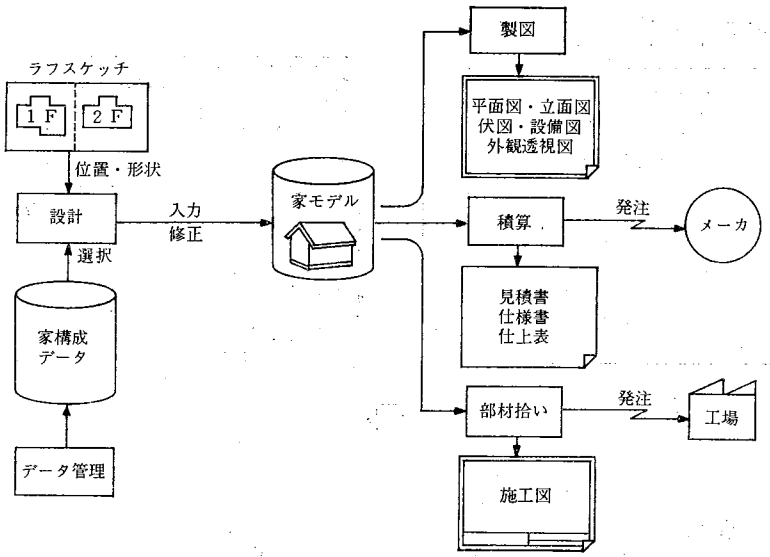


図 1 システムの概要

Fig. 1 Outline of HCAD system

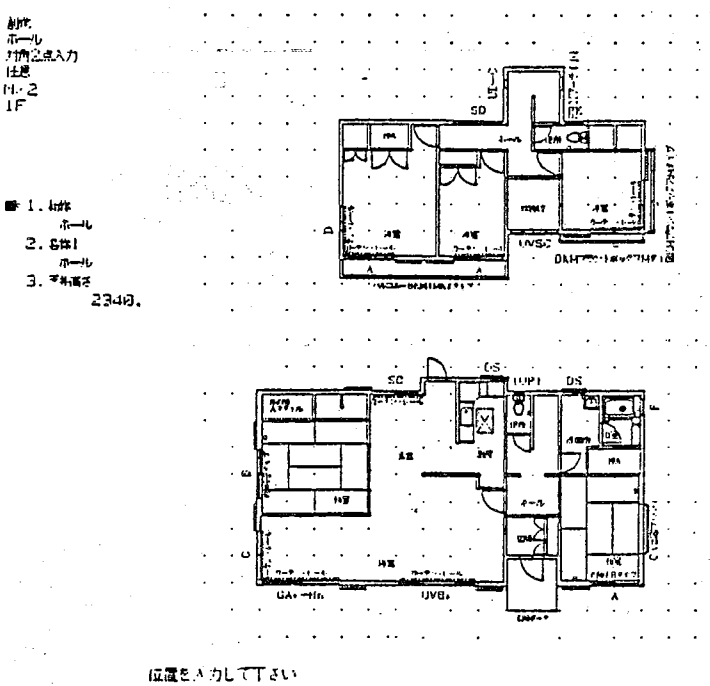


図 2 入力画面

Fig. 2 Display screen

タ」より選び，その形状あるいは配置される位置を与えてコンピュータ内に「3次元家モデル」を作る．

「3次元家モデル」が作られると，自動的に製図処理，積算処理が行われ，各種図面（平面図，立面図，外観透視図）と見積書，仕様書，仕上表が出力される．入力画面を図2に，出力図面を図3に示す．

その後，何度かの設計変更により「3次元家モデル」は更新される．契約が成立する

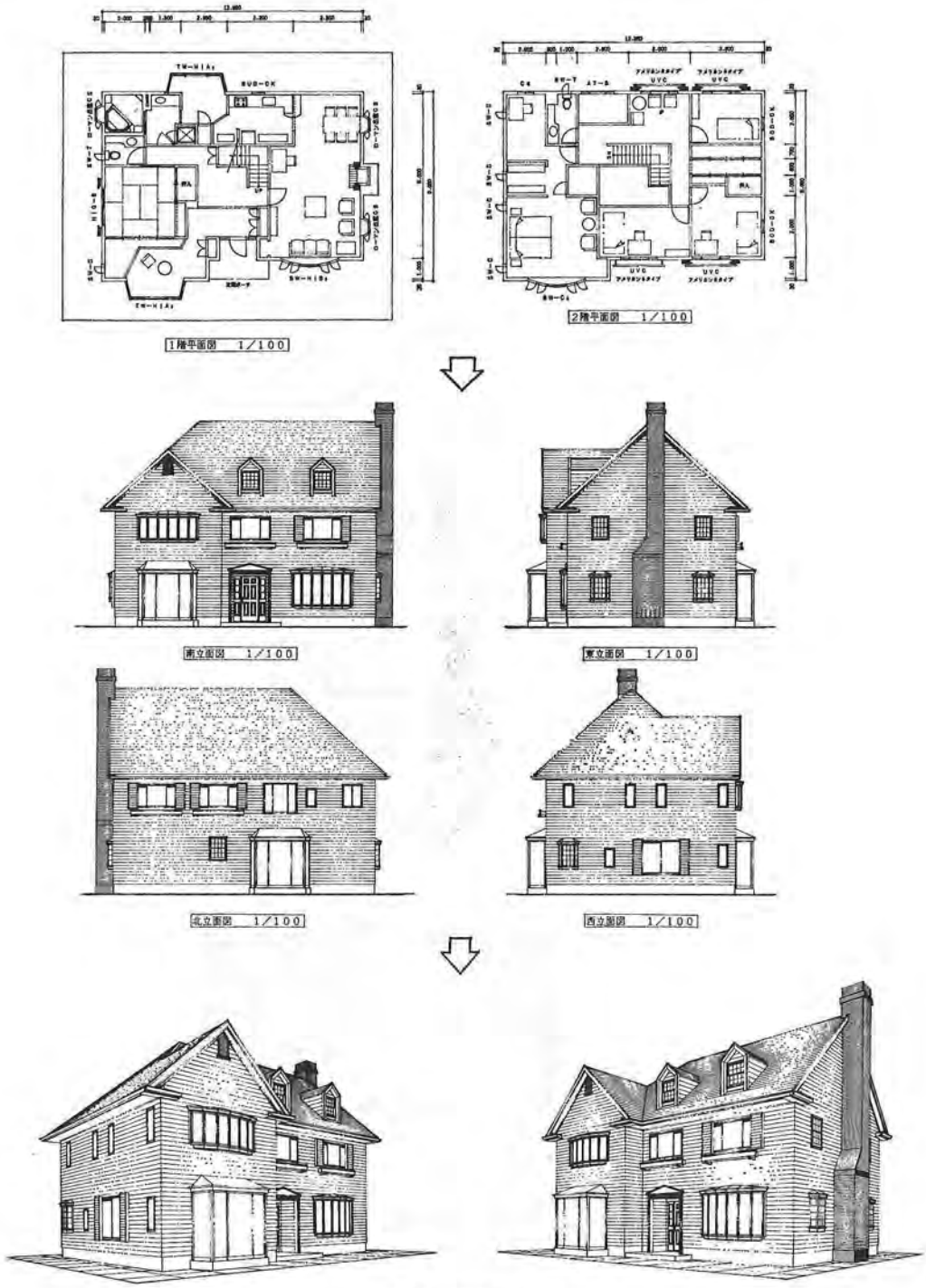


図 3 出力図面

Fig. 3 Output drawings

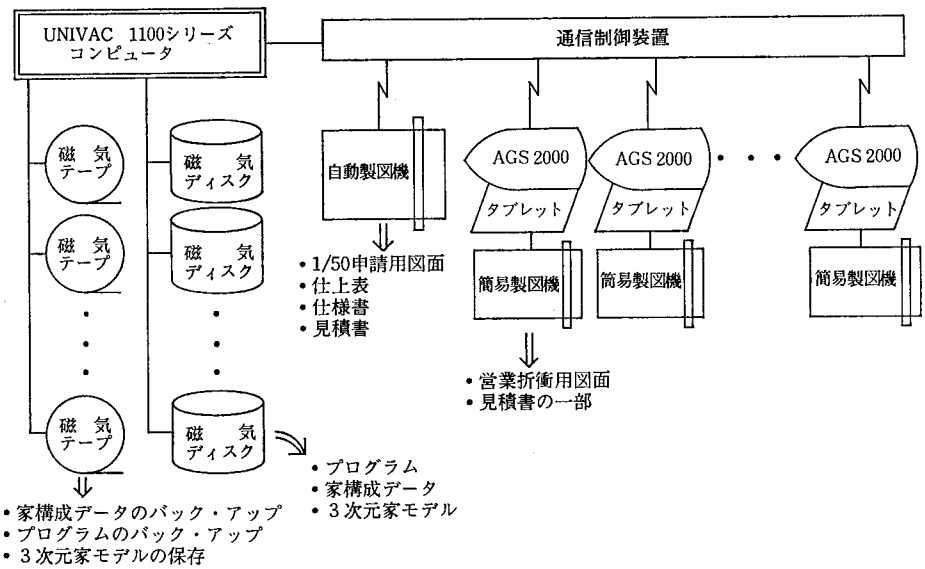


図 4 ハードウェアの構成

Fig. 4 Hardware configuration

と，最終図面として各種伏図と設備図が出力され，同時に部材展開により工場やメーカへ発注がなされ，施工図も出力される．

本システムのもう一つの特徴は，入出力端末装置を営業所に設置してここから入力ができ，かつその場で出力を得ることができることである．図4に，そのハードウェアの構成を示す．

3. 家構成データ

家を構成する各種データは，あらかじめ家構成データとして用意されている．表1に家構成データの区分を，図5に家構成データのデータ型を示す．

{非形状データ} は，各家構成データの特性をもち，次の情報からなる．

1) 名称
2) メーカ名，型番，色
3) 入力情報
4) 形状情報（長さ，幅，間口，奥行等）
5) 部位創成情報（たとえば，床，壁，天井の仕上下地と高さ）
6) 見積り情報（単価，数量単位，丸め）
7) 部材情報（部材の自動展開のための情報）
8) 基礎創成情報（基礎を自動創成するための情報）
9) 他の家構成データとの関連情報（たとえば，窓と雨戸，窓と窓手摺，窓と面格子，窓と内障子，部屋と内部建具等）
10) 付帯情報（子とする他の家構成データの情報．たとえば，便器は紙巻器，タオル掛，給水栓，汚水排水を子にもつ）
11) その他

{形状データ} は，各家構成データの形状をもつ．形状は，平面図・立面図・伏図・設

表 1 家構成データの区分

Table 1 Classification of housing component and material data

分類	家構成データ	分類	家構成データ
外形	外形	屋内	部屋
	袖壁		収納
	出窓		間仕切壁
	アルコーブ		腰壁
	外部建具(ドア)		垂れ壁
	〃 (窓)		目隠し壁
	外部建具付帯(雨戸)		壁ふかし
	〃 (面格子)		壁補強
	〃 (窓手摺)		上り床・下り床
	〃 (内障子)		床ゾーン
	〃 (その他)		床下補強
屋根	屋根		ボーダ天井
	軒先カット		下り天井
	葺降し		天井補強
	寄棟納り		造作
	屋根建具		タタミ
	屋根付帯		床の間
屋外	屋外造作		階段
	屋外付帯		備品
基礎	基礎		電気設備
	基礎付帯		給排水設備
躯体	断熱		ガス設備
	軸組		家具
	梁小屋組		内部建具
	床組	材料	内部建具面材
	柱		ガラス
			仕上下地材
		その他	接続線
			段差表示
			切断線

備図に出力される2次元形状と，外観透視図に出力される3次元形状がある．

ただし，外形，屋根，部屋，収納，屋外造作，基礎のようにあらかじめ形状が決まっていないものは｛形状データ｝をもたない．「3次元家モデル」を作るときに，入力された領域の座標値と｛非形状データ｝の部位創成情報により形状が作られる．図6に部屋（便所）と備品（便器）の家構成データの例を示す．

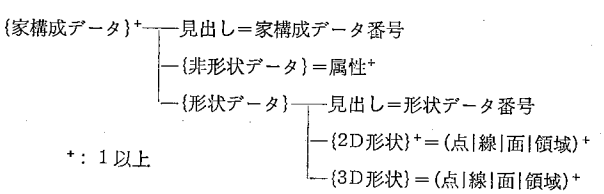


図 5 家構成データのデータ型

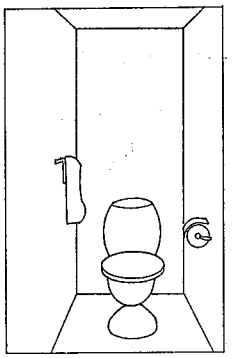
Fig. 5 Data type of housing component and material data

4. 3次元家モデル

3次元家モデルは，一棟分の家を構成する各種の家構成データを分類し，論理的に構造化したものである．操作によって選ばれた家構成データは，ラフスケッチから形状あるいは配置される位置が付加され3次元家モデルに入れられる．図7に3次元家モデルの基本概念を示す．

部屋(便所)

属性名	属性値
名称	便所
入力情報	対角2点
床高さ	0
天井高さ	2300
床仕上下地	長尺ビニール, サネ付合板
壁仕上下地	クロス, 石コウボード
天井仕上下地	クロス, 石コウボード
•	
•	
•	



備品(便器)

属性名	属性値
名称	洋風便器
メーカ	×××
型番	C 44 SCL + DT 811
入力情報	補1点, 壁付, 床付
形状データ番号	
単価	
付帯情報	紙巻器, タオル掛, 給水栓, 汚水排水
•	
•	
•	

備品(紙巻器)	給排水設備(給水栓)
備品(タオル掛)	給排水設備(汚水排水)

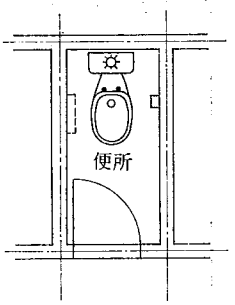


図 6 家構成データ例

Fig. 6 Example of housing component and material data

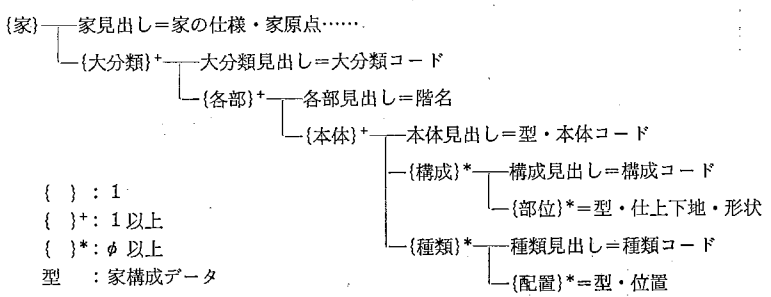


図 7 3次元家モデルの基礎概念

Fig. 7 Basic concept of 3 dimentional housing model

{大分類} は, 家構成データを {基礎}, {外形}, {屋根}, {屋内}, {躯体}, {屋外} に分類した各集合である. {各部} は, {大分類} の中の階単位の集合である. {本体} は, {各部} の中で以下に属するデータをあるまとまった単位で処理する必要があるために設けられた集合である. また {本体} は, 家構成データとして入力する最大の単位でもある. たとえば, 家構成データから「部屋」を選び, その部屋の領域を入力すると, {部屋本体} 以下のデータが自動的に作られる. {構成}, {本体} の中で入力された家構成データのうち, 形状をもたなくて入力された座標値と家構成データにもたれている部位創成情報によって形状が自動的に作られるものに対して, 特性が同じものの集合である. {部屋本体} であれば {床}, {壁}, {天井}, {立上り} が各 {構成} である. {部位} は, {構成} でまとめられるデータであり, 形状の他に仕上下地を属性にもつ. {種類} は, {本体} の中で入力された家構成データのうち, 形状をもつものに対して, 同じ種類のものの集合である. {部屋本体} であれば, {備品}, {造作}, {設備} が各 {種類} である.

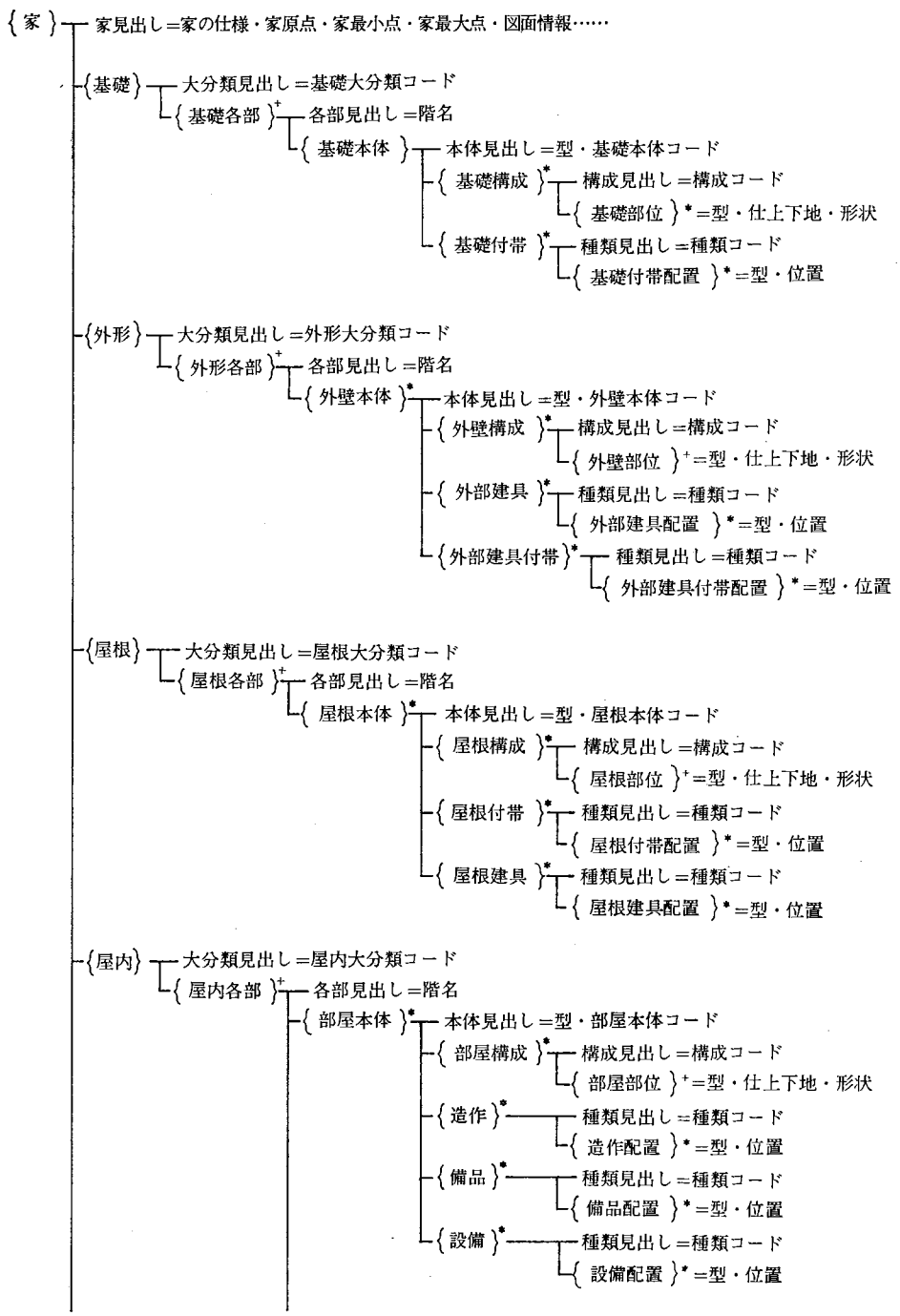
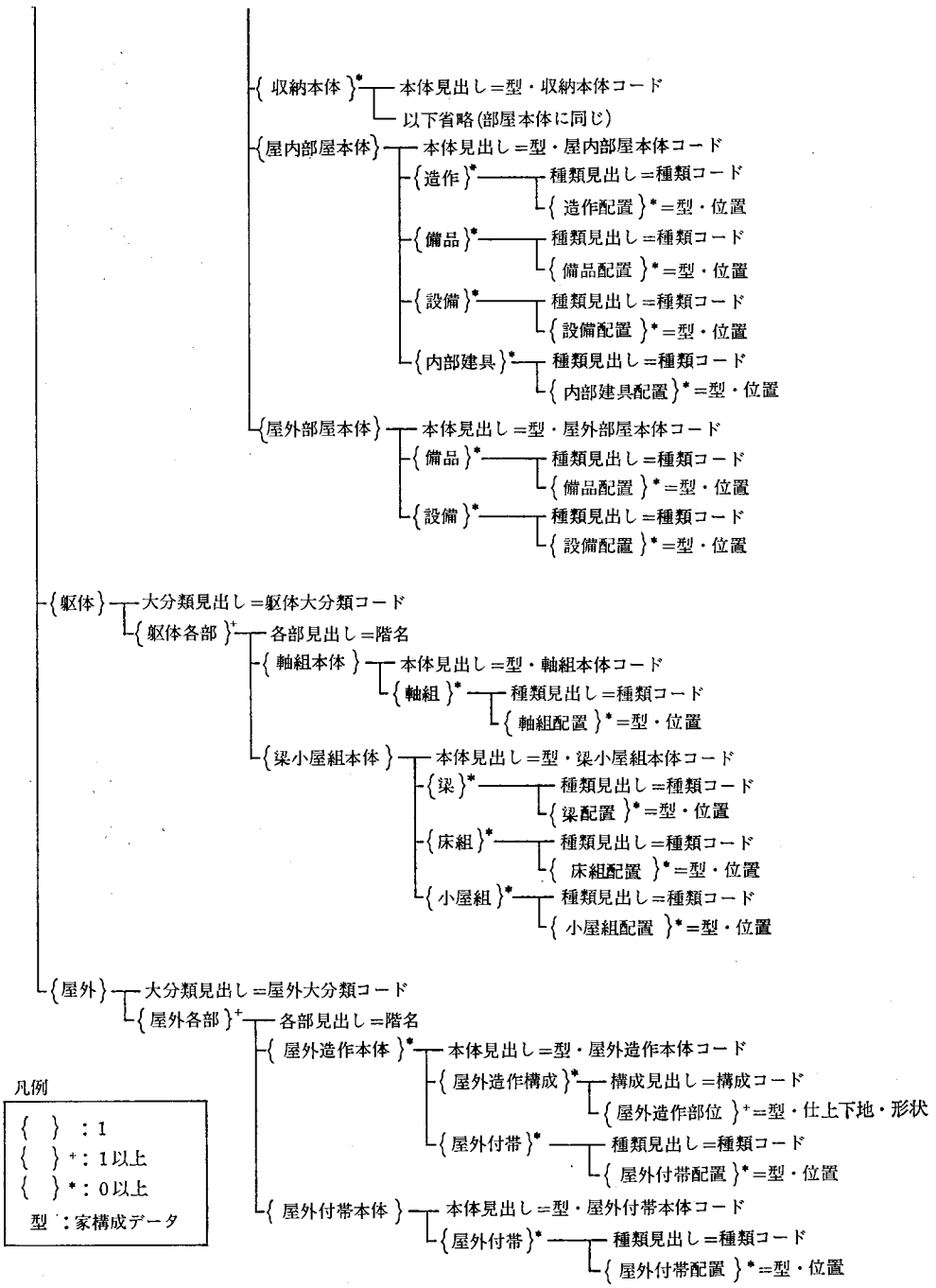


図 8　3次元家

Fig. 8　Data type of



モデルのデータ型

3 dimentional housing model

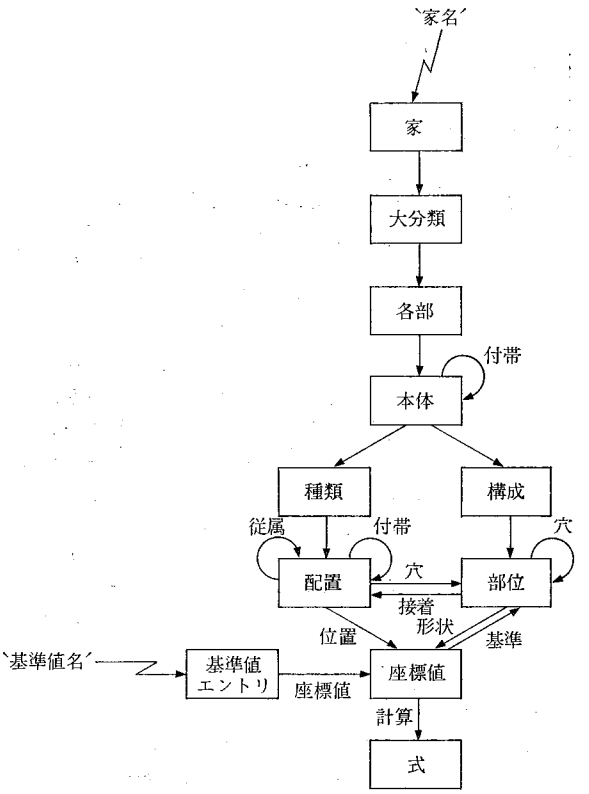


図 9　3次元家モデルの内部構造

Fig. 9　Data structure of 3 dimentional housing model

{配置} は，形状データをもつ家構成データ一つ一つを表現したものである．

以上のように，3次元家モデルに入力された家構成データは，形状があらかじめ決まっているものは {配置} として，形状が自由に作れるものは {部位} として表現され，そして {本体} で集合させられ，その {本体} の中でその特性により {種類} あるいは {構成} でまとめられる．{部位} と {配置} がどの {本体} に属するかは，物理的な位置関係あるいは家構成データの属性である「本体従属コード」によって決まる．ただし，中にはどの {本体} に属するかが一意に決められているものもある．図8に3次元家モデルのデータ型を，図9にその内部構造を示す．また，図6に示した部屋（便所）と備品（便器）の家構成データが入力された場合の3次元家モデルのデータ型を図10に，その内部構造を図11に示す．

3次元家モデルは，関係データ・モデルとして表現され，図9の四角い箱はエンティティを表し，⟶は関係を，¯乙¸はエンティティに名前がつけられていることを表す．関係名の書かれていない⟶は，単なる2項の親子関係である．

付帯（本体，本体）あるいは付帯（配置，配置）は，家構成データの付帯情報に設定された子となる他の家構成データとの関係を表す．

従属（配置，配置）は，第1項の配置が削除されたとき一緒に削除され，変更されたとき一緒に変更される関係を表し，次のエンティティがこの関係をもつ．

従属（窓，雨戸）

従属（窓，窓手摺）

従属（窓，面格子）
従属（窓，内障子）
従属（電灯，スイッチ）
従属（インターフォン，チャイム）

接着（部位，配置）は，部位を削除したとき一緒に削除され，部位を移動したとき一緒に移動する関係を表し，配置の家構成データの「部位従属コード」にこの関係をつけるかどうかを設定する．図12に接着関係の例を示す．

穴（部位，部位）あるいは穴（部位，配置）は，部位に対する穴あけを関係で表現しておき，設計変更を容易に効率良く行うためのものである．図13の穴関係の例において，出窓を削除する設計変更が起きた場合，{出窓本体} 以下を削除することで，穴（外壁D，穴部位）の関係も削除され，外壁Dは元の形状に戻る．

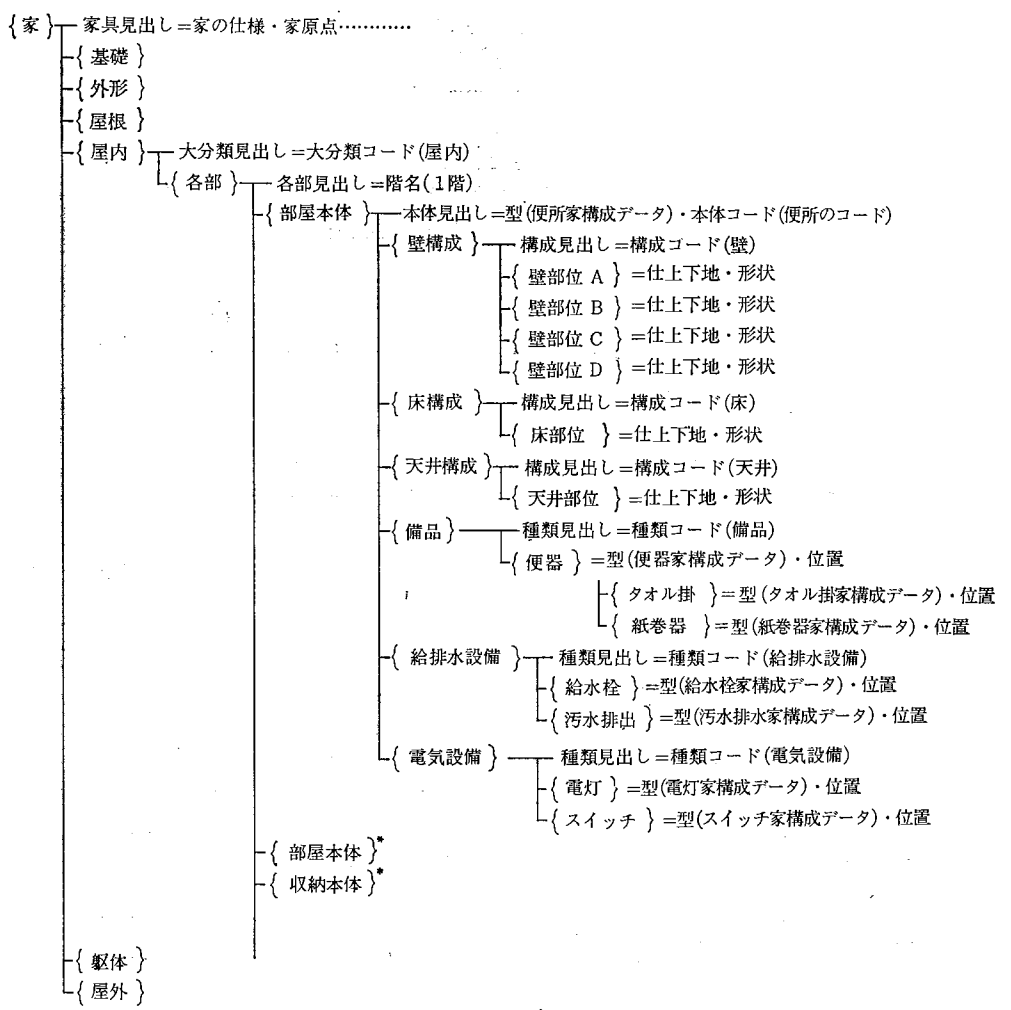


図 10 部屋のデータ型例

Fig. 10 Example of room data type

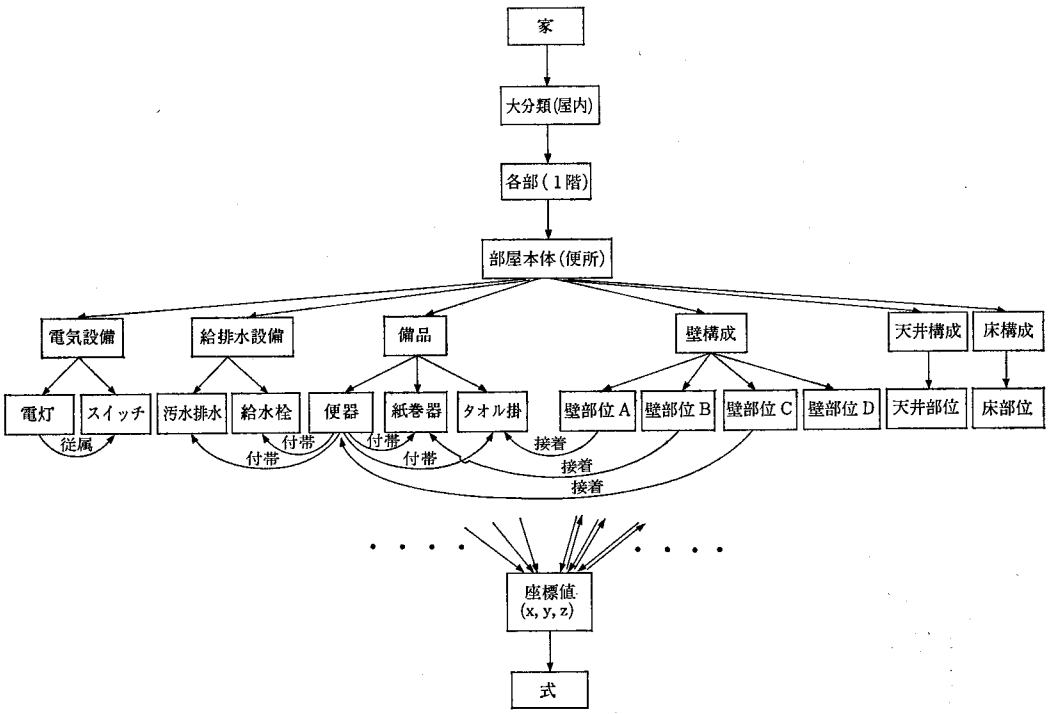


図 11　部屋の内部構造例

Fig. 11　Example of data structure

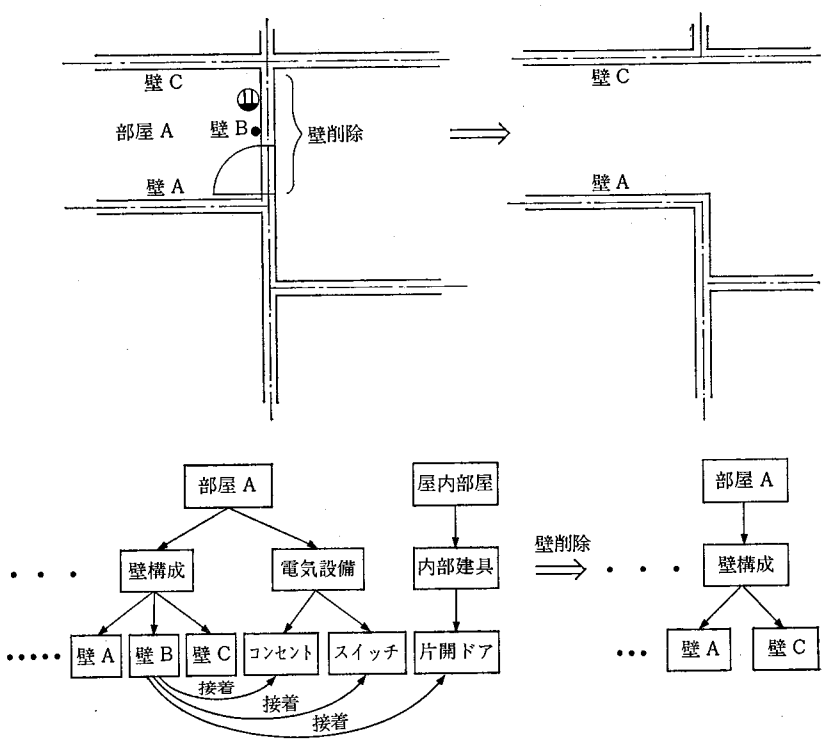


図 12　「接着」関係の例

Fig. 12　Example of adhesive relation

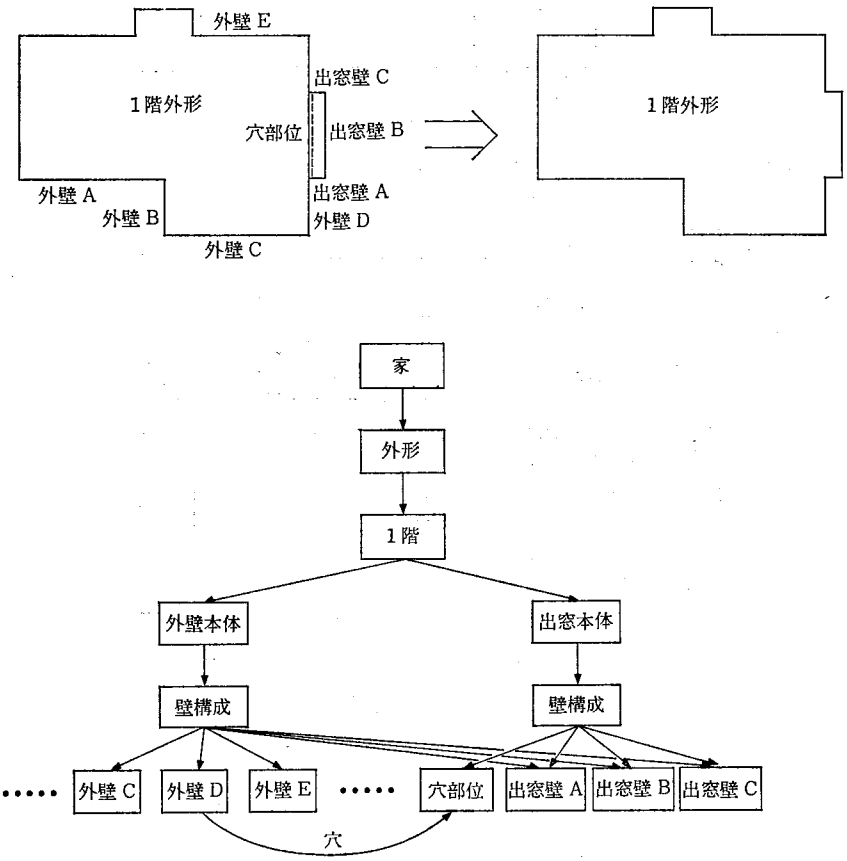


図 13 「穴」関係の例

Fig. 13 Example of「Hole」relation

表 2 {配置} の「位置」関係

Table 2 Positional relation of disposition

配 置	X	Y	Z
窓	X_1	Y_3	Z_3
	X_1	Y_1	Z_3
机	X_1	Y_4	Z_2
	X_1	Y_2	Z_2

表 3 {外壁部位} の形状関係

Table 3 Positional relation of outer wall

部 位	X	Y	Z
外壁①	X_1	Y_4	Z_1
	X_1	Y_3	Z_1
	X_1	Y_2	Z_5
	X_1	Y_4	Z_5
外壁②	X_1	Y_2	Z_1
	X_2	Y_2	Z_1
	X_2	Y_2	Z_5
	X_1	Y_2	Z_5
・ ・ ・	・ ・ ・	・ ・ ・	・ ・ ・
外壁⑥	X_4	Y_4	Z_1
	X_1	Y_4	Z_1
	X_1	Y_4	Z_5
	X_4	Y_4	Z_5

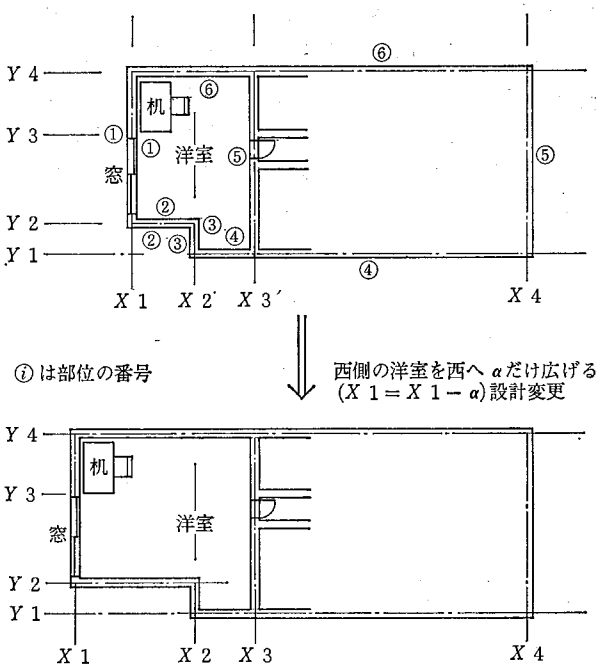


図 14 形状変更例

Fig. 14 Example of positional relation

表 4 洋室の {部位} の「形状」関係

Table 4 Positional relation of inner wall, floor and ceiling

部 位	X	Y	Z	部 位	X	Y	Z	部 位	X	Y	Z
壁 ①	X_1	Y_3	Z_2	壁 ④	X_3	Y_1	Z_2	床	X_1	Y_4	Z_3
	X_1	Y_4	Z_2		X_2	Y_1	Z_2		X_1	Y_2	Z_3
	X_1	Y_4	Z_4		X_2	Y_1	Z_4		X_2	Y_2	Z_3
	X_1	Y_2	Z_4		X_3	Y_1	Z_4		X_2	Y_1	Z_3
									X_3	Y_1	Z_3
壁 ②	X_2	Y_2	Z_2	壁 ⑤	X_3	Y_4	Z_2		X_3	Y_4	Z_3
	X_1	Y_2	Z_2		X_3	Y_1	Z_2				
	X_1	Y_2	Z_4		X_3	Y_1	Z_4	天 井	X_3	Y_4	Z_4
	X_2	Y_2	Z_4		X_3	Y_4	Z_4		X_3	Y_1	Z_4
									X_2	Y_1	Z_4
壁 ③	X_2	Y_1	Z_2	壁 ⑥	X_1	Y_4	Z_3		X_2	Y_2	Z_4
	X_2	Y_3	Z_2		X_3	Y_4	Z_3		X_1	Y_2	Z_4
	X_2	Y_2	Z_4		X_3	Y_4	Z_4		X_1	Y_4	Z_4
	X_2	Y_1	Z_4		X_1	Y_4	Z_4				

位置（配置，X_i, Y_i, Z_i）あるいは形状（部位，X_i, Y_i, Z_i）は，{配置} の配置位置と {部位} の形状の座標値を {配置} と {部位} の属性から独立させ，家の形状変更を簡単な操作で効率よく実現するために用意された関係である．図 14 の形状変更例において，各 {配置} の位置関係を表 2 に，{外壁部位} の形状関係を表 3 に，洋室の各 {部位} の形状関係を表 4 に示す．この形状変更の操作は，移動すべき壁を指示した座標値 X_1 を {座標値} より探し，その値から α を減じるだけで行われる．また Z 座標値は高さを表し，各 Z 座標値は，各階ごとに設定されている複数の基準高さからの追い寸法で表現されている．これを表している関係が，

計算（基準高さ，式，高さ），あるいは，計算（高さ，式，高さ）

であり，図14の各 Z 座標値の計算関係を表5に示す．したがって，基準高さや各高さの寸法が変更になった場合は，「計算」の関係によって高さを再計算するだけでよい．

このように3次元家モデルは，家のもつ情報を論理的に構造化しただけではなく，各種の形状変更や全体的なあるいは部分的な仕様変更（たとえば仕上下地や色等の変更）に効率よく対処できるような構造をもっている．

表5 高さの「計算」関係

Table 5 Positional relation of height

基準高さ	式	高さ
下基準高さ	外壁下端	Z_1
	床高さ	Z_2
	窓取付高さ	Z_3
	天井高さ	Z_4
上基準高さ	外壁上端	Z_5

5. 操　作

本システムへの入力は，入力操作に先立って設定される「家の仕様」とタブレット上のメニュを押すことによって家構成データを選び，その形状あるいは配置される位置をタブレット上のラフスケッチから与えることによってなされる．操作者には，専任オペレータ，設計者あるいは営業担当者が考えられることから，操作は簡便であることと少ない入力で短時間に一棟分の家モデルが完成できることに重点を置いた．現時点で，1か月以上の経験をもつ操作者であれば，一棟分の家モデルを入力するために要する時間は30から40分である．

以下に図6の家構成データを例に，操作方法について述べる．

① メニュの**便所**を押すことによって，「部屋（便所）」の家構成データを選ぶ．

② 便所の領域を家構成データの入力情報（対角2点入力）に従って入力する．対角2点の位置が入力されると図10に示すように {部屋本体}，{壁構成}，{床構成}，{天井構成}，{壁部位} 4件，{床部位} そして {天井部位} が3次元家モデルに創成される．{部位} の仕上下地は，「部屋（便所）」の家構成データあるいは「家の仕様」に設定されている仕上下地がとられ，形状は対角2点の位置からできる矩形と家構成データで与えられる床高さと天井高さにより3次元で表される．

③ メニュの**洋風便器**を押すことによって「備品（便器）」の家構成データを選ぶ．

④ 便器の配置位置を家構成データの入力情報（補1点入力）に従って入力する．入力された位置は入力情報の「壁付」により最も近い壁に丸められ，図10に示すように {備品} と {便器} が3次元家モデルに創成される．{便器} の位置は，丸められた位置と家構成データに設定されている「床付」により3次元で表される．そして付帯情報に設定されている紙巻器，タオル掛，給水栓，汚水排水とその位置により，{紙巻器}，{タオル掛} {給排水設備}，{給水栓}，{汚水排水} が自動的に3次元家モデルに創成される．

このように本システムでは家構成データの「部位創成情報」と「付帯情報」によりシステムが自動創成を行うことによって，少ない入力で家モデルが完成できるように工夫され

ている．また，雨戸，窓手摺，面格子，内障子の外部建具付帯は，外部建具付帯が配置される場所の窓によって決まり，内部建具のうち部屋取合建具は，配置される場所の両側の部屋とその仕上げによって決まる．そのため，これらの家構成データは「他の家構成データとの関連情報」をもち，この情報によって自動決定される．この他に「基礎」も，すでに家モデルに創成されている家構成データの「基礎創成情報」をもとに自動創成される．

6. おわりに

コンピュータ内に実際の家に忠実な3次元家モデルを実現することにより，外観透視図を含む豊富な図面の出力や正確な見積りはもちろんのこと，メーカへの発注，部材展開による工場への部材の発注，そして施工図面の出力まで含めた一貫した住宅 CAD システムを構築することができる．

このシステムの採用により，従来2日かかっていた製図作業と見積り作業（外観透視図まで含めると4日）が約2時間（入力操作は30〜40分）ででき，そのうえ部材の拾い作業や発注作業が大幅に省力化できるとともに，部材の拾いミスや発注ミスも防止できる．また施工図面の出力で，現場での施工ミスも削減することができる．同時に，端末装置を営業所に設置することにより，顧客の前で顧客の要望にそって設計を進められること，そしてその場で平面図・立面図・外観透視図・見積書を出力できることが，顧客に満足感を与え営業効果を高めることになる．

今後は，3次元家モデルに敷地・環境・外構等を追加することと，「室内パース」の開発を行っていく予定である．

最後に，本システムの開発に当たり，御指導と御協力をいただいた，積水ハウス(株)の今井一延氏および日本ユニバック(株)の柳生孝昭氏に深く感謝します．

参考文献 [1] 古川康一，“コンピュータグラフィックにおけるデータ構造の問題”，情報処理，Vol. 11 No.9, 1970.
[2] 積鷹良介，「データベース要論」共立出版，1978.
[3] 穂坂衛「コンピュータグラフィックス」産業図書，1974.
[4] 山口富士夫，「図形処理工学」日刊工業新聞，1983.
[5] 柳生孝昭，“CAD のための data metamodel”，技報，日本ユニバック(株)，No. 6, FEB. 1984.
[6] 平林 繁，“陰線消去プログラム HLE”，技報，日本ユニバック(株)，No 6, FEB. 1984.

執筆者紹介 篠田 博水 (Hiromi Shinoda)
昭和22年生，45年法政大学工学部卒業．同年日本ユニバック(株)入社．各種 CAD システム開発を経て，現在(株)ソフトエクセルに出向し，ハウジング CAD システム開発に従事．

報告 ラスタ図形処理ワークステーション

A Raster Graphics Workstation

R. G. Bond, R. C. Dawson

要 約 現在の Sperry 社製の端末装置は，すべて文字発生機構と文字テーブルによってテキスト・データを生成する．表示する文字は，適当な属性ビットを付加して RAM に格納されており，特定のハードウェアによって相応の文字テーブル・データとして画面に表示される．このことは，一度に1種類の書体しか表示できないということを意味している．

その図形表示の機能にも制約がある．たとえばユーザが別の画面用記憶に図形データを描き，このデータと文字発生機構から取り出したテキスト・データとを重ね合わせて画面に表示する場合である．

しかし，現在の技術によって，これらの限界を解決するいくつかの新しいアプローチが試みられるようになった．VLSI 設計技術によって製造される高密度 RAMと一層強力なチップによって，これらの新しいアプローチの費用効果がさらに増大する．

本稿では，複数書体と図形とを一様にサポートできる一層強力なハードウェアでそれらを置き換えることを述べる．

Abstract The current Sperry product line terminals all generate text data with a character generator and a character lookup table. The characters to be displayed are stored with the appropriate attributes in a RAM and specialized hardware scans the appropriate character table data onto the screen. This means that only one character font can be displayed at a time.

Its graphics options are also limited. The user draws graphical data in a separate screen memory, which then "ored" with textual data from character generator and is displayed on the screen.

Current technology, however, has provided some new approach to solve these limitations. Denser RAMs and more powerful chips fabricated by very large scale integration design techniques make these new approach more cost effective. This paper describes the replacemern with one more powerful piece of hardware capable of uniformly supporting multiple fonts and graphics.

1. はじめに

1.1 現行システムの限界

現在の Sperry 社製端末装置は，すべて文字発生機構と文字テーブルによってテキスト・データを生成する．表示する文字は適当な属性ビットを付加して RAM に格納されており，通常 ASCII 符号で，特殊なハードウェアによって取り出された文字テーブル・データが画面に表示される．このことは，一度に1種類の書体（文字テーブルからロードされた）しか表示できないということを意味している．

現在使用されている図形処理機能にも限界がある．ユーザは別の画面用記憶領域に線や円といった図形データを描く．このデータと文字発生機構から生成されるテキスト・データとが重ね合わされ，画面に表示される．この方法の主な難点は，文字データが画面上の決まったセルに拘束され，ユーザが線をテキストに合わせるのがむずかしいということである．ユーザは，文字を図形モードで自由に描くことはできるが，この方法は時間がかかりすぎ，複雑で文字発生機構の目的をなさない．

画面のピクセル密度が増すと，問題はより深刻になる．ユーザは，画像を設計するための大きな自由度を要求するが，端末装置の入力手段がキーボードであるかぎり増大する要求への対応は困難である．キーボードと文字発生機構の限界によって，ユーザは画面をタイプライタ，つまり画面とは左から右へと水平方向に文字が配列された行が上から下に詰まっているもの，と考えざるをえなくなっている．

1.2 新しいアプローチ

しかし，現在の技術によってこれらの問題を解決するいくつかの新しいアプローチが可能になってきた．VLSI 設計技術によって製造される高密度 RAM と一層強力なチップによって，これらの新しいアプローチの費用効果がさらに増大する．

一つのアプローチは，文字発生機構，文字テーブル，図形処理機能を複数書体および図形を同時にサポートする，より強力なハードウェアに置き換えることである．画面はビットマップ形表示と呼ばれる独立したピクセル（画素）の長方形の配列として構成される．表示領域内の各ピクセルは，RAM 中の 1 ビットあるいはそれ以上に写像される．この形式の表示装置は，固定の文字境界がなく，つまりテキストは図形の別の形態というだけなので，より柔軟性が増すことになる．

ビットマップ形表示装置では，画面はラスタと呼ばれる，ピクセルの任意長のブロックとして構成される．セット，クリヤ，コピーおよび転送といったラスタに対する演算は，画像を作る画面指示語として使用される．ラスタのコピー演算を実行すると，文字が表示され，書体テーブルの文字表現はコピーの送り出し側として使用される．ビットマップ形表示を使った新しい方法は，1970 年代に，ほとんど Xerox 社 Palo Alto 研究所で始まった活動から出現した．最も好評な新しい入力機器の一つはマウスである．マウスは，カーソルを制御する機器として使用される小形の，片手で操作できる機器である．他のシステムでは，マウス操作と同じ機能を得るために図形処理用タブレット，トラックボールおよびジョイスティックを使用している．マウスは基本的には，ソフトウェアが現在位置の変化量を感知できる位置検出器を車輪の一つに取り付けた台車付きの箱である．マウスによって，システムとのより動的なインタフェースが確立できる．つまり，マウスの動きに追随してカーソルが動くため，ユーザはシステムに対して積極的な制御ができると感じるのである．

2. FROG のポートレイト

新しい設計アプローチのいくつかを評価するため，著者は FROG (Fast, Raster-Oriented Graphics project) と呼ばれる汎用ワークステーションを開発した．FROG は，マウスのついたビットマップ形ワークステーションである．

2.1 機 器 構 成

FROG のために特別に設計されたハードウェアは，ビットマップ形表示装置用インタフェースと付加図形処理プロセッサである．このハードウェアは，プログラム記憶用として 256 キロバイト RAM を装備した 68000 ベースのマイクロコンピュータ，5 メガバイト 130mm ディスク装置，非同期の 9600 ボー通信リンクおよびキーボードから成る．

マイクロプログラム化した図形処理プロセッサは，128 キロバイトのデュアルポート表示用 RAM (840×1152 ピクセル)，128 キロバイトのデュアルポート・スクラッチ RAM，AMD 29116 バイポーラ・プロセッサ 4K×48 ビットの制御記憶とマウス・インタフェースをもっている．図形処理プロセッサは，19 インチのモノクローム表示装置を動かす．

2.2 マ ウ ス

FROG との対話は，マウスを通して調整される．ユーザは，テキスト・カーソルを制御してマウスを動かし，マウス・ボタンを押すことによって，通常キーボードをたたいて選択する機能選択を行う．

マウス・ボタンを押すことによって，ある動作を行わせたいということをシステムに知らせる．遂行される動作は，装置のその時点での状態に依存するが，マウスの使用を一層一貫したものとする総合ガイドラインがある．マウス・ボタン1は，通常，ある操作の開始を意味する．また，これはいくつかの条件の一つを選択するときにも使われる．マウス・ボタン2は，進行中の操作が正常に完了したことを意味する．マウス・ボタン3は常にアボート（打ち切り）を意味する．現在の操作は停止し，装置はある既知状態に復帰する．

2.3 ウインドウ

ウインドウは，画面を互いに重ね合わせることのできる長方形の領域に区分する（図1）．すべてのウインドウは，ウインドウの名前を示す見出し，四つの境界および下線（アンダライン）で表現されるテキスト・カーソルをもつ．ウインドウは画面全体にでも，画面上のどの部分にでも置くことができる．各ウインドウは，すでにそのウインドウにあるテキストに影響を与えずにユーザが変更できる現在使用可能な書体をもっている．ユーザは自由に各ウインドウの中で，各種の方法でテキストと図形を混在させることができる．システムは，あるウインドウが別のウインドウの下にかくれてしまったり，上にかぶさっているウインドウが取り払われたりするにしたがって，ピクセル・イメージを自動的に格納したり再現したりする．

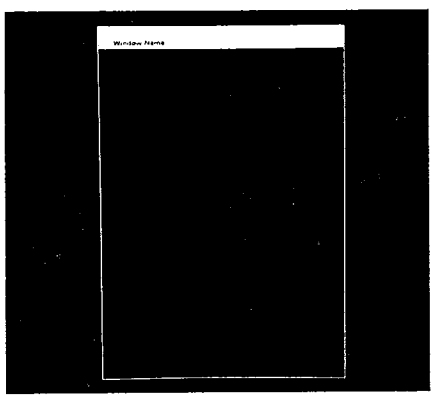


図 1 ウインドウ
Fig. 1 A window

2.4 操 作

主な設計目標の一つは，コマンドのキーインを排除することであった．コマンドのキーインは，ユーザが動作の選択をするときに現われる小さな「ポップアップ」ウインドウから多肢選択をすることに置き換えられている．たとえば，画面に何も表示されていないとき，ユーザはボタン1を押すことによって主システム・メニューを得る．主メニューが表示されたら，メニュー中の必要な項目の上にマウス・カーソルを置き，ボタン1を押す（図2）．ウインドウを新設したいとき，“Window Manager” を選択するとウインドウ管理や選択メニューが現われる．つぎに “Create a Window” を選択し，マウス・カーソルを使ってウインドウの場所と大きさを指定できる．

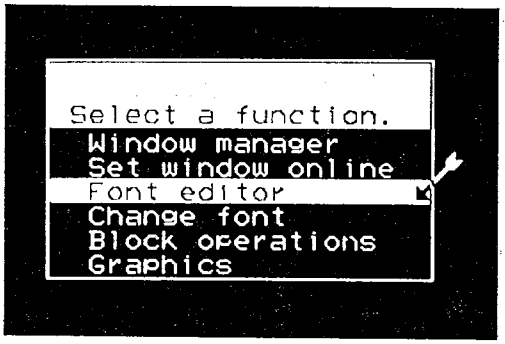


図 2　主システム・メニュー
Fig. 2　Main system menu

マウスと選択メニューを使って，ユーザはテキストをキーインせずにシステムに対して何でもすることができる．最上位のメニューには，ウインドウ管理，書体編集，書体割り当て，ブロック操作，および図形処理といった，ユーザが選択する各種任意選択機能が用意されている．最上位メニューによって，ユーザはどのウインドウをホスト・オンラインにするかを指定することもできる．どのウインドウもオンラインにすることが可能であって，ユーザからホスト・コンピュータへのリンクを可能にする．

2.4.1　ウインドウ管理 (window manager)

ウインドウ管理によって，ユーザは画面レイアウトを制御することが可能である．ウインドウ管理は，画面上のウインドウをデスク上に散らばった用紙と同じように扱う．どのウインドウも画面上の別の場所に動かすことができるし，別のウインドウの下に隠れているウインドウを一番上に引き出すこともできる．一番上のウインドウを画面上から取り除いたり，ディスクに格納したりすることができる．すでにディスク上に格納されているウインドウを画面上に戻すことができる．また，一番上のウインドウを再命名することが可能である（図3）．

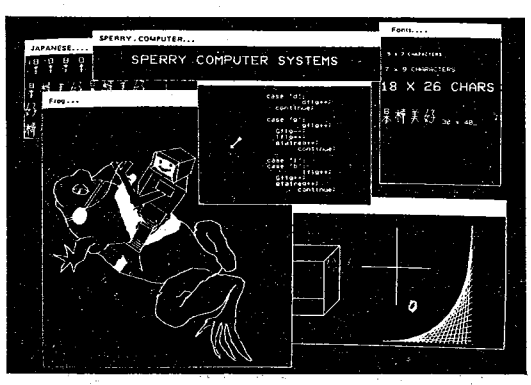


図 3　複数ウインドウ
Fig. 3　Multiple window

これらすべての操作は，ウインドウ管理メニューから行える．ユーザは，マウスとボタン1で操作を選択する．以後，ボタン1は再びウインドウを選択するときに使用される．選択されたウインドウでは，背景色は黒から白へ，文字は白から黒へと変わり，ユーザがボタン2を押すと，このウインドウで操作が行えるようになる．

2.4.2 書体編集 (font editor)

書体編集系（フォント・エディタ）は新しい書体を定義し，現存する書体で文字を編集するのに使用される．書体編集を選択すると，現行書体ウインドウ，ドット字形ウインドウ，状態ウインドウおよびコマンド・ウィンドウの四つのウインドウが現われる（図4）．

最上段のウインドウは，現行のフォントを示す．ユーザがマウス・カーソルをこのウインドウ内の文字に置いてボタン1を押すと，その文字は拡大されてドット字形ウインドウに現われる．

ドット字形ウインドウは，横方向24，縦方向32の四角形に分割された長方形の格子であって，各四角形は文字の一つのピクセル（ドット）を表している．ボタン1を押すことによって格子内の現行文字にピクセルをセットでき，ボタン2を押すことによってピクセルを消すことができる．文字に対して行った変更は，現行ドット字形ウインドウにそのまま現れる．

状態ウインドウは，書体編集セッションの現在の状態を示す．個々の文字は，状態ウインドウに現われるがままの高さと幅をもっている．状態ウインドウはまた，現行文字と現行書体の名前を示す．

4番目のウインドウは，コマンド・ウインドウである．コマンド・ウインドウ内の適当なコマンドを使って状態ウインドウ内のあらゆるものを変更できる．また，コマンド・ウインドウを通して，ディスクから新しい書体をロードしたり，現行書体をディスクに格納したりすることもできる．

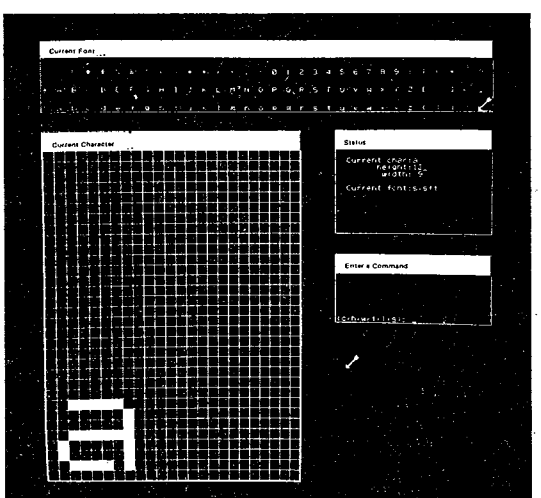


図 4 書体編集
Fig. 4 Font editor

2.4.3 書体割り当て (change font)

書体変更メニューによってディスクに格納した書体を選択ウインドウに表示して，必要とするものをマウスで選択することができる．この書体は，指定されたウインドウで再び変更するまで現行書体となる．すべてのウインドウは，システムで定められている初期状態の書体と同じ書体で始まる．

2.4.4 ブロック操作 (block operations)

まだ実現されていないが，汎用ラスタ・ベースの編集機能を用意されることになる．

2.4.5 図形処理 (graphics)

二つのタイプの図形処理機能，線引きとペイントが利用できる．線引き機能を選択した場合，ユーザはボタン1を押して線の始点を選定する．すると，“ラバーバンド線”が始点から現在のマウス・カソール位置まで延びる．この直線は非破壊で，マウス・カソールの動きに追随する．ボタン2を押すと，線が固定する．このモードをすべての線が完成するまで続け，ボタン3を押す．

図形処理メニューからペイント機能を選択すると，画面に複数の筆先をもつウインドウが現われる．ユーザは，その筆先ウインドウの中でマウス・カーソルを位置決めして筆先を選択し，ボタン1を押す．“筆”は，ウインドウの中でボタン1が押されている間ペイントを行う．ボタン2を押すと，筆先ウインドウから選択された“消ゴム”によってピクセルが消去される．

2.4.6 オンライン操作 (set window online)

FROG システムは，上述の端末側独自の操作に加えてホスト・コンピュータに接続された端末のように交信可能である．一つのウインドウがオンライン用ウインドウとして選択でき，オンライン・ウインドウにキーインされたテキストは通信回線を通して9600ボーでホスト・コンピュータに転送される．端末側から転送され再びホスト側から返送される文字とホスト側で生成した文字は，このウインドウに入れられる．端末エミュレーション・パッケージは，行単位の挿入および削除といった標準 Uniscope 編集機能のほとんどをサポートする．

これは，バイポーラ・プロセッサの速度が目立つ領域の一つであって，9600ボーに通信回線を維持するのに何の障害もない．ウインドウにある文字の全部を動かす文字挿入には0.5秒以下しか要さない．

3. ハードウェア

3.1 コスト・パフォーマンスと柔軟性

FROG プロジェクトのための設定された第1の目標は，ユーザが画面を変更でき，操作と個々の適用業務のインタフェースに個別対応のできる効率的でかつ柔軟性のあるシステムを確立することであった．

以前は，高密度ビットマップ形表示装置を含むシステムは，大容量の記憶装置が必要なため非常に高価であった．大容量記憶を扱うのに十分な能力をもつプロセッサも必要であった．多数の IC，大きな PC ボードのスペースとパッケージング・サイズはすべて高価格の原因であった．FROG プロジェクトの第2目標は，表示装置インタフェース，記憶，ロジックおよびプロセッサを1,000ドル以下の費用で製作することであった．FROG プロジェクトでは，PC ボードを卓上形マイクロコンピュータ内の拡張用ボードか，表示装置のパッケージ内のどちらにでも収容できるくらいに小型化することであった．これらの目標は，最大の部品数を IC 150 個以下にすることを意味した．

技術上の三つの進歩が上述の目標を実現可能にした．INMOS 社の高速“ニブル・モード”64 K×1ダイナミック・メモリを用いることによって，表示用記憶を32個の IC で構成することが可能になった．

既存のマイクロプログラム・シーケンサ（順序制御機構）2910 に Advanced Micro Device 社の新しいバイポーラ・マイクロプロセッサ29116を組み合わせることによって，FROG 図形処理プロセッサは約150個の IC で構成される，マイクロプログラム化された

柔軟性のある装置となった．また高速多機能の新しい TTL チップと Programmable Logic Array devices (PALS) も部品数を最少にするのに役立った．

3.2 ディスプレイ・インタフェース

図5は，表示装置インタフェースと図形処理用プロセッサのブロック図である．図の左側が68000マイクロコンピュータ・インタフェースで，右側が表示装置側である．マイクロプログラム・シーケンサ 2910，4K×48 ビットの書込み可能制御記憶 (WCS)，制御記憶バッファと29116からなる図形処理用プロセッサも示されている．

インタフェースの主な構成要素は，256キロバイトのダイナミック RAM である．記憶領域のうち128キロバイト足らずの部分が表示装置のためのビットマップを保持している．画面をリフレッシュし，ちらつきを防ぐために，毎秒60回ずつ記憶領域にアクセスして画面に転送しなければならない．図形処理プロセッサは，画像を更新するために記憶サイクルを遂行しなければならない．記憶インタフェースは，これらの記憶動作が効率的に生起するようにする．設計の鍵は，ニブル・モード記憶を使用したことである．

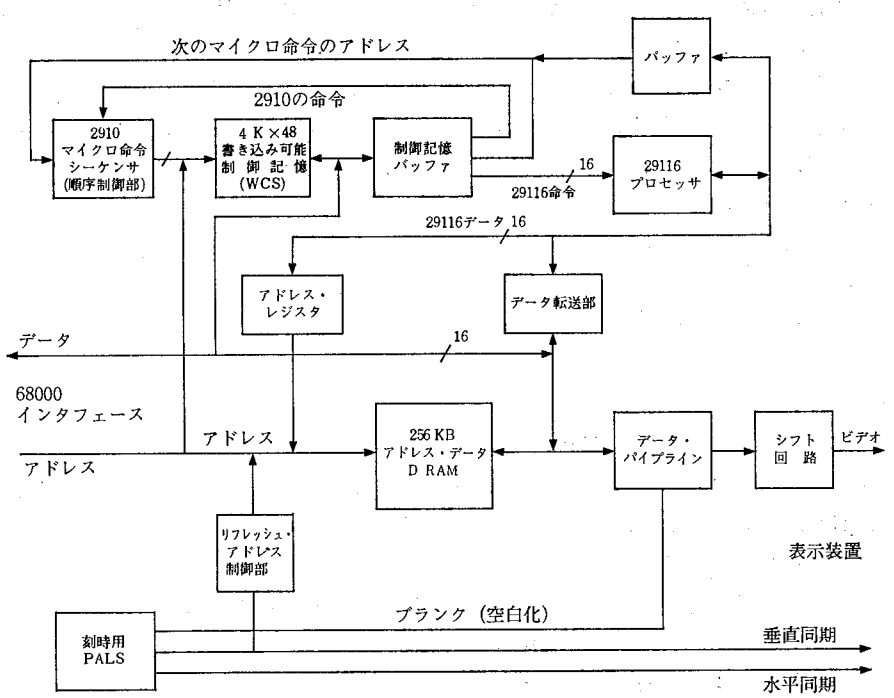


図 5 図形処理プロセッサと表示装置インタフェース

Fig. 5 Graphics processor and display interface

従来の64K×1ダイナミック RAM では，1記憶サイクルごとに行アドレス信号 (RAS) と列アドレス信号 (CAS) を与え，1ビットのデータを取り出す．ニブル・モード記憶でも RAS 信号と CAS 信号を与えるが，最初のデータ・ビットが取り出された後，CAS 信号がつぎつぎに更新され，後続の3ビットにアクセスする．したがって，各記憶サイクルごとに記憶場所から1ビット取り出すかわりに，全部で4ビット取り出すことができる．取り出すビット数が増えても，記憶サイクル時間が40ナノ秒増えるだけである．

記憶インタフェースは，つぎのように作動する．画面リフレッシュ・サイクル間隔内で，32の記憶チップから2ビットずつ取り出され，320ナノ秒の記憶サイクルで64ビットのデータが形成される．100 MHz のビデオ速度では，画面上において1ビット当たり

10ナノ秒あれば間に合い，したがって64ビットでは640ナノ秒で間に合う．このため，図形処理プロセッサの記憶サイクルを遂行するのに320ナノ秒与えられ，表示画面のリフレッシュと図形処理プロセッサとの間で時間幅を半分ずつ分け合うことになる．図形処理プロセッサは，実効記憶サイクル時間640ナノ秒で作動する．マイクロ命令のサイクル時間は160ナノ秒であって，1記憶サイクル当たり4命令を実行することができる．プロセッサ用のマイクロプログラムは，ほとんどデータを読み取り，操作し，記憶装置に書き込む．記憶装置との間で同期動作をとるには，8命令または1.28マイクロ秒を必要とする．アドレスを計算し，データを操作し，ループの終了条件をテストするために8命令が使用される．たとえば，図形処理プロセッサが線を描く速度は，ピクセル当たり1.28マイクロ秒である．ある時間幅に着目すると，残り時間が十分にあるためプロセッサがリフレッシュ・データに割り込むこともなく，高速演算を遂行することができる．

画面をリフレッシュするために記憶装置が常時アクセスされるので，ダイナミックRAM リフレッシュ・サイクルを実際に遂行する必要はない．“リフレッシュ”という語は，表示画面のリフレッシュを指し，ダイナミック RAM 自身を指すのではない．

リフレッシュ・サイクルの間に取り出された64ビットのデータは，高速データ・パイプラインにロードされ，最終的に表示装置（画面）に送られるビデオ信号を形成することになる．表示装置側から要求される別の信号は，水平および垂直同期信号である．これらはリトレース間隔内に画面消去のための空白化信号を生成する二つの PALS によって生成される．

68000マイクロコンピュータ制御システムもまた，図形処理プロセッサのコマンドを組み立て，ディスクとの間でデータを転送するためにダイナミック RAM にアクセスする．68000は，同期が探られ，図形処理プロセッサが記憶サイクルを遂行するのと同じ時間を使う．

68000は優先権をもっており，マイクロエンジン（図形処理プロセッサ）は，68000が処理を終了するのを待つだけである．しかし，このような待ちが発生しても，図形処理プロセッサがコマンドを実行する間に，68000が専用記憶装置の外部でコマンドを形成し実行するため，図形処理プロセッサの速度を低下させることはない．

3.3 図形処理プロセッサ

図形処理プロセッサは，単一レベルのパイプラインのマイクロプログラム化されたプロセッサである．設計の重点は，新しく導入された29116マイクロプロセッサと2910マイクロ命令順序制御部である．図6に29116のブロック図を示す．29116は，TTL コンパチブル I/O と内部 ECL ロジックをもつバイポーラ・プロセッサであって，電力消費は最大3.5ワットである．29116は，1クロック・サイクルで0から15ビットまでシフトできる完全な16ビットのシフト回路をもった最初の商用マイクロプロセッサで，高速でデータ操作をするため特別に設計された構造になっている．また，32語（1語は16ビット）のレジスタ・ファイル，16ビットのアキュムレータ，3オペランドの ALU，状態生成および検出回路，および16ビットのデータ入力レジスタをもつ．そして，制御記憶バッファから16ビットの命令を受け入れ，双方向16ビットのデータ・バス経由でデータを授受する．

2910は，4キロ語までの制御記憶用の記憶領域を制御することができるマイクロ命令順序制御機構であって，制御記憶バッファから4ビットの命令を受け取り，12ビットのアドレスを生成する．これは，内部プログラム・カウンタ，5レベルのサブルーチン戻りスタックとループ制御レジスタをもっており，マイクロプログラムの流れを制御するための16

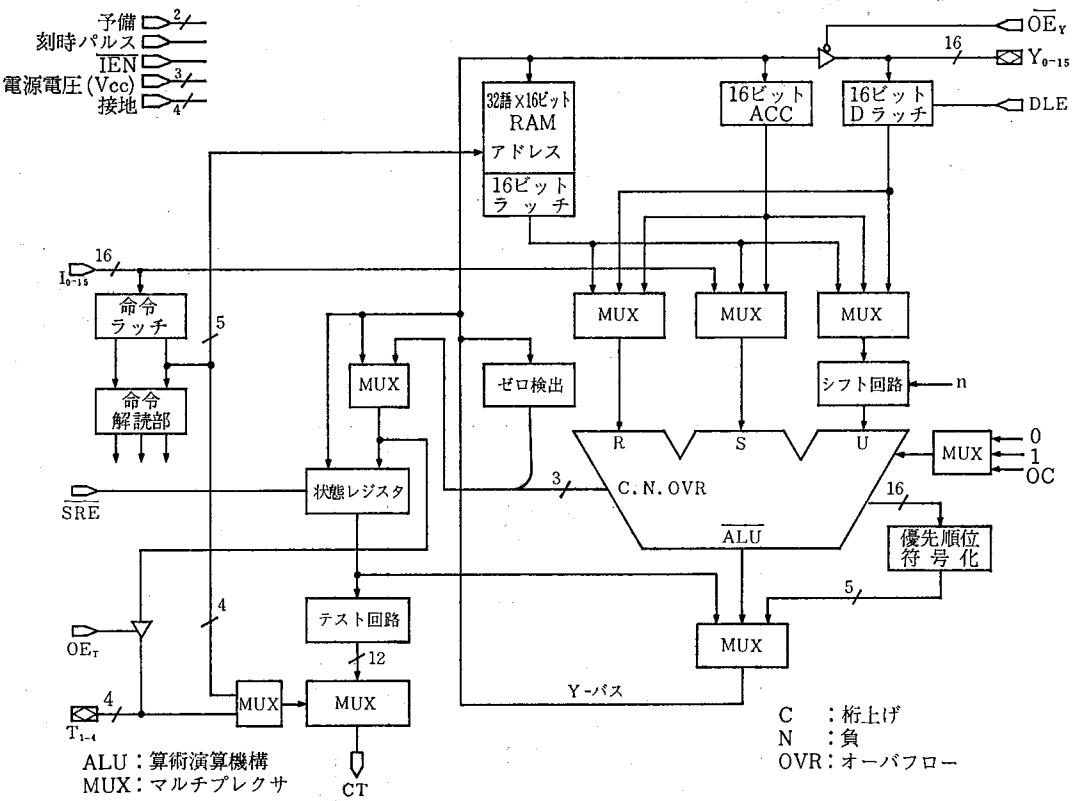


図 6 29116 ブロック図

Fig. 6 29116 block diagram

の異なった命令を実行する．次の命令アドレスの最下位4ビットは，29116側の出力によって動かされ，16とおりの分岐が遂行される．これは，コマンドの解読と他の条件付き分岐に非常に有用である．4キロ語の制御記憶は，68000システムによってロード可能になり，マイクロコード開発の助けとなる．ほとんどのマイクロワードは，29116および2910の制御に使われる．

マイクロエンジンは，部品数を最少に抑え，専用の外部ハードウェアに従属することによって柔軟性が低下しないようにするため，非常に単純化されている．この場合に使用される他のハードウェアとは，記憶アドレス・レジスタおよびデータ転送部である．

3.4 マイクロコード

プロセッサ用のマイクロコード開発は，UNIVAC 1100 で実施した．目的コードはディスケットに転送し，68000システムによって制御記憶にロードした．いくつかのテバッキング・エイドもプロセッサのハードウェアに組み込んだ．これらは，単一操作と区切り点機能，29116データ・バスと2910の次アドレスの表示を含んでいる．

マイクロコードは，コマンド解読部，共通サブルーチンおよび個別のコマンド処理部の三つの領域に分けることができる．29116の32個のレジスタによって，すべての変数の記憶，サブルーチンへのパラメタ引き渡し，定数の記憶が各部分の中で維持できるようになった．こうした方策によって，メモリ・アクセスを最小にして高速演算を保てるようになった．

3.5 ソフトウェア・インタフェース

ソフトウェア・システムは，68000 上で動き，ダイナミック・メモリを通して図形処理プロセッサとインタフェースを保つ．この記憶領域は，68000 および 29116 のアドレス空間に常駐する．記憶マップを図 7 に示す．最上段から数えて，最初の 120960 バイトは 1152×840 ピクセルの表示装置用ビット・マップを保持する．次の領域は，68000 が管理する現在使用中の書体を保持するのに使用される．テキスト文字を画面に表示するために，図形処理プロセッサはこの領域にある文字イメージを表示用記憶に転送する．記憶装置の 128 キロバイトの最後の部分は，図形処理プロセッサによって一時記憶領域として使用される．

記憶装置の 2 番目の 128 キロバイトの最初の部分は，ウインドウおよびウインドウの重なった交差部分，および他の書体のための記憶領域として使用され，68000 によって管理される．

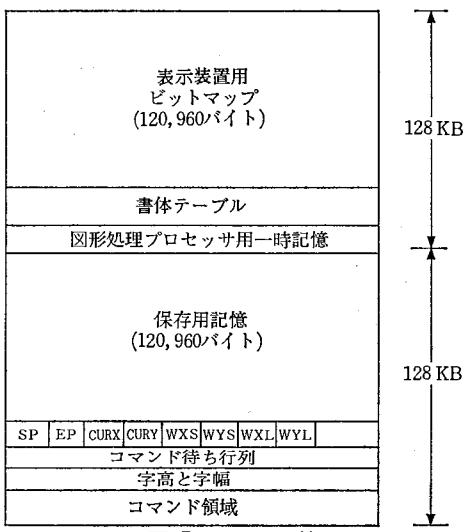


図 7 記　　憶

Fig. 7 Memory map

ウインドウが表示領域で重なり合うときは，68000 が図形処理プロセッサに命令を出して下になったウインドウの交差部をこの一時記憶領域に移す．これらの保存された部分は，“uncover window” 操作が遂行されると重なったウインドウを後で再構成するために使用される．つぎの領域の SP および EP の部分は，それぞれコマンド待ち行列の始めポインタと終わりポインタである．これらはコマンド待ち行列へのオフセットで，コマンド実行に使われる．次の 6 語は，テキスト・カーソルの相対位置 (CURX, CURY)，現役のウインドウ位置 (WXS, WYS) と現役のウインドウ寸法 (WX1, WY 1) を定義する．これらのエントリは，テキスト文字が図形処理プロセッサによって対応付けられる場所を制御する．この後ろに，環状になった 16 個のコマンド待ち行列が続く．コマンド待ち行列のエントリは，コマンド領域への索引ポインタであってコマンド・パケット (表示リスト) を指し示す．つぎに字高と字幅のテーブルが続く．このテーブルには，現在使用中の書体の個々の文字に対して一つずつエントリがあり，その中で文字の寸法を定義する．これによって文字幅に応じた字送り (欧文字送り) が可能になる．さらにコマンド領域が続く．コマンド・リストは，68000 によって組み立てられ，図形処理プロセッサによって実行される．

図形処理プロセッサは，ラスタ演算，パック，アンパック，線引き，塗りつぶし，円，DMA，ASCII 文字の対応付けを実行する．

1) ラスタ演算の転送，否定，論理和，論理積および排他的論理和は，送り側および受け側のラスタ領域で行われ，セットおよびクリアは単一のラスタ領域に適用される．
2) パック・コマンドは，ラスタ内のデータを取り出し，それを上述の保存用記憶領域に線形アドレス方式でパックする．画面上のウインドウのようなラスタ領域(長方形の区域)は，画面用記憶の線形アドレス空間には存在しない．パック・コマンドはまた，データおよび保存領域を圧縮するためにすべてのビットが0である語を抑止する．
3) アンパック・コマンドは，圧縮されたデータを保存領域から取り出し，それを表示領域に写像するという逆の演算を遂行する．
4) 線引きは，始点から終点までの間に全部1,0または非等価 (XOR) の線を引く．
5) 塗りつぶしは，多角形領域内をあるパターンで充塡する．
6) 円描きコマンドは，円弧を描く．本稿執筆時点では，塗りつぶしと円描きはマイクロコード化されていない．
7) DMA コマンドは，データ保存を管理し，データを送り側から受け側まで転送するだけで，68000 システムを支援する．
8) ASCII 文字コマンドは，文字を書体テーブルから表示可能な画面用記憶に写像する文字の寸法を決めるために字高／字幅テーブルを使用する．このコマンドは，またテキスト・カーソルを管理する．

4. ソフトウェア

4.1 ツール

FROG システムすべてのソフトウェアは，すべて UNIVAC 1100/83 で生成された．本来のツールは，PASCALコンパイラ，68000 アセンブラ，29116 アセンブラおよび各種整形用ユーティリティを含む．1100 上で生成された目的モジュールは，標準ディスケット形式で実行用モジュールに結合される．このコードは，UTS 400 を通してディスケットにダンプされ，その後実行のためにディスケットからロードされる．

PASCALコンパイラは，Motorola 社から供給された評価コピーであって，PASCAL で書かれ，1100 PASCALコンパイラ(最適化機能をもつ2パス・コンパイラ)でコンパイルされた．コンパイルされたコード用の実行時の入出力部分は，すべて FROG システム構成用にとくに書かれた．実数演算はサポートしていない．

1100 上の 68000 アセンブラは，標準の Motorola 社の原始コードを受け入れるが，これは PASCAL で書かれている．

29116 アセンブラは，MASM PROC アセンブラである．原始プログラムの記法は，AMD アセンブラの規約に酷似しているので，29116 用マニュアルをプログラム解説書として使用することができる．また，U-1100 の再配置可能モジュールをディスケットにダンプする前に二つの整形プログラムを通した．

4.2 実行時の環境

図8は，システムで走行するソフトウェアとマイクロコードのブロック図である．68000 コードの最下位レベルは，基本監視プログラムであって，プログラムをロードし，始動させ，記憶域管理を扱い，キーボードからの割り込みおよび通信回線をさばく．監視プログラムはアセンブラで書かれており，多重タスクをサポートしていない．

PASCAL の実行時コードもアセンブラで書かれており，SASI ディスク・インタフェースを直接処理し，監視プログラムに対して文字列操作および記憶域管理操作を要求する．

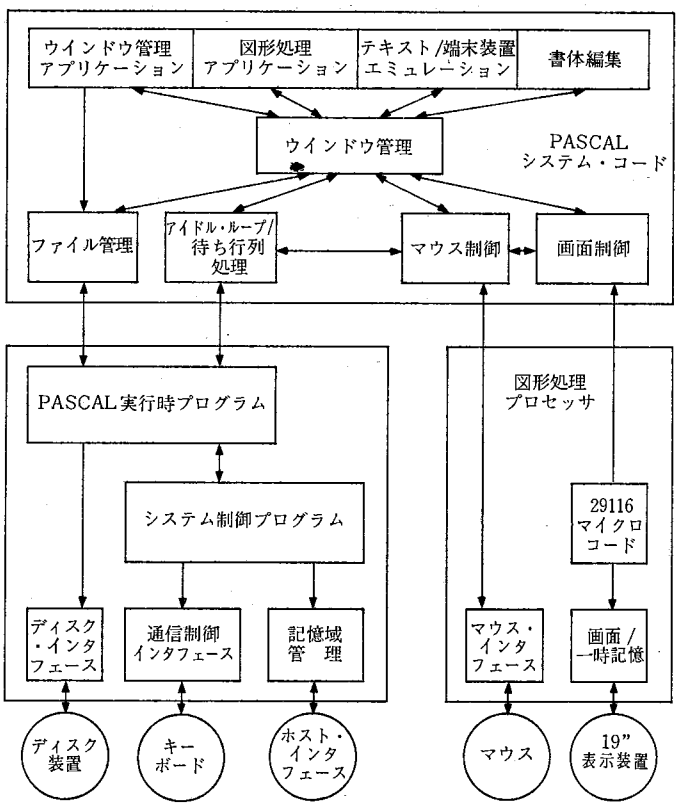


図 8　ソフトウェア・ブロック図

Fig. 8　Software block diagram

ステムのほとんどのコードはPASCALで書かれている．現行のシステムは，約 50 キロバイトの PASCAL で生成された目的コードをもっている．

4.3 FROG コード

4.3.1 ファイル管理

ファイル管理プログラムは，UNIX ファイル・システムの複製で，UNIX (Bell 研究所の商標) のほとんどの機能がサポートされている．システムの呼び出しには，open, close, read, write, stat, link, unlink, sync, mount, unmount と Lseek を含んでいる．ファイル管理系は，記憶ディスクの 10 セクタ分のキャッシュをもっている．これらはバッファを他のセクタに配置しなければならないときだけディスクに書き込まれる．システムは，セクタを 512 バイト・ブロックとして扱い，各ブロックは，実際のディスク・セクタの一連番号に 1 対 1 に対応付けられる．システムは，ファイルのすべてのウインドウに対してパック，アンパックといったデータ圧縮を用いる．

4.3.2 画面制御部

画面制御部は，システム・コマンド領域を管理し，高位レベルのモジュールによって作られたコマンドを待ち行列に入れる役割を果たす．

4.3.3 マウス制御部

マウス制御部は，マウス・インタフェースを管理し，現在のマウス・カーソルの動きを追跡する．この部分では，見かけ上のマウス移動をフィルタにかけなければならない．マウスをわずかに動かすと，画面上でカーソルが 1 ピクセル動く．中間的な動きをすると，

カーソルとマウスはほぼ同じ量だけ動くことになる．急速かつ大まかな位置決めをするためには，2倍の移動量でマウスを大きく，速く動かさなければならない．

マウスのサンプリング率は重大な因子であって，所期の効果を達成するためには，マウスを使ったアプリケーションによって管理しなければならない．たとえば，ペイント・プログラムは，滑らかな線を引くためにマウスの動き1ピクセル分に抑制しなければならない．

4.3.4 アイドル・ループ

アイドル・ループは，キーボードおよび通信回線から入ってくる文字列を監視し，マウス・カーソル更新ルーチンを呼び出し，アクセス可能ならばオンライン・ウインドウを更新する．

4.3.5 ウインドウ管理

ウインドウは，ファイルと同様に取り扱われる．呼び出す側では，ウインドウの大きさ，名前，オープンした時点での位置を指定する．このプログラムによってウインドウの位置を変更し，テキストと数値を読み書きし，書体を変更し，ディスクにウインドウを保存し，ディスクから再現し，ラスタ演算を遂行することができる．

ウインドウはクローズすると，画面から削除される．ウインドウ管理は，どのウインドウが別のウインドウの下に隠れているかを追いかけている．下に隠れたウインドウのデータは，ウインドウのオープン時に自動的に保存される．データは，上にかぶさっているウインドウがクローズされるか，または取り除き (uncover) コマンドが実行されるかすると再現される．ユーザは，上からかぶさったウインドウに書き込みはできない．この決定は，プロジェクトの初期になされ，システム設計を単純にした．図形処理プロセッサは，上にかぶさっているウインドウを速く取り除くのでこれに伴う諸問題が解決できる．

4.3.6 適　用

ウインドウ管理の図形処理パッケージ，書体編集系とテキスト・ユーティリティは，すべて低レベルのハンドラを呼び出すことができる．これらはすべて高度な対話形式であって，マウスおよびウインドウの用法を拡げることになる．

5. おわりに

本稿で報告した活動によって，単純でかつ操作しやすいシステムが生成された．ほとんどの新ユーザは，問題なくマウスとウインドウに適応できると考えられる．アプリケーションとのインタフェースは単純でかつ自然に見える．当初はマウスに対して懐疑的であった著者も現在では信者である．マウスがあるシステムで使用可能ならそれを標準にすべきで，任意選択すべきではない．ユーザとのアプリケーション・インタフェースは，マウスの有無によって非常な影響を受けるため，このことは重要である．両方の環境で同様にうまく機能するプログラムを書くことは非常に困難であろう．このように，マウス機能は，おそらく最小公分母を満たそうとするアプリケーション・プログラマには無視されるであろう．

文字および図形の処理の速度は，われわれの設計目標に合致したし，むしろこれを越えた．Apple 社の Lisa は，図形処理プロセッサとしてシフト回路をわずかに直接装備した 68000 を使っている．これは Lisa の画面上で比較的良好に作動するが，FROG 画面の約 1/3 の大きさしかない．著者のグループによる研究では，1000×1000 ピクセルの画面を文字で埋めるには 68000 で約 15 秒かかる．FROG は，これを 2/3 秒で行う．付加バイポー

ラ・プロセッサはそれが果たす性能と対比して追加費用に見合うと信じている．

マウスは，一定割合の中断でサンプリングしなければならない．このことによってユーザ・インタフェースが，より一貫したものになり，タイミングの計算が単純になる．ほとんどのコードを PASCAL で書いたのは良い判断であった．最も時間のかかるデータ操作を 29116 で行ったので，画面処理の部分を熟練を要する 68000 アセンブラで書く必要がなかった．しかしながら，将来のプロジェクトのために考えなければならない変更点がいくつかある．29116 が FROG ボードの RAM に加えてシステム RAM にアクセスできたら，ウインドウ管理ははるかに単純になったであろう．これによって生じた問題はすべて，結果的に解決されたが，将来のプロジェクトでは再評価する価値があるであろう．考慮すべきもう一つの変更は，われわれが使った単純な単一スレッド EXEC のかわりに多重タスク EXEC の新しい設計にとりかかることであろう．外部事象を待つ場合，ソフトウェアがアイドル・ループを呼び出す必要がなかったら，プログラムの待ちループはより単純になったであろう．このアプローチは，従来の設計より記憶領域を余計に使用することは間違いない．128 K のスクラッチ RAM は，ウインドウ保存と再現には大体妥当なようであった．また，パックおよびアンパック・コマンドにおいてマイクロコードによってデータ圧縮をしなかったら，これは達成できなかったであろう．

（応用ソフトウェア二部　渡辺　啓　訳）

参考文献　[1] W. M. Newman, R. F. Sproull, : *Principals of Interactive Computer Graphics, Second Edition*, McGraw-Hill, 1979.

執筆者紹介　Robert G. Bond

1973年 Utah 大学卒業，1973年7月から1984年7月，Sperry 社において端末および通信機器の研究開発に従事．最近3年間は，図形および音声アプリケーション，UNIX オペレーティング・システムを担当．1984年7月以降，National Semiconductor 社に勤務．

Robert C. Dawson

1979年6月 California 州立大学卒業．1979年7月から1980年6月まで Hughes Aircraft 社において，ソナー・システム用マイクロプロセッサ・ベースの制御装置を担当．1980年6月から1982年3月まで Sperry 社 Irvine 工場において，32ビット・ミニコンピュータの記憶装置の開発を担当．1982年3月から1984年4月まで Salt Lake 工場において先任設計技術者として，モノクローム・ビットマップ形表示装置 (FROG) の設計とマイクロプログラムの開発，1280×1024 ドット・カラー表示装置の開発を担当．1984年5月以降，Evans & Sutherand 社において3次元図形処理装置の開発を担当．

報告 経路術語によるテスト・データの作成

Test Case Selection by Path Predicates

長谷川 光 邦

要 約 本稿では，ホワイト・ボックス法の一つである J. C. Huang の論文 "An Approach to Program Testing" に従い，テスト・データの作成を試みた手順を述べる．Huang の方法は，プログラム・コード（空命令への分岐も含めて）すべてが実行されるようにプログラム経路を選び出し，それぞれの経路を決定する条件（術語）によって入力データ空間をクラス分けし，各クラスから代表データを選出し，それらをテスト・データとするものである．（いわゆるC1カバレッジである．）この方法は，プログラム作成者側のモジュール・テストや単体テストの段階では，効果的な方法であるが，人手で行うにはあまりにも労の多い作業である．したがって，コンピュータによる支援の提案もあわせて行う．

Abstract This tutorial describes the trial to select test cases according to the report issued on the Computing Surveys by J. C. Huang, "An Approach to Program Testing".

1) Select appropreate program pathes so that every code (including null statements in ELSE clauses) will be traversed at least once during the test.
2) Classfy input data space by the predicates which determine each program pathes.
3) Select representatives for each classes. Then, take these representatives as test data sets.

This method is effective especially for the program developers in their module teting or unit testing. However, this work is too heavy for human being; I propose some test supporting programs by the computer.

1. テスト・データの選び方

テスト・データとは，テスト対象プログラムに与える入力データと，その結果の出力データ／現象との対である．正しい出力データ／現象をあらかじめ決定できないプログラムについては，ここでは考えない[2]．

テスト・データを作るには，テスト・データを抽象的に記述したテスト・ケースをいくつか設定する．テスト・ケースの設定には二つの考え方がある．一つは，テスト対象プログラムをブラック・ボックスと考え，外部仕様にもとづいてテスト・ケースを創出する方法である．もう一つはできあがったプログラムにもとづいてテスト・ケースを見つける方法である．後者の方法は，とくにプログラム作成者側でのモジュール・テスト，単体テストの段階で有効である．また，品質保証の意味からも内部構造にもとづくテストは当然であろう．

本稿では，J. C. Huang の論文 "An Approach to Program Testing[1]" に従い，テスト・データの作成について述べる．Huangの方法は，プログラム・コードのすべてが実行されるようにプログラム経路をいくつか選び出し，その経路を決定する条件（術語）によって入力データ空間をクラス分けし，それぞれのクラスから代表データを選出し，これをテスト・データとするものであり，ホワイト・ボックス法の一つである．

2. 経路術語によるプログラム・テスト・データの作成手順

停止するプログラムは，プログラム開始点から終止点に至る経路に沿った（制御文でな

い）文の列からなる部分プログラムの総体と考える．それぞれの部分プログラムに対応して，入力データ空間をクラス分けすることができる．このクラス分けの条件を経路術語という．たとえば図1の流れ図に示される三つの数の最大値を求めるプログラムは，その経路によって四つの部分プログラムに分割される．図2のＮＳ図のケース分けの条件がその経路術語であり，小箱が部分プログラムである．

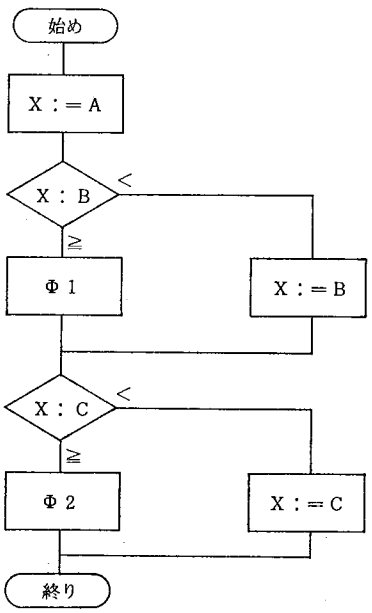


図 1　流れ図

Fig. 1　Sample flow chart

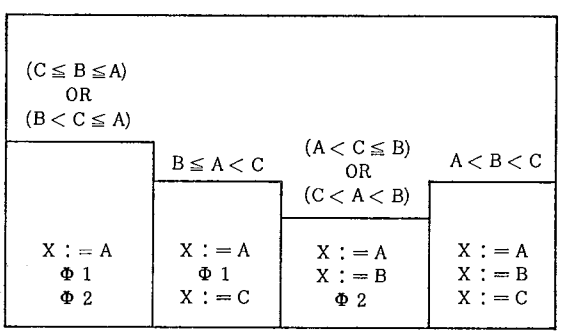


図 2　NS 図

Fig. 2　NS chart

これらの部分プログラムがそれぞれ正しく動作すれば，元のプログラムも正しく動作すると考えられるので，これらの経路ごとに1個以上のテスト・データを作り，元のプログラムに与える．この場合テスト・ケースは，経路または経路術語で表現される．テスト・ケースごとにすべてのテスト・データで期待どおりの結果が得られれば，そのプログラムのテストを完了する．しかし，一般には不定回数の繰り返しがあると，部分プログラムの数は可付番無限個となってしまうので，これらの部分プログラムのうち，空の命令を含めてすべての命令がカバーされるような有限個の経路の集合を選出する．これは流れ図上の

すべての線が，いずれかのテスト・データで通過されるようにテスト・ケースを選出するC1カバレッジと同じものである．

2.1 経 路 術 語

1) 判定間経路と判定間経路術語……プログラム経路の要素として分岐点から分岐点までの小経路を考える．この小経路の並びがプログラム経路となる．

プログラムの開始点または分岐点から，分岐点または終止点までのいずれかの経路で，途中に分岐点を含まないものを**判定間経路**という（セグメント，エッジ，枝ともいう）．

それぞれの判定間経路について，その経路の先頭の分岐点で該当経路側に制御を移す条件をその**判定間経路の経路術語**という．プログラムの開始点からはじまる経路についてはその経路術語を true とする．たとえば，図3で判定間経路A，Bの経路術語はそれぞれP，～Pである．

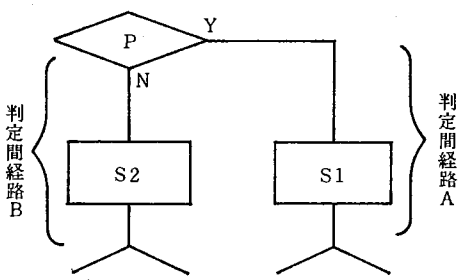


図 3　経路術語

Fig. 3　Path predicate

判定間経路の中に含まれる命令は代入文のみと考える．入力文はプログラム内の変数に対する代入文と考え，出力文は外部変数に対してプログラム内の変数や定数を代入する文とみなす．

2) 経路のたぐり上げ……図4でAからBに至る経路を考える．点Bでは術語Qが成り立っているとすると，この経路についてAでは術語

$$P \wedge Q(E \rightarrow X)$$

が成り立っていなければならない．ここで $Q(E \rightarrow X)$ は Q の中のすべての X を E で置き換えた術語である．

3) プログラム経路術語……プログラム開始点からプログラム終止点に至るプログラム経路について，終止点における術語を true（到達しなければならない）とし，経路に沿ってたぐり上げていく．プログラム開始点で，初期値をもつ変数には初期値を与える代入文があったものとし，初期値をもたない変数には undefined が与えられたもの

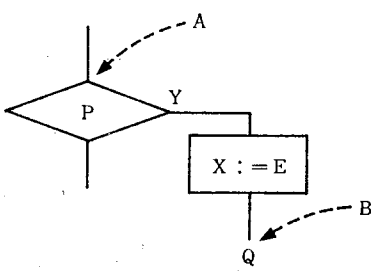


図 4　たぐり上げ

Fig. 4　Backward dragging

とする．これが，そのプログラム経路の経路術語である．

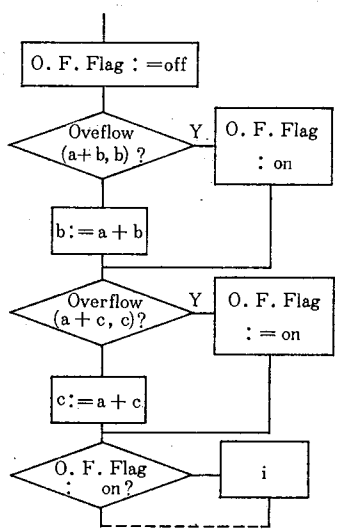
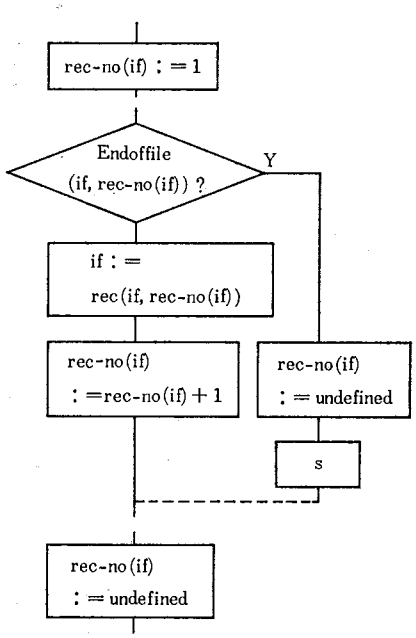
2.2 プログラム経路図の作成

テスト・ケースを選び出す第1段階としてプログラム経路図を作成する．プログラム経路図は流れ図の判定（菱形）と線だけで表現されるものである．プログラム設計時に作成した流れ図で代用してもよいが，次の点に注意し，必要なら書き直す．

1) 命令の読み替え……プログラミング言語によっては，表現上は現れていない代入文や内部変数へのセット等を考えるのが正しいプログラムの実行を表す場合がある．そ

表 1 命令の読み替え例

Table 1 Example of statement refining

COBOLコード	流れ図（例）
ADD a TO b, c ; ON SIZE ERROR i.	
if を順ファイルとする OPEN INPUT if READ if AT END s.	
CLOSE if	

れらの命令はその意味にあわせて，適切な術語を導入して読み替えておく．また，入出力ファイルについては，これをレコードの1次元の並びとして読み替えるものとする．表1に COBOL 言語で例示する．

2) プログラム経路図……制御の流れがわかるように，条件分岐を明示するプログラム経路図を作成する．プログラムと条件の対応がわかるようにすればよいので，処理を示す箱（長方形）は書かなくてもよい．

2.3 判定間経路からプログラム経路術語の作成

1) 判定間経路術語と代入文……つぎに，プログラム経路図から判定間経路をすべて拾い出し，それに名前を付ける．それぞれの判定間経路について，その判定間経路術語とそこに含まれる代入文を表にしておく．

2) プログラム経路の選出……プログラム経路図にもとづき，プログラムの開始点から終止点に至るプログラム経路をいくつか選び出す．このとき，どの判定間経路もいずれかのプログラム経路に含まれるようにする．できるならば，プログラムの意味としてありえない経路（固定回の繰り返し，1回目の特別ルート等）にならないようにする．ここで誤った経路を選ぶと，あとでこの段階までの手戻りになる．

3) プログラム経路のたぐり上げとプログラム経路術語……2)で選出したプログラム経路のそれぞれについてプログラム終止点からたぐり上げ（実は，逆向きの記号実行）を行い，プログラム経路術語を作成する．

このとき恒等的に false になるようなプログラム経路は，このプログラムでは事実上ありえない経路だから，2)に戻り再度プログラム経路を選出しなおす．どうしてもある判定間経路を通るプログラム経路が選べないときは，そのプログラムに誤りがある．経路術語を整理する．たぐり上げた術語は and 結合になっている．その中の一部に false があれば全体として false になるので早めに恒等的な false が見つかる．定数が代入される時にその部分を評価すればよい．術語の整理は次のように行う．

- カッコの外にある否定は，すべて内側に組入れ原子式に付ける．原子式を書き換えて否定を消去できるものは消去する．

$$\sim(P\wedge Q) \rightarrow \sim P\vee\sim Q$$
$$\sim(P\vee Q) \rightarrow \sim P\wedge\sim Q$$
$$\sim(a=b) \rightarrow a\neq b$$
$$\sim(a<b) \rightarrow a\geqq b$$
$$\sim(a\leqq b) \rightarrow a>b$$

- 式は連言標準形に変形する．

$$(P\wedge Q)\vee R \rightarrow (P\vee R)\wedge(Q\vee R)$$
$$P\vee(Q\wedge R) \rightarrow (P\vee Q)\wedge(P\vee R)$$

- 数値データについて，不等式を等式に変換する．

$$a\neq b \rightarrow \exists x\neq 0(x=b-a)$$
$$a<b \rightarrow \exists x>0(x=b-a)$$
$$a\leqq b \rightarrow \exists x\geqq 0(x=b-a)$$

- 同一変数を含む項を組み合わせて（連立方程式を解くように）変数を減らす

$$\exists x\neq 0(x=b-a)\wedge\exists x\neq 0(x=b+1-c)$$
$$\rightarrow \exists x\neq 0, \exists y\neq 0(x=b-a\wedge y=b+1-c)$$
$$\rightarrow \exists x\neq 0, \exists y\neq 0(y-x=a+1-c)$$

2.4 プログラム経路術語からテスト・データの作成

プログラム経路術語は，入力データ空間のあるクラスを表している．内部状態をもつプログラムについては，入力データ空間と内部状態空間との直積空間のクラスになるが，その直積空間を入力データ空間とみなす．

1) テスト・ケース……判定条件が複雑な式の場合，テストが不十分となり，本格的な使用を開始してから問題を起こすことが時々ある．そこで，プログラム経路術語の中でOR結合がある場合，テスト・ケースとしては，OR結合の原子式について，ただ一つのものを true とし，他はすべて false とする組合せを拾い上げるようにする．

$$\cdots\cdots\wedge(P1\vee P2\vee P3)\wedge\cdots\cdots$$

$$\rightarrow\begin{cases}\cdots\cdots\wedge\ \ P1\wedge\sim P2\wedge\sim P3\wedge\cdots\cdots\\ \cdots\cdots\wedge\sim P1\wedge\ \ P2\wedge\sim P3\wedge\cdots\cdots\\ \cdots\cdots\wedge\sim P1\wedge\sim P2\wedge\ \ P3\wedge\cdots\cdots\end{cases}$$

ただし，OR結合が複数個あったとしても，それらのすべての組合せとする必要はない．

2) テスト・データ……テスト・ケースごとに，入力データ空間の中から具体的なデータを割り当てる．このとき，入力データとして範囲が限定されるものについては，その境界にできるだけ近いものを使うことがのぞましい．

入力データが決まったなら，その入力に対して期待されるプログラムの出力／現象を想定し，テスト・データとする．

3. テスト・データ作成の実例

プログラム仕様とプログラムを図5，図6に示す．

1．名称：在庫一覧表作成

2．目的：在庫データを入力し，在庫量を計算して在庫量一覧表を作成する．

3．入力データ様式

区分＝1：前在庫
2：仕入
3：仕入返品
4：売上
5：売上返品

4．出力データ様式

商品コード	区分	数量
100	ゼンザイコ	200
	シイレ	80
	シイレヘンピン	10
	ウリアゲ	140
	ザイコ	130
102	エラー	***
104	ゼンザイコ	100

5．プログラムの機能（一部省略）

在庫データ・ファイルの商品コード・グループごとに

〈在庫〉：＝〈前在庫〉＋〈仕入〉－〈仕入返品〉－〈売上〉＋〈売上返品〉を計算し，入力データと在庫行とを印書する．ただし，エラー・データについてはエラー行を印書する．エラー・データとは，…(略)

図 5 プログラム仕様例

Fig. 5 Sample program specification

1) プログラム経路図の作成……図6のプログラムで下線で示した部分が判定であり，これらをもとに図7のグラフを得る．菱形の中の数字はプログラムの行番号である．判定間経路としてはイ～ケの31個がある．表2に判定間経路術語とその代入文とを整理しておく．

2) プログラム経路の選出……すべての判定間経路を含むようなプログラム経路として

```
 1       IDENTIFICATION DIVISION.
 2       PROGRAM-ID.      ZAIKO-ICHIRAN.
           .
           .
61       PROCEDURE DIVISION.
62       ZAIKO-ICHIRAN.
63           PERFORM HAJIME-SHORI.
64           PERFORM SHORI UNTIL FILE-END = 1.
65           PERFORM OWARI-SHORI.
66           STOP RUN.
67       *
68       *    ｼｮｷｶ ｼｮﾘ
69       *
70        HAJIME-SHORI.
71           OPEN INPUT  CARD-FILE
72                OUTPUT PRINT-FILE.
73           MOVE 0 TO FILE-END.
74           READ CARD-FILE AT END ADD 1 TO FILE-END.
75       *
76       *    ｻﾞｲｺ ｲﾁﾗﾝﾋｮｳ ｻｸｾｲ
77       *
78        SHORI.
79           PERFORM SHOHIN-HAJIME.
80           PERFORM SHOHIN-SHORI UNTIL SHOHIN-END = 1.
81           PERFORM SHOHIN-OWARI.
82       *
83        SHOHIN-HAJIME.
84           MOVE SHOHIN-CODE TO W-SHOHIN-CODE.
85           MOVE 2 TO W-KUBUN-SEQ.
86           PERFORM SURYO-CLEAR VARYING I FROM 1 BY 1
87                               UNTIL I > 5.
88           IF KUBUN =1
89               IF SURYO NUMERIC
90                   MOVE SURYO TO W-SURYO (1)
91                   MOVE 0 TO ERR-FLAG
92               ELSE
93                   MOVE 1 TO ERR-FLAG
94           ELSE
95               MOVE 1 TO ERR-FLAG.
96           MOVE 0 TO SHOHIN-END.
97           READ CARD-FILE AT END ADD 1 TO FILE-END
98                              MOVE HIGH-VALUE TO SHOHIN-CODE.
99           IF SHOHIN-CODE NOT = W-SHOHIN-CODE
100              MOVE 1 TO SHOHIN-END
101          ELSE
102              NEXT SENTENCE.
103       SURYO-CLEAR.
104          MOVE 0 TO W-SURYO (1).
105      *
106       SHOHIN-SHORI.
107          IF SURYO NUMERIC
108              IF KUBUN NOT < W-KUBUN-SEQ AND KUBUN < 6
109                  MOVE KUBUN TO W-KUBUN-SEQ
110                  ADD SURYO TO W-SURYO (KUBUN)
111              ELSE
112                  ADD 1 TO ERR-FLAG
113          ELSE
114              ADD 1 TO ERR-FLAG.
115          READ CARD-FILE AT END ADD 1 TO FILE-END
116                             MOVE HIGH-VALUE TO SHOHIN-CODE.
117          IF SHOHIN-CODE NOT = W-SHOHIN-CODE
118              MOVE 1 TO SHOHIN-END
119          ELSE
120              NEXT SENTENCE.
```

```
121      *
122       SHOHIN-OWARI.
123           MOVE W-SHOHIN-CODE TO P-SHOHIN-CODE.
124           IF ERR-FLAG = 0
125               PERFORM SHOHIN-HYOJI VARYING I FROM 1 BY 1
126                                    UNTIL I > 5.
127               PERFORM ZAIKO-HYOJI
128           ELSE
129               PERFORM ERR-HYOJI.
130       SHOHIN-HYOJI.
131           IF W-SURYO (I) NOT = 0
132               MOVE W-KUBUN (I) TO P-KUBUN
133               MOVE W-SURYO (I) TO P-SURYO
134               WRITE PRINT-REC FROM FRINT-HENSHU
135               MOVE SPACE TO P-SHOHIN-CODE
136           ELSE
137               NEXT SENTENCE.
138       ZAIKO-HYOJI.
139           ADD W-SURYO (1) W-SURYO (2) GIVING ZAIKO.
140           ADD W-SURYO (5) TO ZAIKO.
141           SUBTRACT W-SURYO (3) W-SURYO (4) FROM ZAIKO.
142           MOVE 'ｻﾞｲｺ' TO KUBUN.
143           MOVE ZAIKO TO P-SURYO.
144           WRITE PRINT-REC FROM PRINT-HENSHU.
145       ERR-HYOJI.
146           MOVE 'ｴﾗｰ' TO P-KUBUN.
147           MOVE ' ***' TO P-HOSI.
148           WRITE PRINT-REC FROM PRINT-HENSHU.
149       *
150       *    ｵﾜﾘ ｼｮﾘ
151       *
152        OWARI-SHORI.
153           CLOSE PRINT-FILE
154                 CARD-FILE.
```

図 6　プログラム例

Fig. 6　Sample program

次の四つを得た．（この四つに限られるものではないし，最小のものでもない．）

a）　イロニ

b）　イハホヘヘヘヘヘトチリヲカタノオクオヤオヤオヤオヤマニ

c）　イハホヘヘヘヘヘトチヌワヨレソツラウタケニ

d）　イハホヘヘヘヘヘトルワヨレソネムキレナラウタケニ

3)　プログラム経路術語……2)のプログラムについて，経路のたぐり上げを行い，プログラム経路術語を整理する．以下にその結果を示す．c)について一部途中経過も示す．

a）　イロニ

```
eof(CARD-FILE,1)
```

b）　イハホヘヘヘヘヘトチリヲカタノオクオヤオヤオヤオヤマニ

```
~eof(CARD-FILE,1)
^KUBUN OF rec(CARD-FILE,1)=1
^numeric(SURYO OF rec(CARD-FILE,1))
^eof(CARD-FILE,2)
^HIGH-VALUE≠SHOHIN-CODE OF rec(CARD-FILE,1)
^SURYO OF rec(CARD-FILE,1)≠0
```

c）　イハホヘヘヘヘヘトチヌワヨレソツラウタケニ

- はじめ　ニまで

```
FILE-END= 1
```

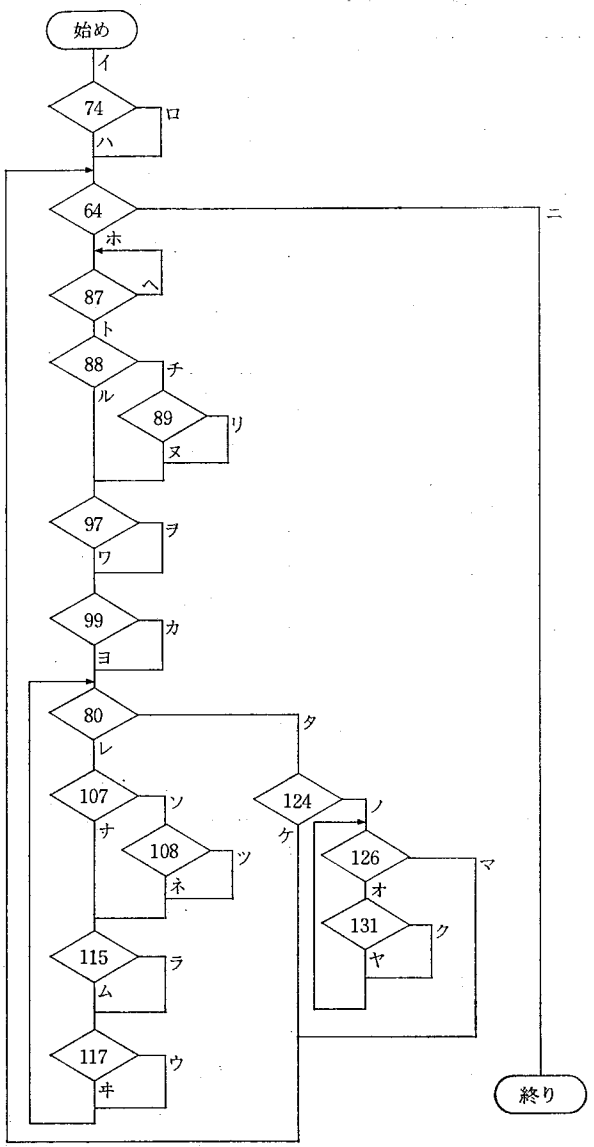


図 7 プログラム経路図

Fig. 7 Program path chart

- ケまで

  ERR-FLAG≠0

  ^FILE-END=1

- タまで

  SHOHIN-END=1

  ^ERR-FLAG≠0

  ^FILE-END=1

表 2　判定間経路とその術語・代入文

Table 2　Dicision-to-dicision pathes and their predicates, assignment statements

イ		rec-no(CARD-FILE):=1 rec-no(PRINT-FILE):=1 FILE-END:=0
ロ	eof(CARD-FILE,rec-no(CARD-FILE))	rec-no(CARD-FILE):=undefined FILE-END:=FILE-END+1
ハ	～eof(CARD-FILE,rec-no(CARD-FILE))	CARD-REC :=rec(CARD-FILE,rec-no(CARD-FILE)) rec-no(CARD-FILE) :=rec-no(CARD-FILE)+1
ニ	FILE-END = 1	rec-no(CARD-FILE):=undefined rec-no(PRINT-FILE):=undefined
ホ	FILE-END ≠ 1	W-SHOHIN-CODE :=SHOHIN-CODE OF CARD-REC W-KUBUN-SEQ:=2 I:=1
ヘ	I ≦ 5	W-SURYO(I):=0 I:=I+1
ト	I > 5	
チ	KUBUN OF CARD-REC = 1	
リ	numeric(SURYO OF CARD-REC)	W-SURYO(1):=SURYO OF CARD-REC ERR-FLAF:=0 SHOHIN-END:=0
ヌ	～numeric(SURYO OF CARD-REC)	ERR-FLAG:=1 SHOHIN-END:=0
ル	KUBUN-OF CARD-REC ≠ 1	ERR-FLAG:=1 SHOHIN-END:=0
ヲ	eof(CARD-FILE,rec-no(CARD-FILE))	rec-no(CARD-FILE):=undefined FILE-END:=FILE-END+1 SHOHIN-CODE OF CARD:=HIGH-VALUE
ワ	～eof(CARD-FILE,rec-no(CARD-FILE))	CARD-REC :=rec(CARD-FILE,rec-no(CARD-FILE)) rec-no(CARD-FILE) :=rec-no(CARD-FILE)+1
カ	SHOHIN-CODE OF CARD-REC ≠ W-SHOHIN-CODE	SHOHIN-END:=1
ヨ	SHOHIN-CODE OF CARD-REC = W-SHOHIN-CODE	
タ	SHOHIN-END = 1	P-SHOHIN-CODE OF PRINT-HENSHU :=W-SHOHIN-CODE
レ	SHOHIN-END ≠ 1	
ソ	numeric(SURYO OF CARD-REC)	
ツ	(KUBUN OF CARD-REC ≧ W-KUBUN-SEQ) ^(KUBUN OF CARD-REC < 6)	W-KUBUN-SEQ:=KUBUN OF CARD-REC W-SURYO(KUBUN OF CARD-REC) :=SURYO OF CARD-REC
ネ	～((KUBUN OF CARD-REC ≧ W-KUBUN-SEQ) ^(KUBUN OF CARD-REC < 6))	ERR-FLAG:=ERR-FLAG+1
ナ	～numeric(SURYO OF CARD-REC)	ERR-FLAG:=ERR-FLAG+1
ラ	eof(CARD-FILE,rec-no(CARD-FILE))	rec-no(CARD-FILE):=undefined FILE-END:=FILE-END+1 SHOHIN-CODE OF CARD-REC:=HIGH-VALUE
ム	～eof(CARD-FILE,rec-no(CARD-FILE))	CARD-REC :=rec(CARD-FILE,rec-no(CARD-FILE)) rec-no(CARD-FILE) :=rec-no(CARD-FILE)+1
ウ	SHOHIN-CODE OF CARD-REC ≠ W-SHOHIN-CODE	SHOHIN-END:=1
	SHOHIN-CODE OF CARD-REC = W-SHOHIN-CODE	
ノ	ERR-FLAG = 0	I:=1
オ	I ≦ 5	
ク	W-SURYO(I) ≠ 0	P-KUBUN OF PRINT-HENSHU:=W-KUBUN(I) P-SURYO OF PRINT-HENSHU:=W-SURYO(I) PRINT-REC:=PRINT-HENSHU rec(PRINT-FILE,rec-no(PRINT-FILE)) :=PRINT-REC rec-no(PRINT-FILE) :=rec-no(PRINT-FILE)+1 P-SHOHIN-CODE OF PRINT-HENSHU :=SPACE I:=I+1

ヤ	W-SURYO(I) = 0	I:=I+1
マ	I > 5	ZAIKO:=W-SURYO(1)+W-SURYO(2) ZAIKO:=ZAIKO+W-SURYO(5) ZAIKO:=ZAIKO-W-SURYO(3)-W-SURYO(4) P-KUBUN OF PRINT-HENSHU:='ｻﾞｲｺ' P-SURYO OF PRINT-HENSHU:=ZAIKO PRINT-REC:=PRINT-HENSHU rec(PRINT-FILE,rec-no(PRINT-FILE)) :=PRINT-REC rec-no(PRINT-FILE) :=rec-no(PRINT-FILE)+1
ケ	ERR-FLAG ≠ 0	P-KUBUN OF PRINT-HENSHU:='ｴﾗｰ' P-HOSHI OF PRINT-HENSHU:=' ***' PRINT-REC:=PRINT-HENSHU rec(PRINT-FILE,rec-no(PRINT-FILE)) :=PRINT-REC rec-no(PRINT-FILE) :=rec-no(PRINT-FILE)+1

- ウまで

  SHOHIN-CODE OF CARD-REC≠W-SHOHIN-CODE

  ^1=1

  ^ERR-FLAG≠0

  ^FILE-END=1

  下線の部分は恒等的に true なので，ここで評価して以下には引きずらない．

- ラまで

  eof(CARD-FILE,rec-no(CARD-FILE))

  ^HIGH-VALUE≠W-SHOHIN-CODE

  ^ERR-FLAG≠0

  ^FILE-END+1=1

- 少しとんでヨまで

  SHOHIN-CODE OF CARD-REC=W-SHOHIN-CODE

  ^SHOHIN-END≠1

  ^numeric(SURYO OF CARD-RED)

  ^KUBUN OF CARD-REC≧W-KUBUN-SEQ

  ^KUBUN OF CARD-REC<6

  ^eof(CARD-FILE,rec-no(CARD-FILE))

  ^HIGH-VALUE≠W-SHOHIN-CODE

  ^ERR-FLAG≠0

  ^FILE-END=1

- さらにとんでへへへへへのループが終わってホまで

  FILE-END≠1

  ^1≦5

  ^1+1≦5

  ^1+1+1≦5

  ^1+1+1+1≦5

  ^1+1+1+1+1≦5

  ^1+1+1+1+1+1>5

  ^KUBUN OF CARD-REC=1

  ^～numeric(SURYO OF CARD-REC)

^～eof(CARD-FILE,rec-no(CARD-FILE))
^SHOHIN-CODE OF rec(CARD-FILE,rec-no(CARD-FILE))
＝SHOHIN-CODE OF CARD-REC
^numeric(SURYO OF rec(CARD-FILE,rec-no(CARD-FILE))
^KUBUN OF rec(CARD-FILE,rec-no(CARD-FILE))≧2
^KUBUN OF rec(CARD-FILE,rec-no(CARD-FILE))＜6
^eof(CARD-FILE,rec-no(CARD-FILE)+1)
^HIGH-VALUE≠SHOHIN-CODE OF CARD-REC
^FILE-END+1＝1

ループが終わるとループ制御変数に初期値が与えられ，ループ制御の術語が一斉に true となる．

- 最後のイまで進めて

～eof(CARD-FILE,1)
^0≠1
^KUBUN OF rec(CARD-FILE,1)＝1
^～numeric(SURYO OF rec(CARD-FILE,1))
^～eof(CARD-FILE,1+1)
^SHOHIN-CODE OF rec(CARD-FILE,1+1)＝SHOHIN-CODE OF rec(CARD-FILE,1)
^numeric(SURYO OF rec(CARD-FILE,1+1)
^KUBUN OF rec(CARD-FILE,1+1)≧2
^KUBUN OF rec(CARD-FILE,1+1)＜6
^eof(CARD-FILE,1+1+1)
^HIGH-VALUE≠SHOHIN-CODE OF rec(CARD-FILE,1)
^0+1＝1

- これを整理して次の術語を得る．

～eof(CARD-FILE,1)
^KUBUN OF rec(CARD-FILE,1)＝1
^～numeric(SURYO OF rec(CARD-FILE,1))
^～eof(CARD-FILE,2)
^SHOHIN-CODE OF rec(CARD-FILE,2)＝SHOHIN-CODE OF rec(CARD-FILE,1)
^numeric(SURYO OF rec(CARD-FILE,2))
^KUBUN OF rec(CARD-FILE,2)≧2
^KUBUN OF rec(CARD-FILE,2)＜6
^eof(CARD-FILE,3)
^HIGH-VALUE≠SHOHIN-CODE OF rec(CARD-FILE,1)

d) イハホヘヘヘヘトルワヨレソネムキレナラウタケニ

～eof(CARD-FILE,1)
^KUBUN OF rec(CARD-FILE,1)≠1
^～eof(CARD-FILE,2)
^numeric(SURYO OF rec(CARD-FILE,2))
^(KUBUN OF rec(CARD-FILE,2)＜2 ∨
KUBUN OF rec(CARD-FILE,2)≧6)
^～eof(CARD-FILE,3)

```
^SHOHIN-CODE OF rec(CARD-FILE,3)=SHOHIN-CODE OF rec(CARD-FILE,1)
^~numeric(SURYO OF rec(CARD-FILE,3))
^eof(CARD-FILE,4)
^HIGH-VALUE≠SHOHIN-CODE OF rec(CARD-FILE,1)
```

4) テスト・ケースとテスト・データ……以上の術語からテスト・ケースを作成する.

a) は1回目のレコードの読込みでファイル終了に出会う (eof(CARD-FILE, 1)) という術語なので，空のファイルを用意すればよい.

b) は1回目のレコードは読み込めて (~eof(CARD-FILE, 1)), 1件目のレコードの区分欄が1で (KUBUN OF rec(CARD-FILE, 1)=1), その数量欄が数値で (numeric (SURYO OF rec(CARD-FILE, 1))), 2回目のレコードの読み込みでファイル終了に出会う (eof(CARD-FILE, 2)), さらに，1件目のレコードの商品コード欄は表意定数 HIGH-VALUE と等しくはない (HIGH-VALUE≠SHOHIN-CODE OF rec (CARD-FILE, 1)), また，1件目のレコードの数量欄はゼロではない (SURYO OF rec(CARD-FILE, 1)≠0), という条件を満足するテスト・データを用意する．ところで，最後の非ゼロの条件は仕様にあったのだろうか．明確に記述されていないなら，仕様作成者に確認しておかなければならない.

c) についても同様にテスト・ケースを考える．ここには2番目のレコードの区分欄について二つの条件

KUBUN OF rec(CARD-FILE, 2)≧2
∧KUBUN OF rec(CARD-FILE, 2)<6

があるので，これらを満足してしかもそれぞれの境界値に最も近い値（2と5）でテストを行うようにテスト・ケースは二つにする.

d) では，OR で結ばれた条件

(KUBUN OF rec(CARD-FILE, 2)<2∨
KUBUN OF rec(CARD-FILE, 2)≧6)

があるので，一つだけを true とする条件の組み合せに変える.

(KUBUN OF rec(CARD-FILE, 2)<2)
∧~(KUBUN OF rec(CARD-FILE, 2)≧6)

および

~(KUBUN OF rec(CARD-FILE, 2)<2)
∧(KUBUN OF rec(CARD-FILE, 2)≧6)

との二つのテスト・ケースに分ける.

それぞれのテスト・ケースごとに，具体的な入力データを定め，対応する出力データを予測する．表3にテスト・データの一覧表を示す.

4. テスト・ケース作成支援プログラム

以上に述べた方法は，プログラム作成者としてのテストでは非常に効果があると考えられる．そこで，いくつかの望ましい支援プログラムを提案する.

1) 判定間経路洗い出しプログラム……原始プログラムの静的流れ解析を行い，すべての判定間経路と，その判定間経路術語，そこに含まれる代入文の表を作成する．プログラム経路図（グラフ）を作ればさらに良い.

表 3 テスト・データー覧表

Table 3 Test data

ケース番号	入力データ			出力データ		
	商品コード	区分	数量	商品コード	区分	数量
1	(空)			(空)		
2	ABC	1	999	ABC	ゼンザイコ	999
					ザイコ	999
3—1	123	1	200	123	ゼンザイコ	200
	123	2	50		シイレ	50
					ザイコ	250
3—2	$!%	1	100	$!%	ゼンザイコ	100
	$!%	5	10		ウリアゲヘンピン	10
					ザイコ	110
4—1	4—1	0	0	4—1	エラー	***
	4—1	1	111			
	4—1	2	XYZ			
4—2	4—2	2	500	4—1	エラー	***
	4—2	6	80			
	4—2	7	$$$			

2) 経路のたぐり上げプログラム……プログラム経路を与えると，1)で作成した表を参照し，経路をたぐり上げてプログラム経路術語を作成する．

3) プログラム経路選出プログラム……2)の機能が充実し，人手の介入なしでほとんどの処理が可能となれば，さらに終止点を含む判定間経路から逆向きに成長していき，開始点に至るプログラム経路選出プログラムが考えられる．このプログラムは内部に2)を含み，恒等的に false となる場合は，自動的に代替経路を探していく．

4) 逆方向記号実行プログラム……2)の機能に出力データの内容のたぐり上げ機能を付加することにより，与えられたプログラム経路から，その出力を記号的に計算できるようになる．

5. おわりに

本稿ではホワイト・ボックス法の一つ “An Approach to Program Testing” に従って，テスト・データの作成の手順について述べたが，これらの支援プログラムの作成については，別の機会にゆずることにする．

本稿は，昭和58年度の RSDM (Reliable System/Software Development Method) テスト技法プロジェクトにおける検討の成果である．調査・検討の過程で助言をいただいた同プロジェクトの諸氏に感謝する．

参考文献 [1] J.C. Huang, “An Approach to Program Testing”, *Computing Surveys*, Vol. 7, No. 3, Sep. 1975, pp. 113-128.

[2] E.J. Weyuker, “The Oracle Assumption of Program Testing”, *Proc. 13th Hawaii International Conference on System Sciences*, Jan. 1980, pp. 44-49. [邦訳] “プログラム・テストにおける神託の仮説について”, 技報, 日本ユニバック(株), Vol. 2, FEB. 1982, pp. 57-63.

執筆者紹介　長谷川光邦 (Mitsukuni Hasegawa)
昭和 21 年生，44 年東京教育大学理学部卒．同年日本ユニバック(株)入社．汎用分類プログラムの開発，大学関連ＳＥサービス等を経て，現在労働省労働保険（雇用・労災・徴収）システムの開発に従事．

報告 LISP を用いて UNIVAC シリーズ1100上に作った4種の PROLOG処理系

Four PROLOG Systems in LISP on UNIVAC Series 1100 System

桑野龍夫

要 約 自然言語処理のための実験を UNIVAC シリーズ 1100 システムで行うため，UNIVAC シリーズ 1100 システム上に LISP を用いて作った異る4種の PROLOG の処理系について報告する．応用例として，英文から日本文への単純な翻訳の事例を示す．この事例では，コンパイルしたPROLOG を用いて構文解析の結果得た英文の中間表現を日本文の中間表現に変換し，日本文の生成を行っている．

Abstract Several PROLOG have been implemented on UNIVAC Series 1100 system as tools for conducting a couple of experiments on natural language processing. Four PROLOG systems, different in its strategies and implemented in LISP on UNIVAC Series 1100 System are reported in this paper. As an application, translation from English into Japanese has been tried utilizing compiled version of PROLOG and the result is described also, where an English sentence is syntactically analyzed, the resulting intermediate expression in English is transformed into Japanese intermediate expression and the corresponding Japanese sentence is generated.

1. はじめに

PROLOG のプログラムは一般に，

1) $p(\cdots) \leftarrow q1(\cdots)$ and $q2(\cdots)$ and $\cdots$ and $qn(\cdots)$

2) $p(\cdots) \leftarrow$

3) $\leftarrow q1(\cdots)$ and $q2(\cdots)$ and $\cdots$ and $qn(\cdots)$

のいずれかの節形である．この 1) 2) 3) をそれぞれ Rule, Fact, Goal と呼ぶ．また Rule と Fact をまとめて**主張** (assertion) とも呼ぶ．PROLOG の処理系では，**単一化** (unification) の過程で変数への経済的な代入を行う必要がある．また universal closure に対応して，互いに異なる**単一化代入** (unifier) を異なる代入として扱う必要がある．このための方策としては**変数再命名** (variable renaming) と **構造共有** (structure sharing) の二つを試みた．

また PROLOG の各処理系では，必要に応じ LISP 関数を用いることができるようにした．その方法として LISP 関数の EVAL 表現を，そのまま項または原子式の位置に書くことを許す**埋込み LISP 関数方式** (embedding LISP function) と，特別な式によって **LISP 関数を呼出す方式** (EVAL および CALL) との二つを試みた．以下の記述において PROLOG と LISP の知識を仮定する．

2. プログラムの記法

プログラムの3種の形 Rule, Fact, Goal は S 式を用いてそれぞれ次のように表す．ただし，P_i は述語名，T_{ij} は項を表し，$k, n, \cdots n_{kl}$ は整数で $k \geqq 2, n_1 \cdots n_k \geqq 0, l \geqq 1$ を満たす．

Rule: $((P_1\ T_{11}\ T_{12}\ \cdots\ T_{1n_1})\ (P_2\ T_{21}\ T_{22}\ \cdots T_{2n_2})$
$\vdots$
$(P_k\ T_{k1}\ T_{k2}\ \cdots\ T_{kn_k}))$

Fact : $((P_1\ T_{11}\ T_{12}\ \cdots\ T_{1n_1}))$

Goal : $?((P_1\ T_{11}\ T_{12}\ \cdots\ T_{1n_1})\ (P_2\ T_{21}\ T_{22}\ \cdots T_{2n_2})$
$\vdots$
$(P_l\ T_{k1}\ T_{k2}\ \cdots\ T_{kn_k}))$

たとえば，二つのリストを併せ，あるいは2分する述語 APPEND は次のように書ける．先頭に＊の付くものは変数 (variable) である．

① ((APPEND NIL ＊X ＊X))

② ((APPEND (＊A・＊X) ＊Y (＊A・＊Z)) (APPEND ＊X ＊Y ＊Z))

①は NIL と ＊X とを APPEND した結果が ＊X であることを示す．②は ＊X と ＊Y を APPEND して ＊Z となるなら，(＊A・＊X) と ＊Y を APPEND して (＊A・＊Z) となることを示す．リスト (A B) と (C D) とが APPEND できるかどうかを問う Goal は次のように書く．

③ ?((APPEND (A B) (C D) ＊L))

LISP 関数の呼出しを行うとき埋込み LISP 関数方式で行う系では，たとえば次のように書く．ただし，P は (NLAMBDA (X) (PRIN1 X)(TERPRI)), PLUS SUB1 および PRINT は通常の LISP 関数である．

④ ?((APPEND (A B) (C D) ＊L) (P ＊L))

⑤ ((SUM-TO 1 1)/)

⑥ ((SUM-TO ＊N (PLUS ＊N ＊RESULT1))(SUM-TO (SUB1 ＊N) ＊RESULT1))

⑦ ?((SUM-TO 3 ＊J) (PRINT ＊J))

④はリスト (A B) と (C D) とを APPEND した結果の ＊L の値 (A B C D) を関数 P の副作用により印刷する．⑤と⑥は正整数Nが与えられたとき，1からNまでの和が得られることを証明する述語 SUM-TO を定める．⑤は終了条件であり，／はカット・オペレータである．⑦は1から3までの和として ＊J を求めた後，LAMBDA 関数 PRINT の副作用により ＊J の値を印刷する．

リストの2分割を行うには⑧のような Goal を書く．分割されたリストのすべての組合せを得てこれを印刷するには⑨のように FAIL を使う．

⑧ ?((APPEND ＊J ＊K (A B C D))

⑨ ?((APPEND ＊J ＊K (A B C D)) (P ＊J) (P ＊K) FAIL)

一方，LISP 関数の呼出しを特別な式——CALL 式と EVAL 式——を用いて行う系では次のように書く．このときPは，(LAMBDA (X) (PRIN1 X) (TERPRI)) である．

⑩ ?((APPEND (A B) (C D) ＊L) (P ＊L))

⑪ ((SUM-TO 1 1)/)

⑫ ((SUM-TO ＊N ＊RESULT) (EVAL (SUB1 ＊N) ＊$N1$)
(SUM-TO ＊$N1$ ＊RESULT1)
(EVAL (PLUS ＊N ＊RESULT1) ＊RESULT))

⑬ ?((SUM-TO 3 ＊J) (P ＊J))

⑭ ((P ＊X) (CALL P ＊X)/)

⑫では (SUB1 ＊N) の値が ＊$N1$ に単一化され (PLUS ＊N ＊RESULT1) の値が ＊RESULT に単一化される．⑭は述語名Pが関数Pを呼ぶように定義している．

なお，次に PROLOG の構文法 (syntax) を BNF 風にまとめる．

埋込み LISP 関数方式を用いる PROLOG Opus 1.02, Opus 2.02 の構文則

(PROLOG syntax for PROLOG Opus 1.02 and Opus 2.02, where embedding LISP functions are allowed)

```
<term>::=<constant>|<variable>|(<list of terms>)|
         (<function name><ares>)
  <constant>::=<literal atom>
  <variable>::=*|<literal atom begining with *>
  <list of terms>::=<term>|<term><list of terms>|
                          <list of terms>·<list of terms>
<atomic formula>::=(<predicate name><list of terms>)
  <predicate name>::=<constant>
<assertion>::=(<atomic formula>)|
              (<atomic formula><list of elementary formula>)
  <list of elementary formula>::=<elementary formula>|
           <elementary formula><list of elementary formula>
    <elementary formula>::=atomic formula>|
              (<function name><args>)|/|FAIL
<goal>::=?(<elementary formula>)|
         ?(<list of elementary formula>)
```

CALL と EVAL を用いる PROLOG Opus 1.03, Opus 3.02 の構文則での変更点

(Changes in PROLOG syntax for PROLOG Opus 1.03 and Opus 3.02, where CALL and EVAL are the exclusive vehicles to invocate LISP functions)

```
<term>::=<constant>|<variable>|(<list of terms>)
<elementary formula>::=<atomic formula>|/|FAIL|
           (CALL <function name><args>)|
           (EVAL (<function name><args>)<variable>)
```

3. PROLOG 処理系の動作

関数 PROLOG を呼び出すと，PROLOG 処理系が起動し，「WELCOME TO LOGIC WORLD OF PROLOG OPUS X.XX」改行後，「SHINICHI YAMADA, MAY 1984」他のメッセージを印刷し，その後入力待ちの状態となる．このとき，QT(Quit), T(Trace), U(Untrace), ⟨assertion⟩, ⟨goal⟩ および LISP の表現が入力できる．まず，QT と入力すると，PROLOG は終了する．QT 以外の上記各項を入力すると，その処理の後に再びこの入力待ちの状態となる．T と入力すると PROLOG の推論過程をトレース (trace) するモードが設定される．U と入力するとこのモードが解除 (untrace) される．？を入力すると？の次の入力を Goal として扱う．これは関数 REFUTE (Opus 1.02 および Opus 1.03 の場合) を呼ぶか，関数 REFUTE-GOAL (Opus 2.02 および Opus 3.02 の場合) を呼んで処理する．Rule または Fact を入力すると，これを定義する処理を行う．このため関数 DEFINE-PROC を呼ぶ．LISP の表現はそのまま評価する．

Rule と Fact の定義は述語名を鍵として行う．Rule と Fact は述語名のアトムの属性

リストに写像される．同一述語名で複数の Rule ないし Fact を定義するときは，最初の定義の後に第2の定義が入れられ，以後もつぎつぎと新たな定義が後に入れられる．推論の際，この定義が使われるときは積まれた定義が（PROLOG における線形の探策ルール (search rule) に従って）前から一つづつ順次取り出される．たとえば，APPEND の定義では

((APPEND NIL $*X$ $*X$))

が入力されるとアトム APPEND の属性インディケータ DICT の属性リストが

(((APPEND NIL $*X$ $*X$)))

とセットされる．つぎに APPEND の第2の定義

((APPEND ($*A$ · $*X$) $*Y$ ($*A$ · $*Z$)) (APPEND $*X$ $*Y$ $*Z$))

が入力されると APPEND の属性リストは

(((APPEND NIL $*X$ $*X$))
((APPEND ($*A$ · $*X$) $*Y$ ($*A$ · $*Z$)) (APPEND $*X$ $*Y$ $*Z$)))

とセットされる．関数 DEFINE-PROC は，定義を一つ行うたびにメッセージ ACCEPTED を印刷する．

REFUTE (Opus 1.02, 1.03) または REFUTE-GOAL (Opus 2.02, 3.02) で始まる推論では Rule および Fact が構成する探索空間を Depth-First and Left-to-Right 法に従って探索する．次のような Assertion とGoal を仮定する．

((P1…)(P11…)…(P1m_1…))
((P2…)(P21…)…(P2m_2…))
⋮　⋮　⋮
((Pn…)(Pn1…)…(Pnm_n…))
?((P…)(R_1…)…(R_r…))

ただし，どの Assertion も Subgoal P とマッチする，すなわち P と P1～Pn は同じ述語名であると仮定する．

この Goal について推論するとき，Goal の中の個々の Subcoal の推論は左から右 (Left-to-Right) へ順次実行される．Subgoal (P…)は，まず最初の Assertion ((P1…)(P11…)…(P1m_1…)) の頭部 (P1…) とマッチするので，残りの Assertion ((P2…)…)…((Pn…)…) は将来のバックトラックに備えてスタック LASTBACK にプッシュされると共に，現在の Goal は Depth-First 探索を行うために，

((P11…)…(P1m_1…)(R_1…)…(R_r…))Q

となる．ただし，Q は (P1…) と (P…) との Most General Unifier (MGU) である．

つぎに (P…) の代りに Subgoal (P11…)Q の推論が行われる．一つ一つの Subgoal の推論が成功していく限り，すべての Subgoal の推論が終わるまで，上のプロセスが繰り返される．(P11…) が失敗した場合は，(P…) と ((P2…)(P21…)…(P2m_2…)) の頭部 (P2…) とがマッチし同様の推論が試みられる．

4. 変数の持ち方

Horn 節の universal closure の約束から Rule, Fact および Goal の中の変数のスコープは，その Rule, Fact および Goal の中に限定される．したがって，推論の過程で Goal から次々と Rule や Fact を呼び出すとき，表記が同じ変数であっても異なる変数として扱う必要がある．変数再命名 (variable renaming) では単一化する Horn 節内の

(環境番号 1	(変数 11	スケルトン 11)	(環境 1)
	(変数 12	スケルトン 12)	(環境 1)
	⋮	⋮	⋮
	(変数 1n	スケルトン 1n)	(環境 1))
(環境番号 2	(変数 21	スケルトン 21)	(環境 2)
	⋮	⋮	⋮
	(変数 2m	スケルトン 2m)	(環境 2))
⋮			
(環境番号 N	(変数 $N1$	スケルトン $N1$)	(環境N)
	⋮	⋮	⋮
	(変数 Nk	スケルトン Nk)	(環境N))

環境概念図

Concept of environment

universal 変数に異なる変数名を割り当てる．構造共有 (structure sharing) では変数とその値が正しく表現されることを保証するために，変数に対し特定のスケルトンおよび環境を指定する．上図は一連の単一化の後，作られる環境の概念図である．

図では同じ環境が繰返して示されているけれども，これはもっぱら図示の都合によるものであり，実際には環境へのポインタが保持され，同一の環境はただ一つだけ存在する．したがって，変数の値を定める環境は，変数が複数ある場合に共有される．

5. 処理系のアルゴリズム

各 PROLOG 処理系のアルゴリズムを以下に示す．また処理系 Opus 3.02 の LISP プログラムだけを添付する．

PROLOG Opus 1.02

```
(PROLOG)
print message WELCOME TO LOGIC WORLD OF PROLOG OPUS 1.03
              SHINICHI YAMADA,MAY 1984
OFF, RESULT, TRACE-SET:=NIL
LOOP
INPUT:=(READ)
  if INPUT=QT then OFF:=T and RESULT:=NIL
  if INPUT=T  then TRACE-SET:=T and RESULT:=TRACING
  if INPUT=U then TRACE-SET:=NIL and RESULT:=UNTRACING
  if INPUT=? then RESULT:=(REFUTE(READ)NIL NIL)
  if (CAR INPUT) is ATOM then evaluate INPUT
  if (CAR INPUT) is LIST then RESULT:=(DEFINE-PROC INPUT)
  otherwise RESULT:=IGNORED
  if OFF=T then return QUIT
  if RESULT=NIL then print message FAIL
  otherwise print the value of RESULT
  go LOOP

(DEFINE-PROC INPUT)
if(GET(CAAR INPUT)"DICT)=NIL then DFN:=NIL
  otherwise DFN:=(GET(CAAR INPUT)"DICT)
if DFN=NIL then (PUT(CAAR INPUT)"DICT(LIST INPUT))
  otherwise (NCONC DFN(LIST INPUT))
return ACCEPTED
```

```
(REFUTE with GOAL=(READ), CURR-ENV=NIL, GOAL-STACK=NIL)
PROCS, CURR-PROC, CUTBACK, BACK, LASTBACK:=NIL
LOOP
if GOAL=NIL then
  if GOAL-STACK=NIL then return QED
  otherwise GOAL:=(CAR GOAL-STACK) with its instantiated
            variables replaced by the corresponding values
            GOAL-STACK:=(CDR GOAL-STACK)
            CUTBACK:=(CDR CUTBACK)
            go LOOP
  if (CAR GOAL) is ATOM then
     if (CAR GOAL)=/    then LASTBACK:=(CAR CUTBACK)
                             GOAL:=(CDR GOAL)
                             go LOOP
     if (CAR GOAL)= FAIL then go BACKTRACK
     otherwise print message PROC (CAR GOAL) IS UNDEFINED
                  return NIL
  PROCS:=(GET (CAAR GOAL)"DICT)
  if PROCS=NIL then
     if (CAR GOAL) is a LISP expression then
        evaluate (CAR GOAL)
        GOAL:=(CDR GOAL)
        go LOOP
     otherwise print message PROC (CAAR GOAL) IS UNDEFINED
               return NIL
  NEXT
  if the PROC is the last procedure then
     CURR-PROC:=(RENAME(CAR PROCS))
     if unification is successful such that (CAR GOAL) is
        instantiated to (CAR CURR-PROC) then
        CUTBACK:=(CONS LASTBACK CUTBACK)
        GOAL-STACK:=(CONS(CDR GOAL) GOAL-STACK
        GOAL:=(CDR CURR-PROC) with its instantiated variables
                              replaced by corresponding values
        go LOOP
     otherwise go BACKTRACK
  BACK:=LASTBACK
  LASTBACK:=(CONS(LIST GOAL GOAL-STACK CUTBACK(CDR PROCS)
            CURR-ENV) LASTBACK)
  CURR-PROC:=(RENAME(CAR PROCS))
  if unification is successful such that (CAR GOAL) is
       instantiated to (CAR CURR-PROC) then
       CUTBACK:=(CONS BACK CUTBACK)
       GOAL-STACK:=(CONS (CDR GOAL) GOAL-STACK)
       GOAL:=(CDR CURR-PROC) with its instantiated variables
       replaced by corresponding values
       go LOOP
    otherwise go BACKTRACK
```

```
BACKTRACK
if LASTBACK=NIL then return FAILED
otherwise
   GOAL-STACK:=(CADAR LASTBACK)
   CUTBACK:=(CADDAR LASTBACK)
   PROCS:=(CADDDAR LASTBACK)
   CURR-ENV:=(CADDDDAR LASTBACK)
   GOAL:=(CAAR LASTBACK) with its instantiated variables
   replaced by corresponding values.
   LASTBACK:=(CDR LASTBACK)
   go NEXT
```

PROLOG Opus 1.03

```
(PROLOG)
print message WELCOME TO LOGIC WORLD PROLOG OPUS 1.02
              SHINICHI YAMADA, MAY 1984
OFF, RESULT, TRACE-SET:=NIL
LOOP
INPUT:=(READ)
  if INPUT=QT then OFF:=T and RESULT:=NIL
  if INPUT=T  then TRACE-SET:=T and RESULT:=TRACING
  if INPUT=U  then TRACE-SET:=NIL and RESULT:=UNTRACING
  if INPUT=?  then RESULT:=(REFUTE(READ)NIL NIL)
  if (CAR INPUT) is ATOM then evaluate INPUT
  if (CAR INPUT) is LIST then RESULT:=(DEFINE-PROC INPUT)
  otherwise RESULT:=IGNORED
  if OFF=T then return QUIT
  if RESULT=NIL then print message FAIL
  otherwise print the value of RESULT
  go LOOP

(DEFINE-PROC INPUT)
  if (GET(CAAR INPUT)"DICT)=NIL then DFN:=NIL
     otherwise DFN:=(GET(CAAR INPUT)"DICT)
  if DFN=NIL then (PUT(CAAR INPUT)"DICT (LIST INPUT))
     otherwise (NCONC DFN (LIST INPUT))
  return ACCEPTED

(REFUTE with GOAL=(READ), CURR-ENV=NIL, GOAL-STACK=NIL)
PROCS, CURR-PROC, CUTBACK, BACK, LASTBACK:=NIL
LOOP
if GOAL=NIL then
   if GOAL-STACK=NIL then return QED
   otherwise GOAL:=(CAR GOAL-STACK) with its instantiated
          variables replaced by the corresponding values
```

```
            GOAL-STACK:=(CDR GOAL-STACK)
            CUTBACK:=(CDR CUTBACK)
            go LOOP
if (CAR GOAL)is ATOM then
  if (CAR GOAL) is / then LASTBACK:=(CAR CUTBACK)
                          GOAL:=(CDR GOAL)
                          go LOOP
  if (CAR GOAL) is FAIL then go BACKTRACK
  otherwise print message PROC(CAR GOAL) IS UNDEFINED
            return NIL
if (CAAR GOAL) is EVAL then
  if unification is successful such that (CADDAR GOAL) is
     instantiated to the value obtained by evaluating
     the LISP expression then
     GOAL:=(CDR GOAL)
     go LOOP
  otherwise go BACKTRACK
if (CAAR GOAL) is CALL then
   evaluate the LISP expression soliciting the side-effect
   GOAL:=(CDR GOAL)
   go LOOP
PROCS:=(GET(CAAR GOAL)"DICT)
if PROCS=NIL then print message PROC (CAAR GOAL) IS UNDEFINED
                  return NIL
NEXT
if the PROC is the last procedure then
   CURR-PROC:=(RENAME(CAR PROCS))
   if unification is successful such that (CAR GOAL) is
                  instantiated to (CAR CURR-PROC) then
                  CUTBACK:=(CONS LASTBACK CUTBACK)
                  GOAL-STACK:=(CONS(CDR GOAL)GOAL-STACK)
                  GOAL:=(CDR CURR-PROC)with its
                  instantiated variables replaced by
                  corresponding values
                  go LOOP
   otherwise go BACKTRACK
BACK:=LASTBACK
LASTBACK:=(CONS(LIST GOAL GOAL-STACK CUTBACK(CDR PROCS)
           CURR-ENV) LASTBACK)
CURR-PROC:=(RENAME(CAR PROCS))
if unification is successful such that (CAR GOAL) is
                  inatantiated to (CAR CURR-PROC) then
                  CUTBACK:=(CONS BACK CUTBACK)
                  GOAL-STACK:=(CONS(CDR GOAL)GOAL-STACK)
                  GOAL:=(CDR CURR-PROCS) with its instantiated
                  variables replaced by corresponding values
                  go LOOP
otherwise go BACKTRACK
```

```
BACKTRACK
if LAST BACK=NIL then return FAIL
otherwise GOAL-STACK:=(CADAR LASTBACK)
          CUTBACK:=(CADDAR LASTBACK)
          PROCS:=(CADDDAR LASTBACK)
          CURR-ENV:=(CADDDDAR LASTBACK)
          GOAL:=(CAAR LASTBACK) with its instantiated
          variables replaced by corresponding values
          LASTBACK:=(CDR LASTBACK)
          go NEXT
```

PROLOG Opus 2.02

```
(PROLOG)
print message WELCOME TO LOGIC WORLD OF PROLOG OPUS 2.02
              SHINICHI YAMADA,MAY 1984
OFF, RESULT, TRACE-SET:=NIL
LOOP
INPUT:=(READ)
  if INPUT=QT then OFF:=T and RESULT:=NIL
  if INPUT=T  then TRACE-SET:=T and RESULT:=TRACING
  if INPUT=U  then TRACE-SET:=NIL and RESULT:=UNTRACING
  if INPUT=?  then INDEX:=NIL
              INPUT:=(READ)
              examine if there are unregistered
              variables within INPUT and register if any
              RESULT:=(REFUTE(CONS INPUT(EMPTY-ENV))NIL)
  if (CAR INPUT) is ATOM then evaluate INPUT
  if (CAR INPUT) is LIST then
              examine if there are unregistered
              variables within INPUT and register if any
              RESULT:=(DEFINE-PROC INPUT)
  otherwise RESULT:=IGNORED
  if OFF=T then return QUIT
  if RESULT=NIL then print message FAIL
  otherwise print the value of RESULT

(DEFINE-PROC INPUT)
if(GET(CAAR INPUT)"DICT)=NIL then DFN:=NIL
  otherwise DFN:=(GET(CAAR INPUT)"DICT)
if DFN=NIL then (PUT(CAAR INPUT)"DICT(LIST INPUT))
  otherwise (NCONC DFN(LIST INPUT))
return ACCEPTED

(REFUTE with GOAL-EXP=(CONS INPUT(EMPTY-ENV))
            GOAL-STACK=NIL)
GOAL,GOALENV,PROCS,PROCENV,CUTBACK,BACK,LASTBACK,TRAIL:=NIL
OUTLOOP
  GOAL:=(CAR GOAL-EXP)
  GOALENV:=(CDR GOAL-EXP)
```

```
INLOOP
  if GOAL=NIL then
     prepare for deeper exploration in seach space
     go LOOP
  otherwise return QED
  if(CAR GOAL)is CUT symbol then
     prepare for cutting short the backtrack path
     GOAL:=(CDR GOAL)
     go INLOOP
  if(CAR GOAL)is FAIL then go BACKTRACK
  otherwise print message PROC (CAR GOAL) IS UNDEFINED
            return NIL
  PROCS:=(GET(CAAR GOAL)"DICT)
  if PROCS=NIL then
     if(CAR GOAL)is a LISP expression then
        evaluate(CAR GOAL)
        GOAL:=(CDR GOAL)
        go INLOOP
     otherwise print message PROC (CAAR GOAL) IS UNDEFINED
               return NIL
NEXT
  if the PROC is the last procedure then
     assign the next environment identifier
     if unification is successful such that (CAR GOAL) with
        its instantiated variables replaced by corresponding
        values and evaluated provided that it is a LISP
        expression is instantiated to (CAAR PROCS) then
        push the current environment onto the appropriate
        stacks for future backtracking
        prepare for deeper exploration in search space
        go INLOOP
  otherwise go BACKTRACK
PROCENV:=(EMPTY-ENV)
BACK:=LASTBACK
LASTBACK:=(CONS(LIST(CONS GOAL GOALENV)GOAL-STACK CUTBACK
                    (CDR PROCS)TRAIL)
                LASTBACK)
if unification is successful such that (CAR GOAL) with its
   instantiated variables replaced by corresponding values
   and evaluated provided that it is a LISP expression is
   instantiated to (CAAR PROCS) then
   push the current environment onto the appropriate stacks
   for future backtracking
   prepare for deeper exploration in search space
   go INLOOP
 otherwise go BACKTRACK
BACKTRACK
  if LASTBACK=NIL then return NIL
  otherwise resume information from the closest backtrack node
            abandon the parts of instantiation generated
            during the unification resulted in FAIL
            go NEXT
```

PROLOG Opus 3.02

```
(PROLOG)
print message WELCOME TO LOGIC WORLD OF PROLOG OPUS 3.02
        SHINICHI YAMADA,MAY 1984
OFF, RESULT, TRACE-SET:=NIL
LOOP
INPUT:=(READ)
  if INPUT=QT then OFF:=T and RESULT:=NIL
  if INPUT=T  then TRACE-SET:=T and RESULT:=TRACING
  if INPUT=U then TRACE-SET:=NIL and RESULT:=UNTRACING
  if INPUT=? then INDEX:=NIL
                  INPUT:=(READ)
                  examine if there are unregistered variables
                  within INPUT and register if any
                  RESULT:=(REFUTE(CONS INPUT(EMPTY-ENV)) NIL)
  if (CAR INPUT) is ATOM then evaluate INPUT
  if (CAR INPUT) is LIST then
                  examine if there are unregistered variables
                  within INPUT and register if any
                  RESULT:=(DEFINE-PROC INPUT)
  otherwise RESULT:=IGNORED
  if OFF=T then return QUIT
  if RESULT=NIL then print message FAIL
  otherwise print the value of RESULT
  go LOOP

(DEFINE-PROC INPUT)
if(GET(CAAR INPUT)"DICT)=NIL then DFN:=NIL
  otherwise DFN:=(GET(CAAR INPUT)"DICT)
if DFN=NIL then (PUT(CAAR INPUT)"DICT(LIST INPUT))
  otherwise (NCONC DFN(LIST INPUT))
return ACCEPTED
(REFUTE with GOAL-EXP=(CONS INPUT (EMPTY-ENV))
             GOAL-STACK=NIL )
GOAL,GOALENV,PROCS,PROCENV,CUTBACK,BACK,LASTBACK,TRAIL:=NIL
OUTLOOP
  GOAL:=(CAR GOAL-EXP)
  GOALENV:=(CDR GOAL-EXP)
```

```
INLOOP
  if GOAL=NIL then
    if GOAL-STACK is not NIL then
       prepare for deeper exploration in search space
       go OUTLOOP
    otherwise return QED
  if (CAR GOAL) is /···the CUT symbol···then
    prepare for cutting short the backtrack path
    GOAL:=(CDR GOAL)
    go INLOOP
  if (CAR GOAL) IS FAIL then go BACKTRACK
  if (CAR GOAL) is ATOM other than/and FAIL then return NIL
  if (CAAR GOAL) is EVAL then
    if unification is successful such that (CADDAR GOAL) is
      instantiated to the value obtained by evaluating the
      LISP expression, then
      GOAL:=(CDR GOAL)
      go INLOOP
    otehrwise go BACKTRACK
  if (CAAR GOAL) is CALL then
    evaluate the LISP expression solicitting the side-effect
    GOAL:=(CDR GOAL)
    go INLOOP
  PROCS:=(GET (CAAR GOAL) "DICT)
  if PROCS=NIL then return NIL producing error message
                    PROC(CAAR GOAL) is UNDEFINED
NEXT
  if the PROCS is the last one then
  assign the next environment identifier
  if unification is successful such that (CAR GOAL) is
     instantiated to (CAAR PROCS) then
     push the current environment onto the approrriate stacks
     for future backtracking
     prepare for deeper exploration in search space
     go INLOOP
  othewise go BACKTRACK
PROCENV:=(EMPTY-ENV)
BACK:=LASTBACK
LASTBACK:=(CONS(LIST(CONS GOAL GOALENV) GOAL-STACK CUTBACK
                    (CDR PROCK) TRAIL)
               LASTBACK)
  if unification is succesful such that (CAR GOAL) is
                    instantiated to (CAAR PROCS) then
                    CUTBACK:=(CONS BACK CUTBACK)
                    GOAL-STACK:=(CONS(CONS(CDR GOAL) GOALENV)
                                     GOAL-STACK)
                    GOAL:=(CDAR PROCS)
                    GOALENV:=PROCENV
                    go INLOOP
  otherwise go BACKTRACK
```

```
BACKTRACK
  if LASTBACK=NIL then return NIL
  otherwise resume information from the closest backtrack node
            abandon the parts of instantiation generated
            during  the unification resulted in FAIL
            go NEXT
```

PROLOG Opus 3.02 の LISP プログラム

```
CR(OP3FNS)
SETQ(OP3FNS
(DEFINEQ
(PROLOG
[LAMBDA (X)
  (PROG (OFF RESULT TRACE-SET PREDLIST VARLIST)
        (DECLARE (FLUID OFF TRACE-SET PREDLIST VARLIST))
        (PRIN1 "'WELCOME TO LOGIC WORLD OF PROLOG OPUS 3.02')
        (TERPRI)
        [COND
          (X (PRIN1 "'STRUCTURE SHARING, CUT/FAIL, CALL/EVAL AND DEVISED VARP']
        (TERPRI)
        (PRIN1 "'SHINICHI YAMADA,MAY 1984')
        (TERPRI)
        (SETQ OFF NIL)
    LOOP(SETQ RESULT (EXEC))
        [COND
          (OFF (RETURN "QUIT]
        (COND
          ((NULL RESULT)
            (PRINT "FAIL))
          (T (PRINT RESULT)))
        (GO LOOP])
(EXEC
[LAMBDA NIL
  (PROG (INPUT INDEX)
        (DECLARE (FLUID INDEX))
        (COND
          ((EQ (SETQ INPUT (READ))
               "QT)
            (SETQ OFF T)
            (RETURN NIL))
          ((EQ INPUT "T)
            (SETQ TRACE-SET T)
            (RETURN "TRACING))
          ((EQ INPUT "U)
            (SETQ TRACE-SET NIL)
            (RETURN "UNTRACING))
          ((EQ INPUT "?)
            (SETQ INDEX 0)
            (PUT-VARP (SETQ INPUT (READ)))
            (RETURN (REFUTE (CONS INPUT (EMPTY-ENV))
                            NIL)))
          [(ATOM (CAR INPUT))
            (RETURN (EVAL INPUT (ALIST]
          ((LISTP (CAR INPUT))
            (PUT-VARP INPUT)
            (RETURN (DEFINE-PROC INPUT)))
          (T (RETURN "IGNORED])
(DEFINE-PROC
[LAMBDA (PROC)
  (PROG (DFN)
        (SETQ DFN (GET (CAAR PROC)
                       "DICT))
        [COND
          ((NULL DFN)
          [PUT "PREDLIST "VALIDPRED
               (CONS (CAAR PROC)
                     (GET "PREDLIST "VALIDPRED]
          (PUT (CAAR PROC)
               "DICT
               (LIST PROC)))
          (T (NCONC DFN (LIST PROC]
      (RETURN "ACCEPTED])
```

```
(REFUTE
[LAMBDA (GOAL-EXP GOAL-STACK)
  (DECLARE (FLUID GOAL-EXP GOAL-STACK))
  (PROG (GOAL GOALENV PROCS PROCENV CUTBACK BACK LASTBACK TRAIL)
        (DECLARE (FLUID GOAL GOALENV PROCS PROCENV CUTBACK BACK LASTBACK TRAIL))
    OUTLOOP
        (SETQ GOAL (CAR GOAL-EXP))
        (SETQ GOALENV (CDR GOAL-EXP))
    INLOOP
        [COND
          ((NULL GOAL)
            (COND
              ((NOT (NULL GOAL-STACK))
                (DESCEND)
                (GO OUTLOOP))
              (T (RETURN "QED]
        (COND
          (TRACE-SET (TRACE-E)))
        (COND
          [(ATOM (CAR GOAL))
            (COND
              ((EQ "/ (CAR GOAL))
                (SETQ LASTBACK (CAR CUTBACK))
                (SETQ GOAL (CDR GOAL))
                (GO INLOOP))
              ((EQ "FAIL (CAR GOAL))
                (GO BACKTRACK))
              (T (PRINT (LIST "PROC (CAR GOAL)
                              "IS "UNDEFINED))
                 (RETURN NIL]
          [(EQ "EVAL (CAAR GOAL))
            (COND
              ((GETVAL)
                (COND
                  (TRACE-SET (TRACEINFO)))
                (SETQ GOAL (CDR GOAL))
                (GO INLOOP))
              (T (GO BACKTRACK]
          ((EQ "CALL (CAAR GOAL))
            (SIDEEFFECT)
            (COND
              (TRACE-SET (TRACEINFO)))
            (SETQ GOAL (CDR GOAL))
            (GO INLOOP)))
        (SETQ PROCS (GET (CAAR GOAL)
                         "DICT))
        (COND
          ((NULL PROCS)
            (PRINT (LIST "PROC (CAAR GOAL)
                         "IS "UNDEFINED))
            (RETURN NIL)))
    NEXT[COND
          ((NULL (CDR PROCS))
            (SETQ PROCENV (EMPTY-ENV))
            (COND
              (TRACE-SET (TRACE-W)))
            (COND
              ((INSTANTIATE)
                (COND
                  (TRACE-SET (TRACEINFO)))
                (PUSHBTA)
                (GO INLOOP))
              (T [COND
                   (TRACE-SET (PRINT "FAILED:]
                 (GO BACKTRACK]
        (REPOSITION)
        (COND
          (TRACE-SET (TRACE-W)))
        (COND
          ((INSTANTIATE)
            (COND
              (TRACE-SET (TRACEINFO)))
            (PUSHBTB)
            (GO INLOOP))
          (T [COND
               (TRACE-SET (PRINT "FAILED:]
             (GO BACKTRACK)))
```

```
    BACKTRACK
        (COND
          ((NULL LASTBACK)
            (RETURN NIL))
          (T (POPBT)
             (COND
               (TRACE-SET (TRACE-E)))
             (GO NEXT])
(DESCEND
[LAMBDA NIL (SETQ GOAL-EXP (CAR GOAL-STACK))
  (SETQ GOAL-STACK (CDR GOAL-STACK))
  (SETQ CUTBACK (CDR CUTBACK])
(GETVAL
[LAMBDA NIL
  (UNIFY (CONS [APPLY (REDUCE (CONS (CAADAR GOAL)
                                    GOALENV))
                      (MAPCAR (CDADAR GOAL)
                              "(LAMBDA (X)
                                 (REDUCE (CONS X GOALENV]
               GOALENV)
         (CONS (CADDAR GOAL)
               GOALENV])
(TRACE-E
[LAMBDA NIL (PRINT "ENTER)
  (PRINT (REDUCE (CONS GOAL GOALENV])
(TRACE-W
[LAMBDA NIL (PRINT "WITH;)
  (PRINT (CAR PROCS])
(TRACEINFO
[LAMBDA NIL (PRINT "SUCCEEDED:)
  (PRINT (REDUCE (CONS (CAR GOAL)
                       GOALENV])
(SIDEEFFECT
[LAMBDA NIL
  (APPLY (REDUCE (CONS (CADAR GOAL)
                       GOALENV))
         (MAPCAR (CDDAR GOAL)
                 "(LAMBDA (X)
                    (REDUCE (CONS X GOALENV])
(INSTANTIATE
[LAMBDA NIL (UNIFY (CONS (CAR GOAL)
                         GOALENV)
                   (CONS (CAAR PROCS)
                         PROCENV])
(PUSHBTA
[LAMBDA NIL (SETQ CUTBACK (CONS LASTBACK CUTBACK))
  (SETQ GOAL-STACK (CONS (CONS (CDR GOAL)
                               GOALENV)
                         GOAL-STACK))
  (SETQ GOAL (CDAR PROCS))
  (SETQ GOALENV PROCENV])
(REPOSITION
[LAMBDA NIL (SETQ PROCENV (EMPTY-ENV))
  (SETQ BACK LASTBACK)
  (SETQ LASTBACK
    (CONS (LIST (CONS GOAL GOALENV)
                GOAL-STACK CUTBACK
                (CDR PROCS)
                TRAIL)
          LASTBACK])
(PUSHBTB
[LAMBDA NIL (SETQ CUTBACK (CONS BACK CUTBACK))
  (SETQ GOAL-STACK (CONS (CONS (CDR GOAL)
                               GOALENV)
                         GOAL-STACK))
  (SETQ GOAL (CDAR PROCS))
  (SETQ GOALENV PROCENV])
(POPBT
[LAMBDA NIL (UNDO TRAIL (CADDDDAR LASTBACK))
  (SETQ TRAIL (CADDDDAR LASTBACK))
  (SETQ GOAL (CAAAR LASTBACK))
  (SETQ GOALENV (CDAAR LASTBACK))
  (SETQ GOAL-STACK (CADAR LASTBACK))
  (SETQ CUTBACK (CADDAR LASTBACK))
  (SETQ PROCS (CADDDAR LASTBACK))
  (SETQ LASTBACK (CDR LASTBACK])
```

```
(UNIFY
[LAMBDA (X Y)
  (COND
    ((EQUAL X Y)
      T)
    ((VARP (CAR X))
      (COND
        ((ISBOUND X)
          (UNIFY (XVALUE X)
                 Y))
        (T (XBIND X Y)
           T)))
    ((VARP (CAR Y))
      (UNIFY Y X))
    ((ATOM (CAR X))
      (EQUAL (CAR X)
             (CAR Y)))
    ((ATOM (CAR Y))
      NIL)
    [(UNIFY (CONS (CAAR X)
                  (CDR X))
            (CONS (CAAR Y)
                  (CDR Y)))
      (UNIFY (CONS (CDAR X)
                   (CDR X))
             (CONS (CDAR Y)
                   (CDR Y]
    (T NIL])
(VARP
[LAMBDA (X)
  (AND (ATOM X)
       (NOT (NUMBERP X))
       (GET X "VARP])
(OUT-VARP
[LAMBDA (X)
  (COND
    ((NUMBERP X)
      NIL)
    ((ATOM X)
      (EQUAL (CAR (DECAT X))
             "*))
    (T NIL])
(PUT-VARP
[LAMBDA (X)
  (COND
    ((OUT-VARP X)
      [PUT VARLIST "VALIDVAR (CONS X (GET VARLIST "VALIDVAR]
      (PUT X "VARP T))
    ((ATOM X)
      NIL)
    (T (PUT-VARP (CAR X))
       (PUT-VARP (CDR X])
(EMPTY-ENV
[LAMBDA NIL (SETQ INDEX (ADD1 INDEX))
  (LIST (LIST INDEX])
(XVALUE
[LAMBDA (X)
  (COND
    [(VARP (CAR X))
      ([LAMBDA (V)
          (DECLARE (FLUID V))
          (COND
            ((NULL V)
              X)
            (T (XVALUE (CONS (CADR V)
                             (CDDR V]
        (FASSOC (CAR X)
                (CDDR X]
    (T X])
(ISBOUND
[LAMBDA (X)
  (FASSOC (CAR X)
          (CDDR X])
(XBIND
[LAMBDA (X Y)
  (SETQ TRAIL
    (CONS (RPLACD (CDR X)
                  (CONS (CONS (CAR X)
                              (XVALUE Y))
                        (CDDR X)))
```

```
          TRAIL])
(UNDO
[LAMBDA (X Y)
  (COND
    ((EQ X Y)
      NIL)
    ((NULL X)
      (PRINT "EMPTY-TRAIL)
      T)
    (T (RPLACD (CAR X)
               (CDDAR X))
       (UNDO (CDR X)
             Y])
(REDUCE
[LAMBDA (X)
  (COND
    [(VARP (CAR X))
      ([LAMBDA (V)
          (COND
            ((NULL V)
              (CAR X))
            (T (REDUCE (CONS (CADR V)
                             (CDDR V]
        (FASSOC (CAR X)
                (CDDR X]
    ((ATOM (CAR X))
      (CAR X))
    (T (CONS (REDUCE (CONS (CAAR X)
                           (CDR X)))
             (REDUCE (CONS (CDAR X)
                           (CDR X])
(ALLPRED
[LAMBDA NIL (SETQ CONTENTS (GET "PREDLIST "VALIDPRED))
  (COND
    ((NULL CONTENTS)
      "VACANT)
    (T CONTENTS])
(SHOW
[NLAMBDA (X)
  (COND
    ((GET X "DICT))
    (T "UNDEFINED])
(SHOWM
[NLAMBDA (PRED M)
  (SETQ AUXVAL (SHOWM-AUX (GET PRED "DICT)
                          M))
  (COND
    ((NULL AUXVAL)
      (PRIN1 "'FEWER ASSERTIONS READY FOR ACCESS')
      (TERPRI))
    (T (PRETTYPRINT AUXVAL)
       (TERPRI)
       (PRIN1 "'IF DELETION PREFERRED , INPUT Y, OTHERWISE INPUT N')
       (TERPRI)
       (COND
         ((EQ (READ)
              "Y)
           (DEM1 PRED M))
         (T "UNCHANGED])
(SHOWM-AUX
[LAMBDA (PROPERTY CT)
  (COND
    ((NULL PROPERTY)
      NIL)
    ((ZEROP (SUB1 CT))
      (CAR PROPERTY))
    (T (SHOWM-AUX (CDR PROPERTY)
                  (SUB1 CT])
(ADM
[NLAMBDA (CLAUSE M)
  (SETQ PROPERTY (GET (CAAR CLAUSE)
                      "DICT))
  (PUT (CAAR CLAUSE)
       "DICT
       (CDR (ADM-AUX (CONS NIL NIL)
                     PROPERTY CLAUSE M])
```

```
(ADM-AUX
[LAMBDA (RESVR ORILIST CLAUSE CT)
  (COND
    ((ZEROP (SUB1 CT))
      (NCONC RESVR (NCONC (LIST CLAUSE)
                          ORILIST)))
    (T (ADM-AUX (NCONC RESVR (LIST (CAR ORILIST)))
                (CDR ORILIST)
                CLAUSE
                (SUB1 CT]))
(ADMR
[NLAMBDA (CLAUSE M)
  [SETQ PROPERTY (REVERSE (GET (CAAR CLAUSE)
                               "DICT]
  (PUT (CAAR CLAUSE)
       "DICT
       (REVERSE (CDR (ADM-AUX (CONS NIL NIL)
                              PROPERTY CLAUSE M]))
(ERS
[NLAMBDA (X)
  (ERS-AUX X]))
(ERS-AUX
[LAMBDA (X)
  (COND
    ((GET X "DICT)
      (REMPROP X "DICT)
      [PUT "PREDLIST "VALIDPRED
           (DELEMENT X (GET "PREDLIST "VALIDPRED]
      "-ERASED-)
    (T "DEFINITION-DOES-NOT-EXIST]))
(DELEMENT
[LAMBDA (X Y)
  (COND
    ((EQ X (CAR Y))
      (CDR Y))
    (T (CONS (CAR Y)
             (DELEMENT X (CDR Y]))
(ERSA
[NLAMBDA (X)
  (ERSA-AUX X]))
(ERSA-AUX
[LAMBDA (X)
  (COND
    ((NULL X)
      (TERPRI)
      "ERASED-ALL-PREDICATES-SPECIFIED)
    (T (ERS-AUX (CAR X))
       (ERSA-AUX (CDR X]))
(DEM
[NLAMBDA (PRED M)
  (DEM1 PRED M]))
(DEM1
[LAMBDA (PRED M)
  (SETQ PROPERTY (GET PRED "DICT))
  (COND
    ((NOT (LISTP PROPERTY))
      "IMPROPER-PROPERTY)
    ((OR (NOT (GREATERP M 0))
         (GREATERP M (LENGTH PROPERTY)))
      "IMPROPER-NUMBER-SPECIFIED)
    (T (PUT PRED "DICT (CDR (DEM-AUX (CONS NIL NIL)
                                     PROPERTY M]))
(DEM-AUX
[LAMBDA (RESVR ORILIST CT)
  (COND
    ((ZEROP (SUB1 CT))
      (NCONC RESVR (CDR ORILIST)))
    (T (DEM-AUX (NCONC RESVR (LIST (CAR ORILIST)))
                (CDR ORILIST)
                (SUB1 CT]))
(DEMR
[NLAMBDA (PRED M)
  [SETQ PROPERTY (REVERSE (GET PRED "DICT]
  (COND
    ((OR (NOT (GREATERP M 0))
         (GREATERP M (LENGTH PROPERTY)))
      "UNACCEPTABLE-NUMBER)
    (T (PUT PRED "DICT (REVERSE (CDR (DEM-AUX (CONS NIL NIL)
                                              PROPERTY M]))
```

```
(PRINZ
[LAMBDA (X)
  (TERPRI)
  (PRIN1 X])
(OP3FNS
NIL)
]
CR((DICT PROPERTY))
DEFLIST((
(P
(((P *X)
  (CALL PRINZ *X)
  /)))
(R
(((R *X)
  (EVAL (READ)
        *X)
  /)))
) DICT)
CR((VALIDPRED PROPERTY))
DEFLIST((
(PREDLIST
(R P))
) VALIDPRED)
CR((VARP PROPERTY))
DEFLIST((
(*X
T)
) VARP)
CR((VALIDVAR PROPERTY))
DEFLIST((
) VALIDVAR)
```

6. 応 用 例

応用例として英語から日本語への簡単な翻訳を述べる．ただし，この翻訳では意味解析部をはぶいている．入力した英文は構文解析されて英語の中間表現に作り直される．つぎにそのまま日本語の中間表現に変換され，ここから日本文が生成される．例の英文は，UNIVAC シリーズ 1100/60 システムの概説書から採った．

英文；

THE DISKETTE DRIVE IS A STORAGE DEVICE USING A INTERCHANGEABLE FLEXIBLE DISKETTE STORAGE MEDIA

英語の中間表現；

```
 (ESENTENCE (ENP (EDET THE) (ENE (EN DISKETTE) (ENE DRIVE))) (EVP (EVEL1 IS) (EN
P (EDET A) (ENE (EN STORAGE) (ENE DEVICE)) (EPPP (
 EPPA USING) (ENP (EDET A) (ENE (EADJ INTERCHANGEABLE) (ENE (EADJ FLEXIBLE) (ENE
 (EN DISKETTE) (ENE (EN STORAGE) (ENE MEDIA)))))))))
```

日本語の中間表現；

```
 (JSENTENCE (JNP (JDET コノ) (JNE (JN ﾃﾞｲｽｹｯﾄ) (JNE ｸﾄﾞｳｿｳﾁ))) (JSC ハ) (JVP (JNP (
JRTS (JNP (JDET ヒトツノ) (JNE (JADJ ｺﾞｶﾝｾｲﾉｱﾙ) (JNE (
 JADJ ﾌﾚｷｼﾌﾞﾙ) (JNE (JN ﾃﾞｲｽｹｯﾄ) (JNE (JN ｷｵｸ) (JNE ﾍﾞｲﾀｲ))))))) (JOC ヲ) (JRT ﾓﾁｲ
ﾙ)) (JDET ヒトツノ) (JNE (JN ｷｵｸ) (JNE ｿｳﾁ))) (JVE ﾃﾞｱﾙ))
 )
```

日本文

```
 (コノ ﾃﾞｲｽｹｯﾄ ｸﾄﾞｳｿｳﾁ ハ ヒトツノ ｺﾞｶﾝｾｲﾉｱﾙ ﾌﾚｷｼﾌﾞﾙ ﾃﾞｲｽｹｯﾄ ｷｵｸ ﾍﾞｲﾀｲ ヲ ﾓﾁｲﾙ ヒトツノ ｷｵｸ
ｿｳﾁ ﾃﾞｱﾙ)
```

プログラムの一部を次に示す．

7. お わ り に

UNIVAC シリーズ 1100 システム上に LISP を用いて製作した 4 種の PROLOG を紹

介した．これらは早稲田大学での講義のために山田真市氏がパーソナル・コンピュータ用に作った PROLOG をもとにしている．どの PROLOG がよいかは，個人的な好みや使用目的等により異るので一概に決められない．ガーベッジ・コレクション，入出力機能，記憶領域の自動拡張，編集機能等は LISP のものを利用するので PROLOG は推論機能を中心とした小振りなものとなっている．ただし，LISP の Structured Editor 以外に本文では言及しなかったが PROLOG の述語を単位とする Editor も試みた．今後は使用経験を積み，よりよい PROLOG 環境を構築したい．

最後に原作者であり今回の作業のきっかけを作っていただいた山田真市氏および，たびたびの技術上の相談に快く応じていただいた高橋肇氏に感謝する．

参考文献

[1] W. F. Clocksin, C. S. Mellish, *Programming in Prolog*, Springer-Verlag 1981.
[2] M. Bruynooghe, *An Interpreter for preoicate logic programs*, Report CW 10, October 1976, Applied Mathematics and Programming Division, Katholieke Universiteit Leuven, Belgium.
[3] M. Bruynooghe, "The Memory Management of Prolog Implementations", *Logic Programming*, Academic Press. 1982, pp. 83-98.
[4] R. S. Boyer, J. S. Moore, "*The Sharing of Structure in Theorem-proving Programs*", *Machine Intelligence 7*, Edinburgh University Press, 1972, pp. 101-116.
[5] *LISP Programmer Reference Manual*, Sperry Co., 1982.
[6] J. McCarthy, P. W. Abrahams, D. J. Edwards, T. P. Hart, M. I. Levin, *LISP 1.5. Programmer's Manual*, The M. I. T. Press, 2nd Edition 1981.
[7] 山田真市，高橋　肇，桑野龍夫，*PROLOG on Series 1100*, tech memo number TMY-00115.
[8] 梅村恭司，"Small Prolog インタプリタ移植のすゝめ"，*bit*, Vol. 15, No. 6, 共立出版，pp. 581-588.
[9] 桑野龍夫，*PROLOG on Series 1100 with Iterative Inference*, tech memo number TMY-00116.
[10] M. Carlsson, "On Implementing Prolog in Functional Programming", *New Generation Computing*, Vol. 2, No. 4, 1984, pp. 347-359.

執筆者紹介

桑野　龍夫 (Tatsuo Kuwano)
昭和 13 年生，38 年東京大学工学部卒．日本ユニバック(株)入社，コミュニケーション・ハードウェア担当を経て，テクニカル・パブリケーション室で機械翻訳の調査・研究に従事．

報告 セミカスタム・セル・ライブラリを使った 32ビットCMOSマイクロプロセッサ

A 32 Bit CMOS Microprocessor Using a Semicustom Cell Library

K. LeClair, R. Bell, D. Breid
P. Torgerson, D. Fier, B. Jensen

要 約 マイクロプログラマブル32ビット CMOS マイクロプロセッサ, PROTEUS のアーキテクチャ, マイクロプログラマブル・フィーチャ, エミュレーション機能等について述べる. アーキテクチャは次の6個の部分からなり, 各部は32ビット内部バスを介して通信する.

1)演論理ユニット (ALU)
2)レジスタ・ファイル (RF)
3)アドレス操作とストレージ (AMS)
4)データ・パス・スイッチ (DPS)
5)マイクロプログラム・コントロール (MPC)
6)外部インタフェース (EXT)

チップでは2ステップのマイクロプログラム・パイプラインを用い, 命令の先読みを行って処理を高速化している. また, 命令のサイクル・タイムは先端 CMOS 技術を用いることによって等価な TTL による設計と比較して速い.

Abstract A microprogrammable 32 bit CMOS microprocessor will be described including the processor architecture, microprogrammable features, and emulation capability. The architecure supports the following six major sections which communicate with each other via an internal 32 bit bus structure:

1) Arithmetic Logic Unit
2) Register File
3) Address Manipulation and Storage
4) Data Path Switch
5) Micro Progam Control
6) External Interface

The chip features a two step microprogram pipeline which greatly enhances execution time by prefetching the next instruction while executing the present cycle. Instruction cycle times using an advanced CMOS technology are faster than an equivalent TTL design.

1. はじめに

本稿では, CMOS セミカスタム技術を使ったマイクロプログラマブル32ビット・マイクロプロセッサについて, そのチップのアーキテクチャ, 設計方法, およびソフトウェア開発等の実験システムについて述べている.

PROTEUS は, 当初 CMOS セミカスタム・ライブラリの能力を論証するために設計されたが, 他のマイクロプロセッサをエミュレーションする機能もあり, 潜在的な応用性をもつマイクロプロセッサである. チップは, 2段階のマイクロ・データ・パイプラインを採用しており, 現在の命令サイクルを実行中に次の命令を読み出すことによって実行時間を大幅に縮少している. マクロ・コードとマイクロ・コードはチップの外側に格納される. 1クロック・サイクルで一つのマイクロ命令を実行することができる.

2. アーキテクチャ

チップのアーキテクチャは図1に示すように大きく次の六つの部分，すなわち演算論理ユニット (ALU : Arithmetic Logic Unit)，レジスタ・ファイル (RF : Register File)．アドレス操作とストレージ (AMS : Address Manipulation and Storage)，データ・パス・スイッチ (DPS : Data Path Switch)，マイクロ・プログラム・コントロール (MPC : Micro Program Control)，および外部インタフェース (EXT : External Interface) の各部で構成されている．

ALU 部はビット，バイトまたは語のローテーションを行う32ビットの演算機能，さらに一時的なデータ保管のために使われる入出力レジスタ，マクロ・ステータス・フラッグの一時的な保管を行う特別なレジスタを用意している．80ビットのマイクロ・ワードのうち，18ビットが ALU の機能を制御する．

AMS 部は，三つのマクロ・アドレス・レジスタ（プログラム・カウンタ，スタック・カウンタ，オペランド・アドレス）におけるアドレスの増減機能をもつ．さらに，24ビットのレジスタが，外部マクロ RAM を最高16メガ語までアドレスするために用意されている．AMS 部はマイクロ・ワードの8ビットを使用する．

RF 部には，マイクロまたはマクロ・アドレスによってアクセスが可能な64×32ビットの RAM が内蔵されている．さらにプログラムで制御可能な二つのアップ/ダウン・カウンタが FIFO，あるいはスタック・オペレーションのために用意されている．RF 部のユニークな機能は，1クロック・サイクルの間に RAM を読み書きできることである．RF 部の制御用に22ビットが使われる．

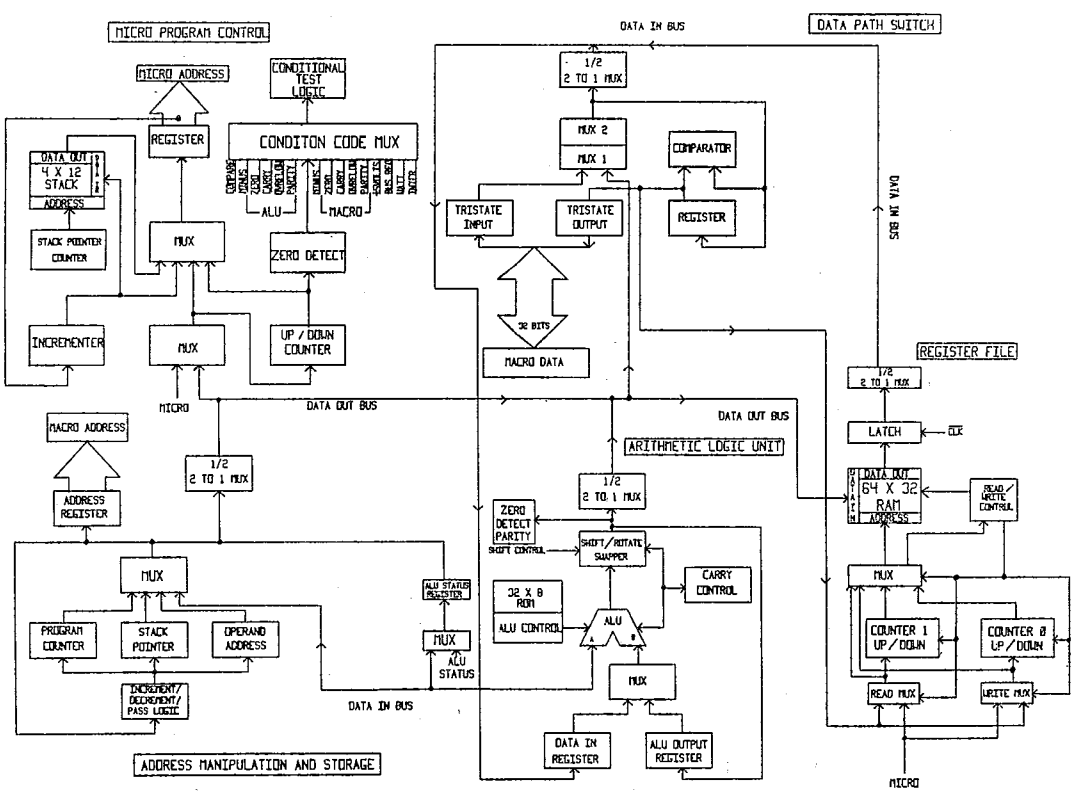


図 1 PROTEUS の構成

Fig. 1 PROTEUS architecture

DPS 部は，32 ビットの比較器の機能とバイトまたはワード・サインの拡張機能がある．DPS 部は 80 ビットのマイクロ語の 4 ビットを使う．

MPC 部は通常のマイクロ命令を順序正しく実行処理する機能の他に，4 レベルのサブルーチン・スタック機能をもつ．16 ビットの条件コード・マルチプレクサが条件付きマイクロ・プログラム・フローの制御に，12 ビットのアップ/ダウン・カウンタがシーケンス繰返し制御に使用される．MPC 部ではマイクロワードの 20 ビットを使用する．

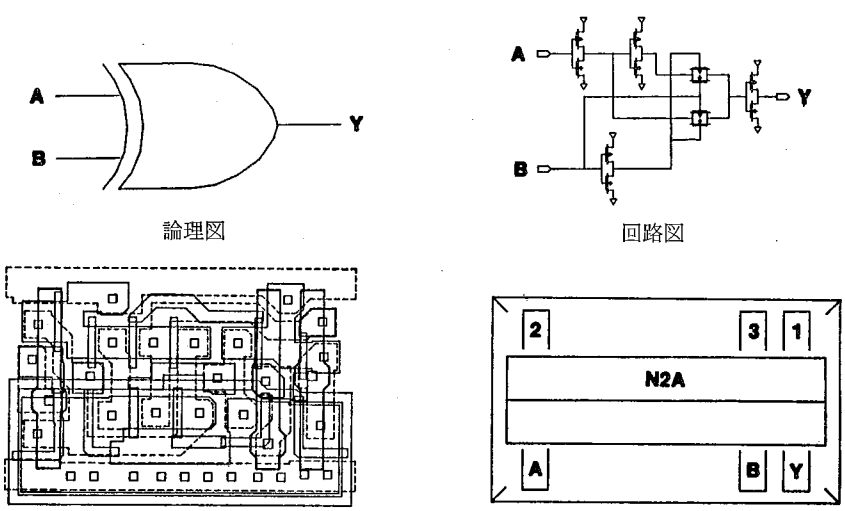
EXT 部はマクロ I/O を制御する八つのポートで構成される．EXP 部では 80 ビットのマイクロ語の 8 ビットに直接これらのマクロ I/O を制御する．

3. 設 計 方 法

PROTEUS の開発の第一目標は，最小の開発コストで必要な機能を実現することであった．この目標の遂行を容易にするために，1.2 μm の二層メタルの CMOS 技術[2]によるセミカスタム設計方法[1]を採用した．このアプローチでは，ゲートの高密度化，高性能化および効率的なシリコン利用を可能とするカスタム設計の論理セルおよびメモリ・セルのライブラリを使用している．

セミカスタム・ライブラリはおよそ 60 個の論理セルとメモリ・セルからなり，基本的な論理ゲートからスタティックな RAM および 16 ビットの ALU に至るまでを構成することができる．SSI セルおよび MSI セルの多くは 7400 シリーズの TTL のコンポーネントと類似したものである．

ライブラリ中のすべてのセルは，レベル・ピッチを相互に接続するのに最適化された格子システムに従って物理的に配列できるようにした．SSI セルと MSI セルに対しては，水平にいくつもくしの歯のように結合することを考慮して，その大きさを統一した．この“ブリッスル・ブロック”方式はパイプラインのアーキテクチャのレイアウトに大きく貢献している[3]．SSI セルあるいは MSI セルには，さらに経路や位置に柔軟性をもたせるために，その内部に第二のメタルを使用しなかった．このため，ルート・チャネルは，その位置に入出力部分がない場合は，セルを直接通過する可能性があるので，経路決定が容



論理図　回路図

レイアウト　スティック・オン

図 2　セルの開発

Fig. 2 Cell development

易になる．この制約がない LSI セル，たとえば，RAM, ROM または ALU については，垂直経路決定に抵触することが最も少なくなる場所に配置した．

図2は，典型的なセルの開発サイクルの例である．PROTEUSのレイアウトではファイナル・スティック・オン・セルがセルの配置やプロッティングに必要なコンピュータ負荷を削減するために使われた．“ブリッスル・ブロック”セルのスティック・オンについて図3に示す．ブロック化されたチャネルおよび信号線はオープンのままである．

図4に PROTEUS の顕微鏡写真を示す．トランジスタ数は約 42,000 個である．チップ寸法は 7.64 mm×7.78 mm で，I/O ピン数は電源とアース用の 16 ピンを含む 177 ピンである．チップは TTL と互換性があり +5V の電圧で動作するものの，I/O ピン数が 151 ピンに限定されており，したがってこの例ではチップの約 60 パーセントのみが使用され，残りは使われない．

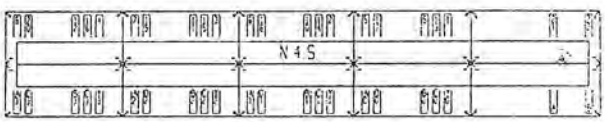


図 3 ブリッスル・ブロック・ステック・オン

Fig. 3 Bristle block stick on

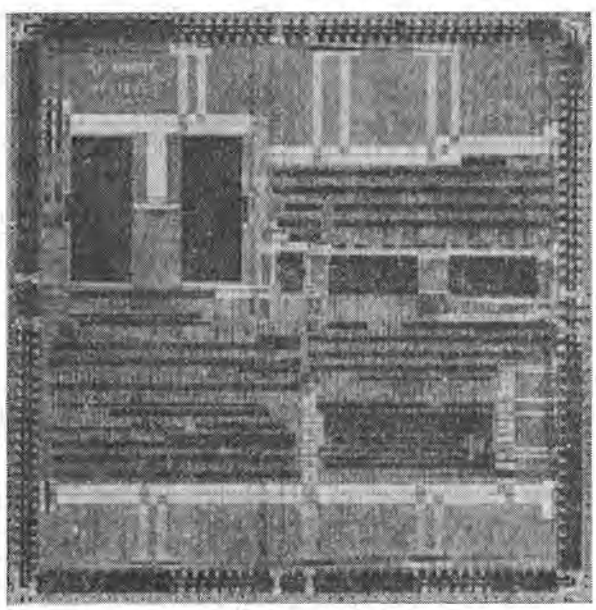


図 4 PROTEUS

Fig. 4 PROTEUS

4. CAD ツール

PROTEUSの設計では，速いターン・アラウンド・タイムと大きな成果を実現するためさまざまな CAD ツールが使われた．チップの主要な6部分の各部を完全にシミュレートするのに SIMLOGIC というゲート・レベルの論理分析プログラムを使用した．その理由は，マイクロ・コードとマクロ・コードの開発にリアル・タイムの環境下でチップ全体をシミュレートすることができるためである．INDICATES という障害生成および分析プログラムによりテスト・パターンを作成し，レイアウト・ロジック (LTL) ソフトウェ

```
MNEMONIC>>>EXG XN1, XN2<<<XN1<=>XN2
[UUUUUUUUUUUUUOOOOOOOORRRROOOORRRR]
ALU,I,B          PASS,A          ・MOVE XN2 TO INREG
RF,L,B           WM,RM,WM
ENDU
ALU,,B           PASS,A          ・MOVE XN1 TO XN2
RF,W.B           RM,RM,WM
MPC              LMP,,MACOP
ENDU
ALU,,B           PASS,I          ・MOVE INREG TO XN1
RF,W,B           C1
ENDU
```

図 5 EXG 命令

Fig. 5 An EXG instruction

表 1 PROTEUS 命令セット

Table 1 PROTEUS instruction set

ADD	(6)	BLE	(11)	EOR	(6)	NOP	(4)
ADDI	(6)	BLT	(9)	EORI	(6)	NOT	(6)
AND	(6)	BMI	(7)	EXG	(7)	OR	(6)
ANDI	(6)	BNG	(7)	JMP	(6)	ORI	(6)
ASL	(14+3n)	BPL	(7)	JSR	(8)	ROL	(14+3n)
ASR	(15+3n)	BVC	(7)	LSL	(14+3n)	ROR	(14+3n)
BBUS	(7)	BVS	(7)	LSR	(14+3n)	RTI	(6)
BCC	(7)	BWAIT	(7)	MOVE	(6)	RTS	(6)
BCS	(7)	CLI	(6)	MOVI	(6)	SEI	(6)
BEQ	(7)	CLR	(6)	MULS	(118)	SUB	(6)
BGE	(9)	CMP	(6)	MULU	(88)	SUBI	(6)
BGT	(11)	CMPI	(6)	NEG	(8)	SWAP	(6)

アによりチップのレイアウトの正確性を確認した．さらに MASM と呼ばれるメタ・アセンブラを使ってモトローラ M68000 に似た命令を実行できるマイクロ・コードとマクロ・コードを開発した．

5. ソフトウェア開発

80 ビットのマイクロ・ワードと，32 ビットのマクロ・ワードの組合せは，事実上，無制限のオペレーションを可能とする．

一般には，マクロ・ワードのビット 21〜31 が，必要な命令を実行するマイクロを指し示すベクトルとして用いられる．

図 5 に，一つのマクロ命令である EXG (Exchange Register) 命令と，その EXGに対応するマクロ・コードを表示した．両方とも，マクロ・アセンブラとマイクロ・アセンブラで使用されるアセンブラ・ニーモニックスで表現している．

表 1 には，現在使われている命令一式と各命令に必要なサイクル数を表示してある．各サイクルには，インタラプトをチェックし，その後マイクロ・ベクトルをアクセスしてロードするために必要な 4 サイクル・オーバヘッドがある．しかし，ユーザがインタラプトをチェックする必要がなければ，命令は 2 サイクル分減らすことができ，サイクル・タイム平均 40 パーセントを削減できる．

6. 性　　能

チップの設計目標として，最適サイクル・タイムが 120 ナノ秒で，最悪の場合でも 200

ナノ秒を設定した．しかし，チップにはさまざまな命令形式が可能であり，無数の多様性と柔軟性があるので，最終的なサイクル時間は，マイクロ・コードの固有の組合せやマイクロ・コードの記憶に使われる外部の RAM または ROM のアクセス時間によって決まる．たとえば，DPS 部を通してデータを読み込み，ALU を通して，RF が RAM に書き込むのにかかる時間は，単に DPS 部を通してデータを読み込み，ALU 部の「データ・イン・レジスタ」に記憶させるよりずっと長くかかる．開発されたマクロ・セットから種々のマクロ・ワードを SIMLOGIC でシミュレーションした結果，サイクル・タイムは 100 ナノ秒に押えることが可能なことがわかった．

CMOS 技術により，静的電力を消費するのは，ほとんど RAM と ROM だけとなっている．チップの待機時の電力は 500 ミリワットを消費する．演算に要する電力消費は，当然のことだが，使用されたサイクル・タイムによって決まる．

7. おわりに

PROTEUS の開発により，セミカスタム設計方式が低い開発コストでかつ大幅開発時間を短縮することが証明された．最初のアーキテクチャ設計後，約 12 か月で第一段階のシリコンが開発されたが，その開発時間は 4 人年以下である．さらに，その開発に伴って，ほとんどすべての CAD ツールをテストし，その機能を強化し拡張した．なお，これら実験システムは現在のところ商品化の計画はない．

最後に J. Vesely, W. Grant, M. Wolfe, S. Paulson の各氏をはじめ MOS オペレーション部の開発スタッフ全員に感謝を述べたい．

（ハードウェア・プロダクト統括部　滝沢　繁　訳）

参考文献 [1] D.F. Fier, W.W. Heikkila, "High Performance CMOS Design Methodologies", *Proceeding CICC*, 1982, pp. 325-328.

[2] S.A. Campbell, D. Dokos, W.N. Grant, J.P. Victorey, "A 1.2 micro double-level-metal CMOS Technology", *Proceeding CICC*, 1983, pp. 61-64.

[3] D.L. Johannsen, "Silicon Compilations", Caltech Technical Report 4530, Department of Computer Science, Caltech 1981.

執筆者紹介 Kevin R. LeClair

1982年，Minnesota 工科大学において電気工学で B.S. を取得．以来，Sperry 社のセミコンダクタ・ディビジョンに在職．CMOS ゲート・アレイやセミカスタム IC のロジック設計と回路の設計に従事．現在，標準的なセル・プロダクトの設計のため，一連のセミ・スティック RAM の回路設計に従事．さらに，Minnesota 工科大学において電気工学の M.S. を修得中．

Ron Bell

17年以上にわたり，コンピュータ企業に在職し，とくに，マイクロプロセッサの設計と実施に従事．現在，個人的にシリコンのコンパイレーションを行っている．マイクロ・プロダクト・ディビジョンのリサーチ・アンド・アドバンスド・テクノロジ部の部長であり，また，同ディビジョンの G.V.P の技術コンサルタントを兼任．Utah 大学において電気工学の B.S. と M.S. を取得．

Duane G. Breid
1974年 Sperry 社のセミコンダクタ MOS オペレーション部に入社. NMOSI セミカスタム・ライブラリの開発と特性表示に従事し, NMOSI プロジェクト JAGUAR においてシリコンのデバッギングを指揮. PROTEUS プロジェクトにおいて, ロジック設計, シミュレーション, テスト・ジェネレーションを行い, マイクロ・アセンブラを開発. 現在, MICRO-1100 プロジェクトの SILICON 設計者 (上級電気技術者).

Paul D. Torgerson
1981年 Minnesota 州立大学より電気工学で B. S. を取得. 1979年から同大学卒業まで, Sperry 社の国防システム部門とセミコンダクタ・ディビジョンに在職. 現在, プロダクト・ディビジョンのマイクロ・プロダクト開発部のシリコン設計技術者 (上級電気技術者).

Donald F. Fier
1974年と1980年にMinnesota州立大学において電気工学でB. S. および M. S. を取得. 1974年から1975年の間, MOSTEK 社に在職し, 主として非揮発性 MOS メモリの設計を担当するメモリ設計技術者であった. その後, Sperry 社に入社, 現在まで9年間にわたり, NMOS と CMOS のカスタムとセミカスタムの IC 設計に従事. CMOS セミカスタム・ライブラリを使う PROTEUS を監督するとともに, NMOS および CMOS セミカスタム・セル・ライブラリの設計を監督. 現在, マイクロ・プロダクト開発部のシリコン設計担当マネージャ.

Bill Jensen
1967年 Sperry 社に入社. 1970年以降, セミコンダクタ・オペレーション部で, レイアウトと CAD の分野を担当. ほとんどの半導体プロジェクトに加わり, I/O やライブラリのセルの開発, メモリ開発技術テスト・チップ, ゲート・アレイ, セミカスタム・チップおよびフル・カスタム・チップの開発に従事. プログラムにおいては, サブセルの開発およびフロア・プランニングと最終チップの設計に従事.

論文 非直交曲線格子系による座標変換差分法(その1)

Coordinate Transformation Finite Difference Method Refering to Non Orthogonal Mesh System (Part 1)

藤野 勉

要 約 本稿は非直交曲線格子系による座標変換差分法の研究に関するものである．直交曲線格子系による座標変換差分法では，その計算精度は良好であるが，任意形状の境界をもつ領域内にこれを設定することは非常に困難である．一方，非直交曲線格子系を設けることは複雑な形状の領域についても容易であり，実用的な工学的問題の解析にも利用することができる．しかしその計算精度は前者に比べ多少劣るものと思われる．

Abstract This paper is concerned with a study of coordinate transformation finite difference method refering to non orthogonal mesh system. Generally, it is very difficult to establish orthogonal curvilinear mesh in a domain with arbitrally shaped boundary but calculation accuracy is preferable. On the other hand, set up of non orthogonal mesh is easy whereas accuracy of calculation is thought to be not so good as former. Therefore we can apply this method for the analysis of practical engineering problems.

1. はじめに

差分法はこれを大別して，(a)直交直線格子系，(b)任意節点配置，(c)座標変換とすることができる．(a)はもっとも古くから用いられた差分法で，計算手続きは簡単で，内部精度（領域内における差分精度）が良好である．しかるに，外部精度（境界適合性）が不良で実用的な工学問題の処理に不適当である．これを改善するため有限要素法が開発されたが，任意節点配置差分法も同様に有効である．しかし，この方法では節点ごとに逆行列を算出することが必要となり，計算量の大型化が懸念される．(c)の座標変換差分法は x-y（物理平面座標）のほかに ξ-η（解析平面座標）を導入し，この平面において物理平面における任意形状の領域を単純な矩形領域に写像し，(a)の差分法が適用されるように変形を行うものである．このようにして内，外精度ともに改善される．写像の方法としては直交曲線座標系を用いることが理想であるが，一般にはほとんど不可能である．ここで精度上多少難があるが，非直交曲線座標系を導入し，解析平面内に格子定数1の直交直線格子系を設定するもので，本稿では主としてその導入法について述べた．

2. 曲線座標系とその微分幾何

2次元 x-y 平面内における曲線座標

$$\xi=\xi(x,y),\quad \eta=\eta(x,y) \tag{2-1}$$

が陽に定義され，下記微係数

$$\left.\begin{aligned}&\xi_x=\partial\xi/\partial x,\ \xi_y=\partial\xi/\partial y,\ \xi_{xx}=\partial^2\xi/\partial x^2,\ \xi_{xy}=\partial^2\xi/\partial x\partial y,\ \xi_{yy}=\partial^2\xi/\partial y^2,\ \cdots\cdots\\&\eta_x=\partial\eta/\partial x,\ \eta_y=\partial\eta/\partial y,\ \eta_{xx}=\partial^2\eta/\partial x^2,\ \eta_{xy}=\partial^2\eta/\partial x\partial y,\ \eta_{yy}=\partial^2\eta/\partial y^2,\ \cdots\cdots\end{aligned}\right\} \tag{2-2}$$

は逐次定められるものとする．ξ,η は一般に非直交である．

2.1 位置ベクトルならびにその微分

図1に示されるように，x,y 方向の単位ベクトルを i_0, j_0 として

$$\boldsymbol{r}=i_0x+j_0y \tag{2-3}$$

によって位置ベクトルが定義され，その微分

$$d\boldsymbol{r}=i_0dx+j_0dy=i_0(x_\xi d\xi+x_\eta d\eta)+j_0(y_\xi d\xi+y_\eta d\eta)=ids+jdt \tag{2-4}$$

によって，ξ,η 方向の測度係数 g_1,g_2，単位ベクトル i,j，微小長 ds,dt は次のように定められる．

$$\left.\begin{aligned} g_1&=\sqrt{x_\xi^2+y_\xi^2},\quad i=(i_0x_\xi+j_0y_\xi)/g_1,\quad ds=g_1d\xi\\ g_2&=\sqrt{x_\eta^2+y_\eta^2},\quad j=(i_0x_\eta+j_0y_\eta)/g_2,\quad dt=g_2d\eta \end{aligned}\right\} \tag{2-5}$$

つぎに図2に示されるように，i,j が i_0 とはさむ角を α,β とするとき，式(2-5)の単位ベクトル i,j の定義と見比べて

$$\cos\alpha=x_\xi/g_1,\ \sin\alpha=y_\xi/g_1,\ \cos\beta=x_\eta/g_2,\ \sin\beta=y_\eta/g_2 \tag{2-6}$$

の関係が導かれる．もし ξ,η が直交ならば

$$i\cdot j=(x_\xi x_\eta+y_\xi y_\eta)/g_1g_2=0 \tag{2-7}$$

が満たされ，さらに等測度係数系ならば $g_1=g_2$ によって

$$x_\xi^2+y_\xi^2=x_\eta^2+y_\eta^2 \tag{2-8}$$

が成立し，これらの条件は

$$x_\eta=-y_\xi,\quad y_\eta=x_\xi \tag{2-9}$$

と置くことによって満たされる．このような曲線座標系を等測度曲線座標系，または正規直交曲線座標系と呼ぶこととする．このとき式(2-6)によって，当然 $\beta=\alpha+\pi/2$ となる．

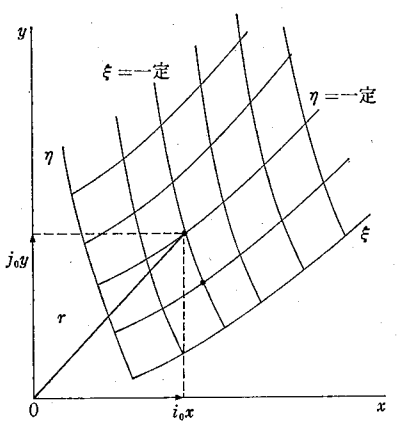


図 1　曲線座標系とベクトル

Fig. 1　Curvilinear coordinate and position vector

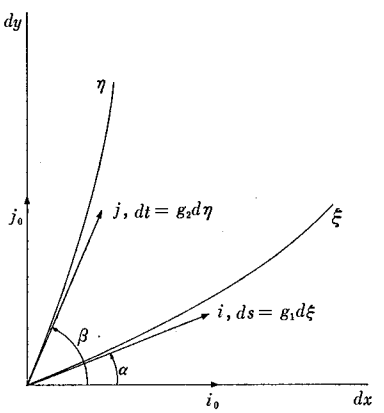


図 2　単位ベクトルと微小長

Fig. 2　Unit vector and infinitesimal length unit

以上によって x,y が ξ,η の関数形として表されたときの諸式が導かれたが，実際は式(2-1)によって ξ,η が x,y の関数形として表現されているので，それに適する式を求めることが必要である．その準備として，次の変形を行う．

$$\left.\begin{aligned} d\xi&=\xi_xdx+\xi_ydy=\xi_x(x_\xi d\xi+x_\eta d\eta)+\xi_y(y_\xi d\xi+y_\eta d\eta)\\ &=(\xi_xx_\xi+\xi_yy_\xi)d\xi+(\xi_xx_\eta+\xi_yy_\eta)d\eta\\ d\eta&=\eta_xdx+\eta_ydy=\eta_x(x_\xi d\xi+x_\eta d\eta)+\eta_y(y_\xi d\xi+y_\eta d\eta)\\ &=(\eta_xx_\xi+\eta_yy_\xi)d\xi+(\eta_xx_\eta+\eta_yy_\eta)d\eta\\ \therefore\quad &\xi_xx_\xi+\xi_yy_\xi=1,\ \xi_xx_\eta+\xi_yy_\eta=0,\ \eta_xx_\xi+\eta_yy_\xi=0,\ \eta_xx_\eta+\eta_yy_\eta=1 \end{aligned}\right\} \tag{2-10}$$

上式を $x_\xi,y_\xi,x_\eta,y_\eta$ について解き

$$x_\xi=\eta_y/J,\ y_\xi=-\eta_x/J,\ x_\eta=-\xi_y/J,\ y_\eta=\xi_x/J \tag{2-11}$$

を得る．ただし，J は次のようである．

$$J=\xi_x\eta_y-\xi_y\eta_x \tag{2-12}$$

である．したがって，正規直交曲線座標系では，

1次：
$$\eta_x=-\xi_y,\ \eta_y=\xi_x \tag{2-13}$$

の関係が導かれる．さらに高次の微係数については，

2次：
$$\eta_{xx}=-\xi_{xy},\ \eta_{xy}=-\xi_{yy}=\xi_{xx},\ \eta_{yy}=\xi_{xy}$$
$$\therefore\ \ \xi_{xx}+\xi_{yy}=0,\ \eta_{xx}+\eta_{yy}=0 \tag{2-14}$$

3次：
$$\eta_{xxx}=-\xi_{xxy},\ \eta_{xxy}=-\xi_{xyy}=\xi_{xxx},$$
$$\eta_{xyy}=\xi_{xxy}=-\xi_{yyy},\ \eta_{yyy}=\xi_{xyy}=-\xi_{xxx} \tag{2-15}$$

4次：
$$\eta_{xxxx}=-\xi_{xxxy},\ \eta_{xxxy}=-\xi_{xxyy}=\xi_{xxxx},\ \eta_{xxyy}=-\xi_{xyyy}=\xi_{xxxy},$$
$$\eta_{xyyy}=\xi_{xxyy}=-\xi_{yyyy},\ \eta_{yyyy}=\xi_{xyyy}=-\xi_{xxxy} \tag{2-16}$$

となり，さらにこれを整理して

1次：
$$\xi_x,\ \xi_y,\ \eta_x=-\xi_y,\ \eta_y=\xi_x \tag{2-17}$$

2次：
$$\xi_{xx},\ \xi_{xy},\ \xi_{yy}=-\xi_{xx},\ \eta_{xx}=-\xi_{xy},\ \eta_{xy}=\xi_{xx},\ \eta_{yy}=\xi_{xy}, \tag{2-18}$$

3次：
$$\xi_{xxx},\ \xi_{xxy},\ \xi_{xyy}=-\xi_{xxx},\ \xi_{yyy}=\xi_{xxy},\ \eta_{xxx}=-\xi_{xxy},$$
$$\eta_{xxy}=\xi_{xxx},\ \eta_{xyy}=\xi_{xxy},\ \eta_{yyy}=-\xi_{xxx} \tag{2-19}$$

4次：
$$\xi_{xxxx},\ \xi_{xxxy},\ \xi_{xxyy}=-\xi_{xxxx},\ \xi_{xyyy}=-\xi_{xxxy},\ \xi_{yyyy}=\xi_{xxxx},$$
$$\eta_{xxxx}=-\xi_{xxxy},\ \eta_{xxxy}=\xi_{xxxx},\ \eta_{xxyy}=\xi_{xxxy},$$
$$\eta_{xyyy}=-\xi_{xxxx},\ \eta_{yyyy}=-\xi_{xxxy} \tag{2-20}$$

が導かれる．このように各次数において自由な微係数は2個のみである．なお非直交の曲線座標系ではすべての微係数が独立である．

3. 曲線座標系の設定

任意の境界 l によって囲まれた領域 S の内部に曲線座標 ξ,η を設定する．このとき曲線座標は境界に適合していること，すなわち境界が $\xi=$一定 または $\eta=$一定 によって表されていることが必要である．もしこの条件が満たされていなければ曲線座標を採用した意味を失うこととなる．したがって，まったく任意な境界について曲線座標系を定めることは困難で，自らある制約を受けることとなる．ここではつぎに述べる形の境界について考察を行うこととする．

3.1 3 境 界 領 域

もっとも単純な形として三つの境界に囲れた領域を取り上げ，その内部に曲線座標 $\xi-\eta$ を設定する．図3(a)に示されるように，3本の境界線を l_1,l_2,l_3 とし，これらに対応する頂点を $P_1(x_1,y_1)$, $P_2(x_2,y_2)$, $P_3(x_3,y_3)$ とする．いま曲線 l_1,l_2,l_3 は，それぞれ次式で表されるものとする．

$$f_1(x,y)=0,\ f_2(x,y)=0,\ f_3(x,y)=0 \tag{3-1}$$

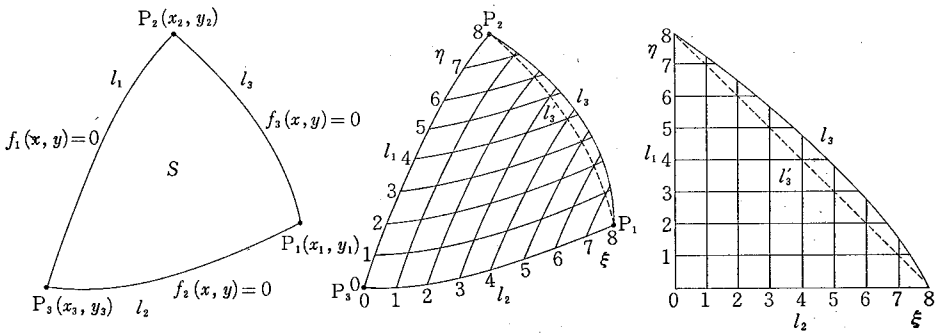
ここで当然，

$$f_1(x_1,y_1)\neq0,\ f_2(x_2,y_2)\neq0,\ f_3(x_3,y_3)\neq0 \tag{3-2}$$

であるものとする．つぎに m,n を正の整数として

$$\xi=mf_1(x,y)/f_1(x_1,y_1),\ \eta=nf_2(x,y)/f_2(x_2,y_2) \tag{3-3}$$

によって曲線座標 ξ,η を定義する．この座標系は一般に非直交である．なお，ここでは便宜上 $m=n$ とする．図3(b)に示されるように $\xi=i$, $\eta=j$ の曲線格子系は図3(c)に



（a） 一般図　　（b） 物理 (x-y) 平面 曲線格子系　　（c） 解析 (ξ-η) 平面 直線格子系

図 3　3境界領域

Fig. 3　A domain with three boundaries

示されるような2等辺直角三角形，格子定数1の直交直線格子系に写される．ただし i, j は m より小さい正の整数である．このとき l_3, と l_3' は必ずしも一致しない．

3.1.1 直線境界

図4（a）に示されるように三つの直線 l_1, l_2, l_3 で囲まれた三角形領域内に直線格子系を設定する．直線 l_1, l_2, l_3 を表す式は

$$\left.\begin{aligned} f_1 &= x_2y_3 - x_3y_2 + (y_2 - y_3)x + (x_3 - x_2)y = 0 \\ f_2 &= x_3y_1 - x_1y_3 + (y_3 - y_1)x + (x_1 - x_3)y = 0 \\ f_3 &= x_1y_2 - x_2y_1 + (y_1 - y_2)x + (x_2 - x_1)y = 0 \end{aligned}\right\} \tag{3-4}$$

によって与えられる．これらの式は有限要素法における面積座標を用いると便利である．

$$\left.\begin{aligned} &\alpha_1 = x_2y_3 - x_3y_2,\ \beta_1 = y_2 - y_3,\ \gamma_1 = x_3 - x_2, \\ &\qquad a_1 = \alpha_1/\alpha,\ b_1 = \beta_1/\alpha,\ c_1 = \gamma_1/\alpha,\ \zeta_1 = a_1 + b_1x + c_1y \\ &\alpha_2 = x_3y_1 - x_1y_3,\ \beta_2 = y_3 - y_1,\ \gamma_2 = x_1 - x_3, \\ &\qquad a_2 = \alpha_2/\alpha,\ b_2 = \beta_2/\alpha,\ c_2 = \gamma_2/\alpha,\ \zeta_2 = a_2 + b_2x + c_2y \\ &\alpha_3 = x_1y_2 - x_2y_1,\ \beta_3 = y_1 - y_2,\ \gamma_3 = x_2 - x_1, \\ &\qquad a_3 = \alpha_3/\alpha,\ b_3 = b_3/\alpha,\ c_3 = \gamma_3/\alpha,\ \zeta_3 = a_3 + b_3x + c_3y \\ &\alpha = \alpha_1 + \alpha_2 + \alpha_3 \end{aligned}\right\} \tag{3-5}$$

ここに $\zeta_1, \zeta_2, \zeta_3$ は面積座標を表す．したがって，

$$f_1 = \alpha_1 + \beta_1x + \gamma_1y = \alpha\zeta_1,\ f_2 = \alpha_2 + \beta_2x + \gamma_2y = \alpha\zeta_2,\ f_3 = \alpha_3 + \beta_3x + \gamma_3y = \alpha\zeta_3 \tag{3-6}$$

とすることができる．つぎに式(3-3)によって，

$$\xi = mf_1(x, y)/f_1(x_1, y_1) = m\zeta_1,\ \eta = mf_2(x, \eta)/f_2(x_2, y_2) = m\zeta_2 \tag{3-7}$$

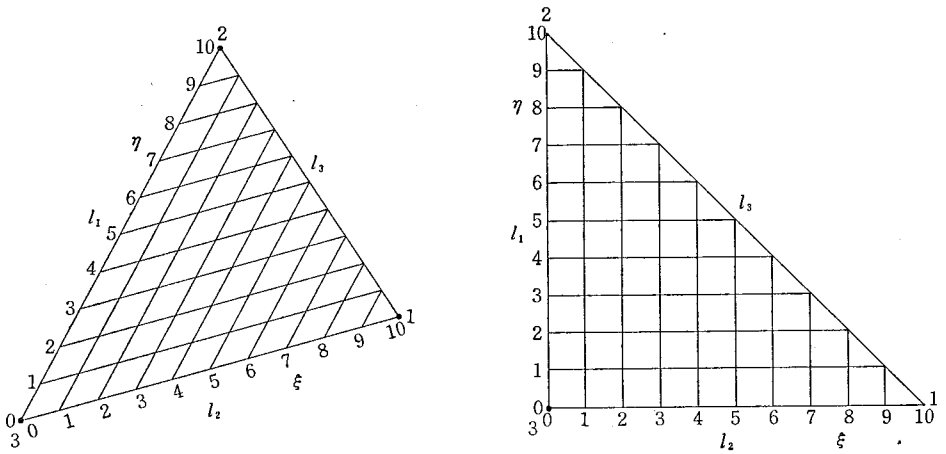
となる．ただし $\zeta_1(x_1, y_1) = 1$, $\zeta_2(x_2, y_2) = 1$ である．境界 l_3 上では $\zeta_3 = 0$ であるから

$$\xi + \eta = m(\zeta_1 + \zeta_2) = m(1 - \zeta_3) = m$$

の関係が導かれる．このように $\xi = i$, $\eta = m - i$ の格子点は l_3 上に存在する．このように $m = n$ を仮定したことによって格子点が l_3 上にのることは，ここで微係数の差分化が容易に行れることを意味している．

3.1.2 曲線境界

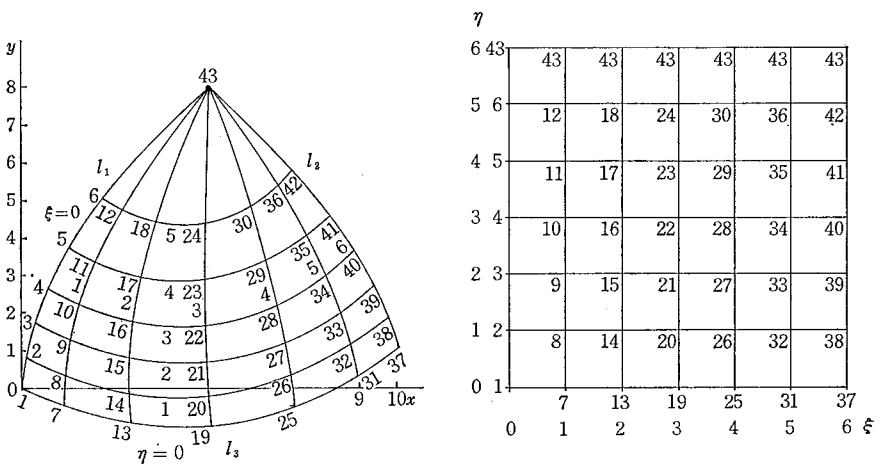
曲線を境界にもつ三境界領域内には，図3に示されるように，一般に l_3 と l_3' が一致するという保証は存在しない．したがって，このような曲線格子系の設定法は不適当で



（a） 物理 (x-y) 平面格子系　　（b） 解析 (ξ-η) 平面格子系

図 4 直線 3 境界領域 ($m=n=10$)

Fig. 4 A domain with linear three boundaries



（a） 物理平面　　（b） 解析平面

図 5 3 境界領域内曲線格子系

Fig. 5 Curvilinear mesh system in a domain with three boundaries

ある．ここではこれに代わるものとして，図 5 に示されるような設定法を提案する．この格子系では 節点 43 は特異点となり，解析平面上で $\eta=6$ 上の節点番号はすべて同一となる．

3.2 4 境界領域

図 6 に示されるように四つの境界 l_1, l_2, l_3, l_4 によって囲まれた領域 S の内部に曲線座標 ξ,η を設定する．これらの境界の形はそれぞれ，

$$\varphi_1(x,y)=0,\ \varphi_2(x,y)=0,\ \varphi_3(x,y)=0,\ \varphi_4(x,y)=0 \tag{3-8}$$

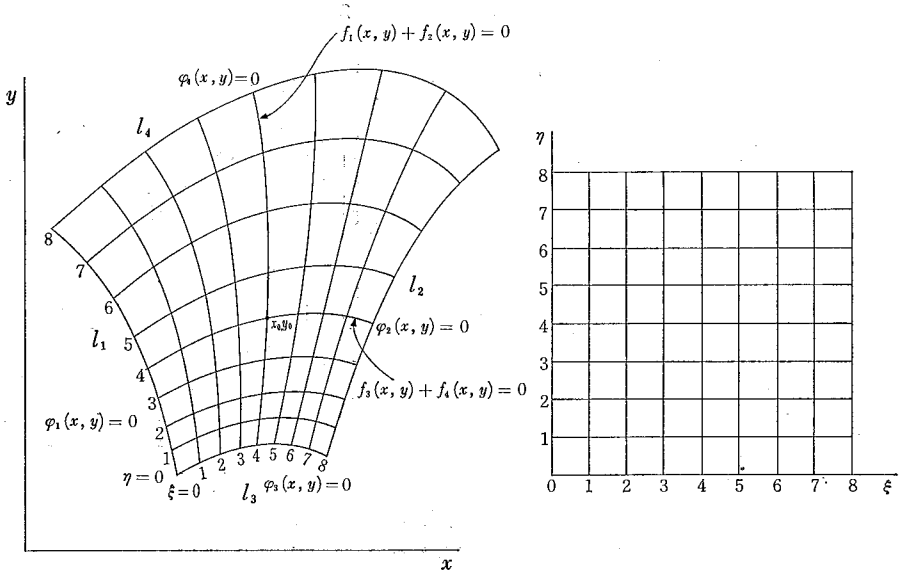
によって表されるものとする．いま領域内に，

$$(m-\xi)\varphi_1-\xi\varphi_2=0,\ (n-\eta)\varphi_3-\eta\varphi_4=0 \tag{3-9}$$

または，

$$\xi=m\varphi_1/(\varphi_1+\varphi_2),\ \eta=n\varphi_3/(\varphi_3+\varphi_4) \tag{3-10}$$

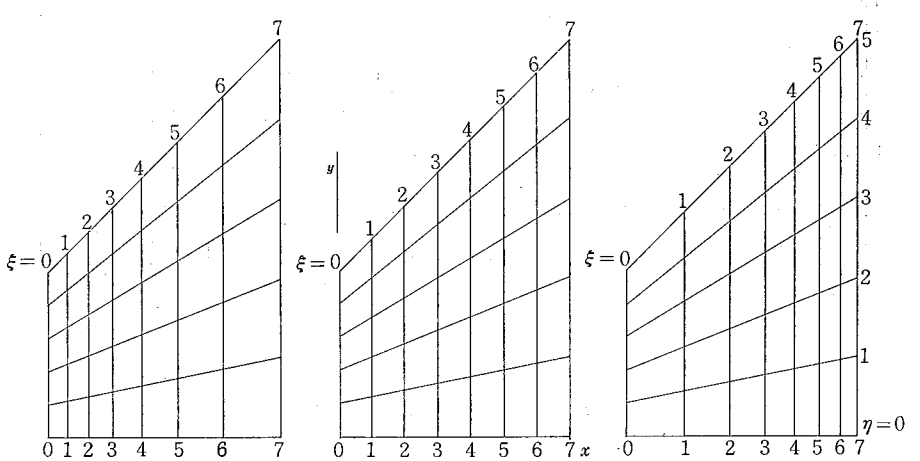
によって曲線座標 ξ,η を定義する．したがって境界 l_1, l_2, l_3, l_4 はそれぞれ $\xi=0, \xi=m$,



（a） 物理 (x-y) 平面格子系　　（b） 解析 (ξ-η) 平面格子系

図 6　4境界領域

Fig. 6　Domain with four boundaries



（a） $w_1=2,\ w_2=1$　　（b） $w_1=1,\ w_2=1$　　（c） $w_1=1,\ w_2=2$

図 7　重み係数による物理平面格子系密度の変化 ($f_1=x,\ f_2=x-7$)

Fig. 7　Weighting factor and it's relation to mesh density in physical plane

$\eta=0$, $\eta=n$ で表されている．ここに m,n はそれぞれ ξ,η 曲線格子の数を表し，正の整数とする．ここで $0<i<m$, $0<j<n$ を正の整数とするとき，$\xi=i$, $\eta=j$ は物理平面 (x-y) 内で曲線格子，解析平面 (ξ-η) 内では格子定数 1 の等間隔直交直線格子系をつくる．いま曲線

$$\varphi_1-\varphi_2=0,\quad \varphi_3-\varphi_4=0$$

が領域の中心を通過するならば物理平面で一様な網目の格子系が得られる．しかし，領域の形によっては不等間隔の曲線格子系が望ましい場合もある．このとき，次のようにして調整を行うことができる．たとえば，ξ に関し，φ_1, φ_2 の重み係数をそれぞれ w_1, w_2 と置き，これらを $f_1=w_1\varphi_1$, $f_2=w_2\varphi_2$ によって置き換える．このようにしても境界 l_1, l_2

における ξ の値に影響を受けることはない．図7に重み係数による物理平面の格子系の変化を簡単な例によって示した．一般に $\xi=m/2$ 曲線が x_0, y_0 を通過する場合には $w_1=1/\varphi_1(x_0, y_0)$, $w_2=1/\varphi_2(x_0, y_0)$ と置けばよい．η 曲線についても同様に $w_3=1/\varphi_3(x_0, y_0)$, $w_4=1/\varphi_4(x_0, y_0)$, $f_3=w_3\varphi_3$, $f_4=w_4\varphi_4$ とする．

式(3-2)または(3-3)によって ξ, η と x, y の関係は1対1の対応を行うように定義されている．ここで座標変換差分を行うためには各格子点の x-y 座標が正確に求められていることが要求される．しかるに $x(\xi, \eta)$, $y(\xi, \eta)$ の形ではないので，直接 x, y 座標を求めることは困難で，何回かの繰返しによって逐次正解に近づけていく．このようにして格子点の座標が正確に定められたのち，その微係数 ξ_x, ξ_y, $\xi_{xx}, \cdots, \eta_x$, $\eta_y, \eta_{xx}, \cdots$ を算出することが必要である．たとえば ξ に関し，式(3-3)によって一般に重み係数も考慮して，

$$f(x, y)=f_1(x, y)+f_2(x, y),\quad g(x, y)=mf_1(x, y) \tag{3-11}$$

とするとき，その x, y に関する微係数は次式によって逐次求められる．

$$\left.\begin{aligned}
&\xi=g/f,\ \xi_x=(g_x-f_x\xi)/f,\ \xi_y=(g_y-f_y\xi)/f,\ \xi_{xx}=\{g_{xx}-(f_{xx}\xi+2f_x\xi_x)\}/f,\\
&\xi_{xy}=\{g_{xy}-(f_{xy}\xi+f_y\xi_x+f_x\xi_y)\}/f,\ \xi_{yy}=\{g_{yy}-(f_{yy}\xi+2f_y\xi_y)\}/f,\\
&\xi_{xxx}=\{g_{xxx}-(f_{xxx}\xi+3f_{xx}\xi_x+3f_x\xi_{xx})\}/f,\\
&\xi_{xxy}=\{g_{xxy}-(f_{xxy}\xi+2f_{xy}\xi_x+f_{xx}\xi_y+f_y\xi_{xx}+2f_x\xi_{xy})\}/f,\\
&\xi_{xyy}=\{g_{xyy}-(f_{xyy}\xi+f_{yy}\xi_x+2f_{xy}\xi_y+2f_y\xi_{xy}+f_x\xi_{yy})\}/f,\\
&\xi_{yyy}=\{g_{yyy}-(f_{yyy}\xi+3f_{yy}\xi_y+3f_y\xi_{yy})\}/f,\\
&\xi_{xxxx}=\{g_{xxxx}-(f_{xxxx}\xi+4f_{xxx}\xi_x+6f_{xx}\xi_{xx}+4f_x\xi_{xxx})\}/f,\\
&\xi_{xxxy}=\{g_{xxxy}-(f_{xxxy}\xi+3f_{xxy}\xi_x+f_{xxx}\xi_y+3f_{xy}\xi_{xx}\\
&\qquad+3f_{xx}\xi_{xy}+f_y\xi_{xxx}+3f_x\xi_{xxy})\}/f,\\
&\xi_{xxyy}=\{g_{xxyy}-(f_{xxyy}\xi+2f_{xyy}\xi_x+2f_{xxy}\xi_y+f_{yy}\xi_{xx}\\
&\qquad+4f_{xy}\xi_{xy}+f_{xx}\xi_{yy}+2f_y\xi_{xxy}+2f_x\xi_{xyy})\}/f,\\
&\xi_{xyyy}=\{g_{xyyy}-(f_{xyyy}\xi+f_{yyy}\xi_x+3f_{xyy}\xi_y\\
&\qquad+3f_{yy}\xi_{xy}+3f_{xy}\xi_{yy}+3f_y\xi_{xyy}+f_x\xi_{yyy})\}/f,\\
&\xi_{yyyy}=\{g_{yyyy}-(f_{yyyy}\xi+4f_{yyy}\xi_y+6f_{yy}\xi_{yy}+4f_y\xi_{yyy})\}/f,\\
&\qquad\vdots
\end{aligned}\right\} \tag{3-12}$$

同様にして，

$$h(x, y)=f_3(x, y)+f_4(x, y),\quad k(x, y)=nf_3(x, y) \tag{3-13}$$

とするとき，f, g をそれぞれ h, k で置き換えれば，η に関する微係数を求めることができる．

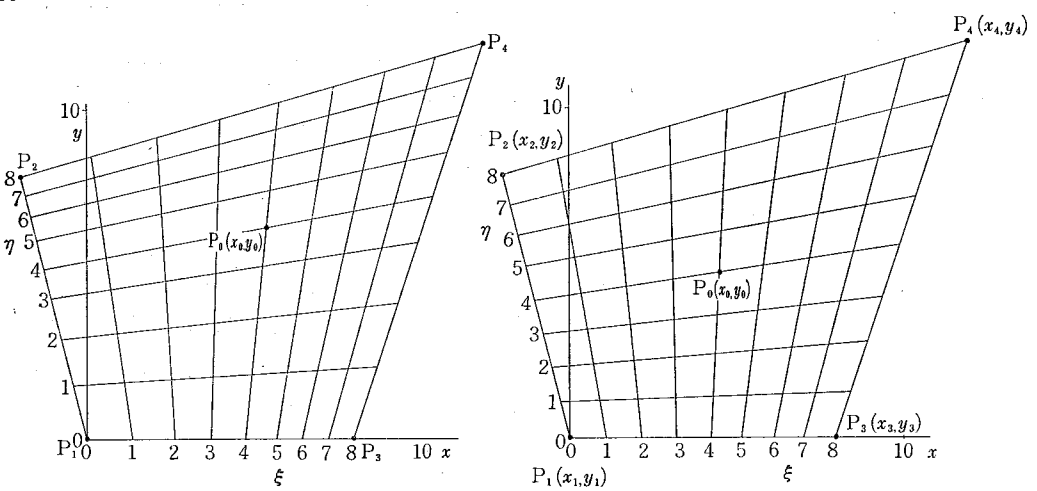
ここで差分が正確に評価されるためには，曲線格子系が正しく定義されていることが必要である．そのためには格子点座標が精度よく求められていることが必要である．

3.2.1 直線境界

図8に示されるように四辺形の四つの頂点 $P_1(x_1, y_1)$, $P_2(x_2, y_2)$, $P_3(x_3, y_3)$, $P_4(x_4, y_4)$ が与えられているときの境界線の式やその他の式は，下記のように表される．

$$\left.\begin{aligned}
&\varphi_1=\alpha_1+\beta_1x+\gamma_1y=0,\ \alpha_1=x_1y_2-x_2y_1,\ \beta_1=y_1-y_2,\ \gamma_1=x_2-x_1\\
&\varphi_2=\alpha_2+\beta_2x+\gamma_2y=0,\ \alpha_2=x_3y_4-x_4y_3,\ \beta_2=y_3-y_4,\ \gamma_2=x_4-x_3\\
&\varphi_3=\alpha_3+\beta_3x+\gamma_3y=0,\ \alpha_3=x_1y_3-x_3y_1,\ \beta_3=y_1-y_3,\ \gamma_3=x_3-x_1\\
&\varphi_4=\alpha_4+\beta_4x+\gamma_4y=0,\ \alpha_4=x_2y_4-x_4y_2,\ \beta_4=y_2-y_4,\ \gamma_4=x_4-x_2
\end{aligned}\right\} \tag{3-14}$$

つぎに $\xi=m/2$, $\eta=n/2$ の格子点 $P_0(x_0, y_0)$ を定め，重み係数を $w_m=1/\varphi_m(x_0, y_0)$, $(m=$



(a) $w_1=w_2=w_3=w_4=1$　　(b) $w_1=31,\ w_2=23,\ w_3=17,\ w_4=10$

図 8　直線4境界内直線格子系

Fig. 8　Mesh system inside a domain with linear four boundaries

1, 2, 3, 4) によって定め

$$a_m=w_m\alpha_m,\ b_m=w_m\beta_m,\ c_m=w_m\gamma_m,\ f_m=a_m+b_mx+c_my,\ (m=1,2,3,4) \tag{3-15}$$

として ξ-η 座標を次式によって定義する.

$$(m-\xi)f_1-\xi f_2=g-\xi f=0,\ f=a_1+a_2+(b_1+b_2)x+(c_1+c_2)y,\ g=m(a_1+b_1x+c_1y) \tag{3-16}$$

$$(n-\eta)f_3-\eta f_4=k-\eta h=0,\ h=a_3+a_4+(b_3+b_4)x+(c_3+c_4)y,\ k=n(a_3+b_3x+c_3y) \tag{3-17}$$

ここで, 次のような例題を考えてみる.

例題: $P_1(x_1=0, y_1=0)$, $P_2(x_2=-2, y_2=8)$, $P_3(x_3=8, y_3=0)$, $P_4(x_4=12, y_4=12)$, $m=n=8$.

図8(a)に示されるように重み係数をすべて1とした格子系は不均等な網目となるが, 領域の中心を $x_0=4.5$, $y_0=5.0$ とし, $\xi=\eta=4$ がこの点を通過するように重み係数を調整した図8(b)では, 一様な網目が得られている. ここでさらに,

$$f=f_0+f_xx+f_yy,\ g=g_0+g_xx+g_yy,\ f_0=a_1+a_2,$$
$$f_x=b_1+b_2,\ f_y=c_1+c_2,\ g_0=ma_1,\ g_x=mb_1,\ g_y=mc_1, \tag{3-18}$$

$$h=h_0+h_xx+h_yy,\ k=k_0+k_xx+k_yy,\ h_0=a_3+a_4,$$
$$h_x=b_3+b_4,\ h_y=c_3+c_4,\ k_0=na_3,\ k_x=nb_3,\ k_y=nc_3 \tag{3-19}$$

の置き換えを行うことによって, 式(3-16), (3-17)を

$$ax+by=f,\ a=g_x-f_x\xi,\ b=g_y-f_y\xi,\ f=-g_0+f_0\xi \tag{3-20}$$

$$cx+dy=g,\ c=k_x-h_x\xi,\ d=k_y-h_y\eta,\ g=-k_0+h_0\eta \tag{3-21}$$

の x, y に関する連立方程式に変形する. これを x, y について解き

$$x^*=(df-bg)/J,\ y^*=(-cf+ag)/J,\ J=ad-bc \tag{3-22}$$

によって ξ, η 格子点座標 x^*, y^* を求めることができる. 格子点座標が定まれば, この点における ξ, η の微係数は次式によって逐次求められる.

$$\left.\begin{aligned} &f^*=f_0+f_xx^*+f_yy^*,\ g^*=g_0+g_xx^*+g_yy^*,\\ &h^*=h_0+h_xx^*+h_yy^*,\ k^*=k_0+k_xx^*+k_yy^*\\ &\xi_x=(g_x-f_x\xi)/f^*,\xi_y=(g_y-f_y\xi)/f^*,\ \xi_{xx}=-2f_x\xi_x/f^*,\\ &\xi_{xy}=-(f_y\xi_x+f_x\xi_y)/f^*,\ \xi_{yy}=-2f_y\xi_y/f^* \end{aligned}\right\} \tag{3-23}$$

$$
\left.\begin{aligned}
&\xi_{xxx}=-3f_x\xi_{xx}/f^*,\ \xi_{xxy}=-(f_y\xi_{xx}+2f_x\xi_{xy})/f^*,\\
&\xi_{xyy}=-(2f_y\xi_{xy}+f_x\xi_{yy})/f^*,\ \xi_{yyy}=-3f_y\xi_{yy}/f^*,\\
&\xi_{xxxx}=-4f_x\xi_{xxx}/f^*,\ \xi_{xxxy}=-(f_y\xi_{xxx}+3f_x\xi_{xxy})/f^*,\\
&\xi_{xxyy}=-2(f_y\xi_{xxy}+f_x\xi_{xyy})/f^*,\\
&\xi_{xyyy}=-(3f_y\xi_{xyy}+f_x\xi_{yyy})/f^*,\ \xi_{yyyy}=-4f_y\xi_{yyy}/f^*
\end{aligned}\right\}
$$

$$
\left.\begin{aligned}
&\eta_x=(k_x-h_x\eta)/h^*,\ \eta_y=(k_y-h_y\eta)/h^*,\ \eta_{xx}=-2h_x\eta_x/h^*,\\
&\eta_{xy}=-(h_y\eta_x+h_x\eta_y)/h^*,\ \eta_{yy}=-2h_y\eta_y/h^*,\\
&\eta_{xxx}=-3h_x\eta_{xx}/h^*,\ \eta_{xxy}=-(h_y\eta_{xx}+2h_x\eta_{xy})/h^*,\\
&\eta_{xyy}=-(2h_y\eta_{xy}+h_x\eta_{yy})/h^*,\ \eta_{yyy}=-3h_y\eta_{yy}/h^*\\
&\eta_{xxxx}=-4h_x\eta_{xxx}/h^*,\ \eta_{xxxy}=-(h_y\eta_{xxx}+3h_x\eta_{xxy})/h^*,\\
&\eta_{xxyy}=-2(h_y\eta_{xxy}+h_x\eta_{xyy})/h^*,\\
&\eta_{xyyy}=-(3h_y\eta_{xyy}+h_x\eta_{yyy})/h^*,\ \eta_{yyyy}=-4h_y\eta_{yyy}/h^*,
\end{aligned}\right\}
\qquad (3\text{-}24)
$$

なお微係数は，2階の微分方程式のときは上記 ξ,η の2次まで，4階の微分方程式のときは4次まで必要である．

3.2.2 2次曲線境界

3個の点 $P_1(x_1,y_1)$, $P_2(x_2,y_2)$, $P_3(x_3,y_3)$ を通過する2次曲線は次式で表される．

$$\varphi(x,y)=a'+b'x+c'y+d'(x^2+y^2)=0 \qquad (3\text{-}25)$$

ただし，

$$
\left.\begin{aligned}
&\alpha_1=x_2y_3-x_3y_2,\ \beta_1=y_2-y_3,\ \gamma_1=x_3-x_2,\ f_1=x_1^2+y_1^2\\
&\alpha_2=x_3y_1-x_1y_3,\ \beta_2=y_3-y_1,\ \gamma_2=x_1-x_3,\ f_2=x_2^2+y_2^2\\
&\alpha_3=x_1y_2-x_2y_1,\ \beta_3=y_1-y_2,\ \gamma_3=x_2-x_1,\ f_3=x_3^2+y_3^2\\
&a'=\alpha_1f_1+\alpha_2f_2+\alpha_3f_3,\ b'=\beta_1f_1+\beta_2f_2+\beta_3f_3,\\
&c'=\gamma_1f_1+\gamma_2f_2+\gamma_3f_3,\ d'=-(\alpha_1+\alpha_2+\alpha_3)
\end{aligned}\right\}
\qquad (3\text{-}26)
$$

当然この3点を通る2次曲線は，上式に限られることはない．つぎに図9に示される点分布による曲線の式

$$\varphi_m(x,y)=a_m'+b_m'x+c_m'y+d_m'(x^2+y^2)=0,$$

$$f_m(x,y)=\varphi_m(x,y)/\varphi_m(x_0,y_0)=a_m+b_mx+c_my+d_m(x^2+y^2)=0 \quad (m=1,2,3,4)$$

を定めるとき曲線座標は次式によって表される．

ξ 座標：

$$(m-\xi)f_1-\xi f_2=ma_1-\xi(a_1+a_2)+\{mb_1-\xi(b_1+b_2)\}x+\{mc_1-\xi(c_1+c_2)\}y+\{md_1-\xi(d_1+d_2)\}(x^2+y^2)=0$$

$$\xi=g/f,\ g=m\{a_1+b_1x+c_1y+d_1(x^2+y^2)\},$$

$$f=a_1+a_2+(b_1+b_2)x+(c_1+c_2)y+(d_1+d_2)(x^2+y^2)$$

η 座標：

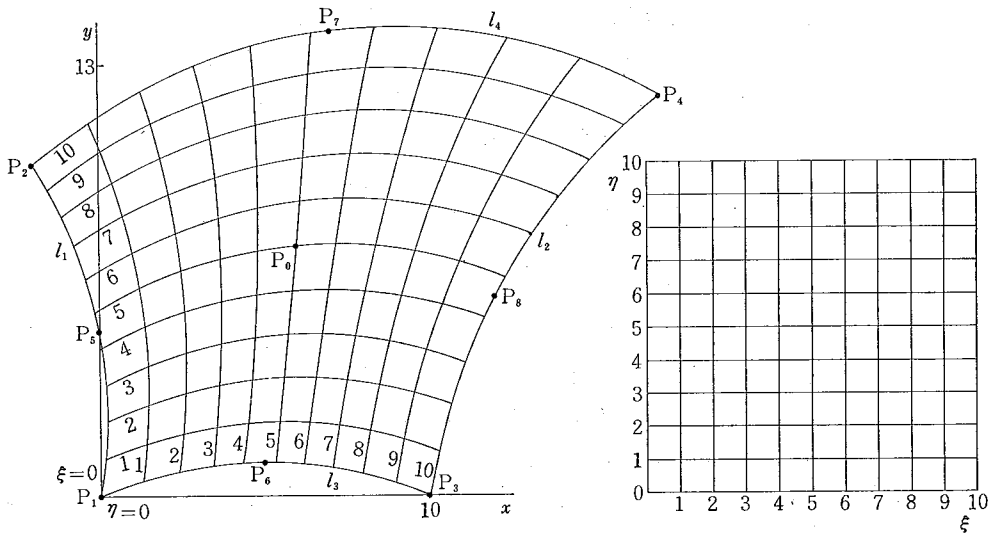
$$(n-\eta)f_3-\eta f_4=\eta a_3-\eta(a_3+a_4)+\{nb_3-\eta(b_3+b_4)\}x+\{nc_3-\eta(c_3+c_4)\}y+\{nd_3-\eta(d_3+d_4)\}(x^2+y^2)=0$$

$$\eta=k/h,\ k=n\{a_3+b_3x+c_3y+d_3(x^2+y^2)\},$$

$$h=a_3+a_4+(b_3+b_4)x+(c_3+c_4)y+(d_3+d_4)(x^2+y^2)$$

つぎに計算例を図9に示す．ここでは境界 l_1, l_2, l_3, l_4 の形を定める3個の点として，それぞれ (P_1,P_2,P_5), (P_3,P_4,P_8), (P_1,P_3,P_6), (P_2,P_4,P_7) を用いる．各点の座標ならびに標準化された境界の式 $f_m(x,y)$, の計算手順は下記のとおりである．

$P_1(x_1=0,y_1=0)$, $P_2(x_2=-2,y_2=10)$, $P_3(x_3=10,y_3=0)$, $P_4(x_4=17,y_4=12)$,



（a） 物理平面　　（b） 解析平面

図 9　4境界領域格子系

Fig. 9　Mesh system inside a domain with four boundaries

$P_5(x_5=0, y_5=5)$, $P_6(x_6=5, y_6=1)$ $P_7(x_7=7, y_7=14)$, $P_8(x_8=12, y_8=6)$

$\varphi_1(x,y)=270x-50y+10(x^2+y^2)$, $\varphi_2(x,y)=-8580+1038x-106y-18(x^2+y^2)$

$\varphi_3(x,y)=100x-240y-10(x^2+y^2)$, $\varphi_4(x,y)=10920+1034x-282y-58(x^2+y^2)$

領域の中心を $x_0=6.0$, $y_0=7.5$, 重み係数 $w_m=1/\varphi_m(x_0,y_0)$, $f_m(x,y)=\varphi_m(x,y)/\varphi_m(x_0,y_0)$ とすると,

$$f_1(x,y)=0.124567x-0.023068y+0.00461361(x^2+y^2),$$
$$f_2(x,y)=1.784711-0.215913x+0.022049y+0.00374415(x^2+y^2)$$
$$f_3(x,y)=-0.0471143x+0.113074y+0.00471143(x^2+y^2),$$
$$f_4(x,y)=1.130610+0.107056x-0.029197y-0.00600507(x^2+y^2)$$

となる.

3.3 多境界領域

この計算法の応用の範囲を拡大するためには，さらに多くの境界形，たとえば 5, 6, 7, … 境界によって囲まれた領域についても適用できることが望ましい．一般に境界の数は 5, 7, …等の奇数と，6, 8, …等の偶数に分けられる．ここでは，その代表として5境界と6境界について考察を行う．

3.3.1 5境界領域

図10(a)に示す5境界領域は図10(b)に示すように1本のカット（点線）を入れて，4境界領域(1)と3境界領域(2)に分解することができる．しかし，3境界領域は前述の理由によって曲線格子系を設定するため好ましくないので，この分解は採用しないこととする．図10(c)は境界 l_2 の途中よりカットを入れて，二つの4境界領域(1)，(2)に分解するものである．

3.3.2 6境界領域

図11(a)に示すように，一つの6境界領域は二つの4境界領域(1)，(2)に分解することができ，それぞれの小領域の中に曲線座標系 $\xi^{(1)}$, $\eta^{(1)}$, $\xi^{(2)}$, $\eta^{(2)}$ をつくることができる．ここで η をカットと交る座標とするとき，$n^{(1)}=n^{(2)}=n$ とすることが望ましい．なお

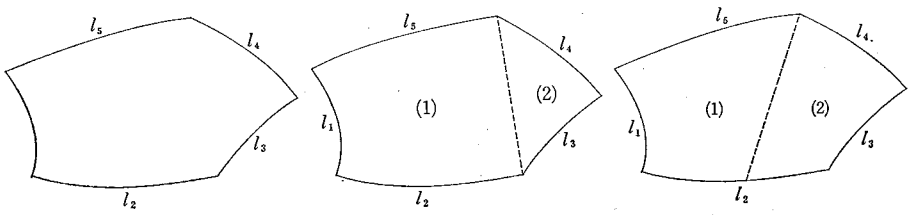
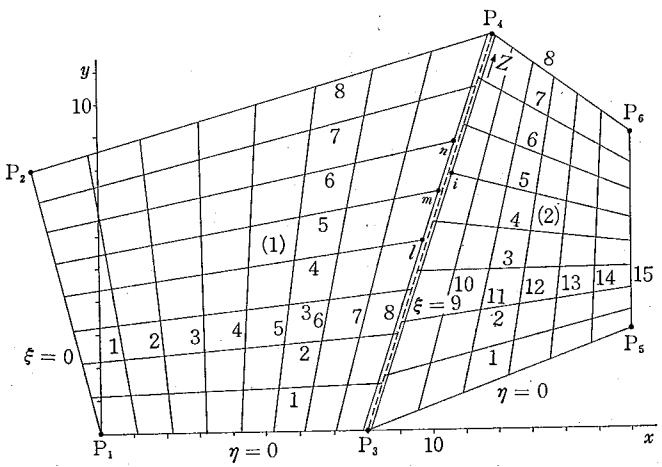
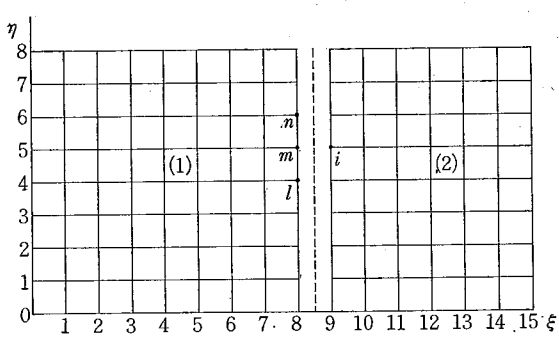


図 10　5境界領域の分解

Fig. 10　Resolution of a five boundary domain



(a)　物理平面格子系



(b)　解析平面格子系

図 11　境界領域の分解

Fig. 11　Resolution of a six boundary domain

図 10 では簡単にするため直線境界としたが，一般には曲線境界である．図 10 に見られるように物理平面においてη格子はカットを境として一般に不連続である．このようにカット部分ではその左右に 2 本のξ格子を設定することが必要で，その一つは小領域(1)，他は(2)に含まれる．したがって，格子点の節点にも異なる番号を付与しなければならない．実際カット線上の格子点は領域(1)と(2)との間における情報の交換場所である．このことはカット線上において変数の値は滑らかに変化することを仮定することによって達せられ，ここでは Lagrange 補間式が用いられる．いまカット線に沿う座標をsとして領域(1)に属する節点を l, m, n，(2)に属する節点を i として

$$k_{ii}u_i+k_{il}u_l+k_{im}u_m+k_{in}u_n=0$$

を満たすものとする．ただし，

$$k_{ii}=1,\ k_{il}=-(z_i-z_m)(z_i-z_n)/(z_l-z_m)(z_l-z_n),$$
$$k_{im}=-(z_i-z_l)(z_i-z_n)/(z_m-z_l)(z_m-z_n),$$
$$k_{in}=-(z_i-z_l)(z_i-z_m)/(z_n-z_l)(z_n-z_m)$$

によって与えられる．たとえば $z_i=z_m$ ならば $k_{il}=0$, $k_{im}=-1$, $k_{in}=0$ である．

3.4 2重連結領域

たとえば，有孔平板の応力解析や一般流内の翼断面周りの流れ解析等のように，2重連結領域問題として処理されることが好都合である．いま図12に示されるように内，外部境界 l_3, l_4 の式 $\varphi_3(x,y)=0$, $\varphi_4(x,y)=0$ が定義されているとき，

$$(n-\eta)f_3(x,y)-\eta f_4(x,y)=0,$$
$$f_3(x,y)=\varphi_3(x,y)/\varphi_3(x_0,y_0),\ f_4(x,y)=\varphi_4(x,y)/\varphi_4(x_0,y_0)$$

によって，曲線座標 η を定めることができる．

つぎに曲線 l_3 内の任意の点0を中心とする角を θ とし

$$\xi=m\theta/2\pi,\ \theta=\tan^{-1}(y/x)$$

によって直線座標 ξ を定める．ここに m,n は ξ,η の格子数である．

ここで注意を要することは内部境界線 l_3 が完全に閉じていること，すなわち内，外境界線ではさまれた領域内で至るところ $\varphi_3(x,y)=0$ を満たさないことである．たとえば図12に示される内部境界は，完全閉曲線であるから，外部境界との間に曲線格子系 η を定めることができる．これに対し，図13に示す内部境界は閉曲線 l_3 とともに枝 l_3' が存在するので上記の方法により所要の曲線座標 η を定めることはできない．このような場合には図14に示すように何本かのカットを入れて小領域に分割すればよい．また図15に示すように，内部または外部境界の形が一つの式では表されない場合にもカットを入れて小領域の集りに置き換えることができる．ここでは無限境界を十分大きな矩形によって置き換えたものである．当然，この境界は内部境界の影響が十分無視されるようなところに置くことが必要である．

4. 微系数の座標変換

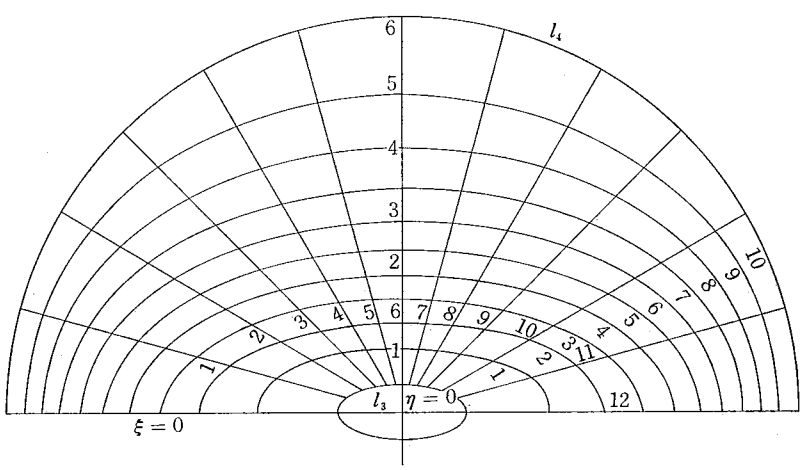


図 12 完全閉曲線，2重連結領域(1)（上半面）

Fig. 12 Doubly conected region (1) (perfect closed curve, upper surface)

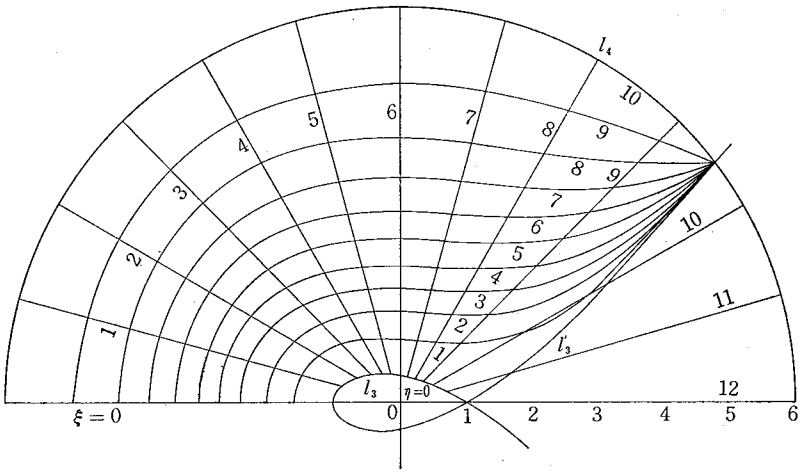


図 13 不完全閉曲線，2重連続領域(2)（上半面）

Fig. 13 Doubly conected region (2) (imperfect closed curve, upper surface)

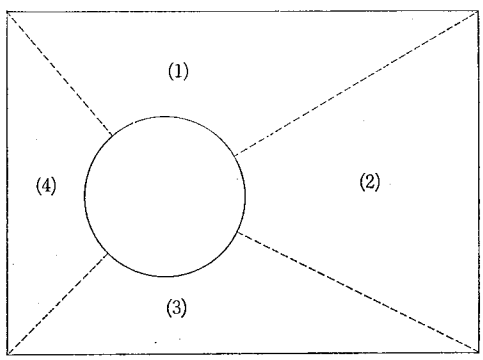


図 14 有孔矩形板の分割

Fig. 14 Resolution of flat plate with a hole

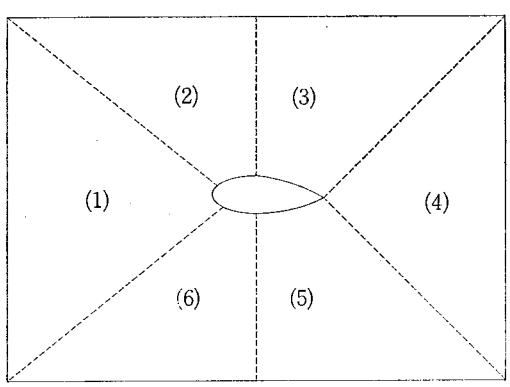


図 15 2次元翼断面周り無限領域の分割

Fig. 15 Resolution of infinite domain around a two dimensional aerofoil section

いま物理平面内における変数を $u(x,y)$ とし，その微係数の解析平面への変換式を求める．一般に x,y ならびに ξ,η に関する微係数を

$$\left.\begin{aligned} &a_1=u,\ a_2=u_x,\ a_3=u_y,\ a_4=u_{xx},\ a_5=u_{xy},\ a_6=u_{yy},\ a_7=u_{xxx},\\ &a_8=u_{xxy},\ a_9=u_{xyy},\ a_{10}=u_{yyy},\ a_{11}=u_{xxxx},\ a_{12}=u_{xxxy},\\ &a_{13}=u_{xxyy},\ a_{14}=u_{xyyy},\ a_{15}=u_{yyyy},\cdots \end{aligned}\right\} \qquad (4\text{-}1)$$

$$\left.\begin{aligned} &b_1=u,\ b_2=u_\xi,\ b_3=u_\eta,\ b_4=u_{\xi\xi},\ b_5=u_{\xi\eta},\ b_6=u_{\eta\eta},\ b_7=u_{\xi\xi\xi},\\ &b_8=u_{\xi\xi\eta},\ b_9=u_{\xi\eta\eta},\ b_{10}=u_{\eta\eta\eta},\ b_{11}=u_{\xi\xi\xi\xi},\ b_{12}=u_{\xi\xi\xi\eta},\\ &b_{13}=u_{\xi\xi\eta\eta},\ b_{14}=u_{\xi\eta\eta\eta},\ b_{15}=u_{\eta\eta\eta\eta},\cdots \end{aligned}\right\} \tag{4-2}$$

とし，その変換を

$$a_m=\sum_{n=1} c_{mn}b_n,\quad (m=1,2,3,\cdots) \tag{4-3}$$

とする．ただし

$c_{11}=1$

$c_{22}=\xi_x,\ c_{23}=\eta_x$

$c_{32}=\xi_y,\ c_{33}=\eta_y$

$c_{42}=\xi_{xx},\ c_{43}=\eta_{xx},\ c_{44}=\xi_x^2,\ c_{45}=2\xi_x\eta_x,\ c_{46}=\eta_x^2$

$c_{52}=\xi_{xy},\ c_{53}=\eta_{xy},\ c_{54}=\xi_x\xi_y,\ c_{55}=\xi_x\eta_y+\xi_y\eta_x,\ c_{56}=\eta_x\eta_y$

$c_{62}=\xi_{yy},\ c_{63}=\eta_{yy},\ c_{64}=\xi_y^2,\ c_{65}=2\xi_y\eta_y,\ c_{66}=\eta_y^2$

$c_{72}=\xi_{xxx},\ c_{73}=\eta_{xxx},\ c_{74}=3\xi_x\xi_{xx},\ c_{75}=3(\eta_x\xi_{xx}+\xi_x\eta_{xx}),$

$c_{76}=3\eta_x\eta_{xx},\ c_{77}=\xi_x^3,\ c_{78}=3\xi_x^2\eta_x,\ c_{79}=3\xi_x\eta_x^2,\ c_{7,10}=\eta_x^3$

$c_{82}=\xi_{xxy},\ c_{83}=\eta_{xxy},\ c_{84}=\xi_{xx}\xi_y+2\xi_x\xi_{xy},\ c_{85}=\xi_{xx}\eta_y+\xi_y\eta_{xx}+2(\xi_x\eta_{xy}+\eta_x\xi_{xy}),$

$c_{86}=\eta_y\eta_{xx}+2\eta_x\eta_{xy},\ c_{87}=\xi_x^2\xi_y,\ c_{88}=\xi_x(\xi_x\eta_y+2\xi_y\eta_x),\ c_{89}=\eta_x(2\xi_x\eta_y+\xi_y\eta_x),$

$c_{8,10}=\eta_x^2\eta_y$

$c_{92}=\xi_{xyy},\ c_{93}=\eta_{xyy},\ c_{94}=\xi_x\xi_{yy}+2\xi_y\xi_{xy},\ c_{95}=\xi_{yy}\eta_x+\xi_x\eta_{yy}+2(\xi_{xy}\eta_y+\xi_y\eta_{xy}),$

$c_{96}=\eta_x\eta_{yy}+2\eta_y\eta_{xy},\ c_{97}=\xi_x\xi_y^2,\ c_{98}=\xi_y(\xi_y\eta_x+2\xi_x\eta_y),\ c_{99}=\eta_y(\xi_x\eta_y+2\xi_y\eta_x),$

$c_{9,10}=\eta_x\eta_y^2$

$c_{10,2}=\xi_{yyy},\ c_{10,3}=\eta_{yyy},\ c_{10,4}=3\xi_y\xi_{yy},\ c_{10,5}=3(\xi_{yy}\eta_y+\xi_y\eta_{yy}),$

$c_{10,6}=3\eta_y\eta_{yy},\ c_{10,7}=\xi_y^3,\ c_{10,8}=3\xi_y^2\eta_y,\ c_{10,9}=3\xi_y\eta_y^2,\ c_{10,10}=\eta_y^3$

$c_{11,2}=\xi_{xxxx},\ c_{11,3}=\eta_{xxxx},\ c_{11,4}=4\xi_x\xi_{xxx}+3\xi_{xx}^2,\ c_{11,5}=4\xi_{xxx}\eta_x+6\xi_{xx}\eta_{xx}+4\xi_x\eta_{xxx},$

$c_{11,6}=4\eta_x\eta_{xxx}+3\eta_{xx}^2,\ c_{11,7}=6\xi_x^2\xi_{xx},\ c_{11,8}=6(2\xi_x\xi_{xx}\eta_x+\xi_x^2\eta_{xx}),$

$c_{11,9}=6(\xi_{xx}\eta_x^2+2\xi_x\eta_x\eta_{xx}),\ c_{11,10}=6\eta_x^2\eta_{xx},\ c_{11,11}=\xi_x^4,\ c_{11,12}=4\xi_x^3\eta_x,$

$c_{11,13}=6\xi_x^2\eta_x^2,\ c_{11,14}=4\xi_x\eta_x^3,\ c_{11,15}=\eta_x^4$

$c_{12,2}=\xi_{xxxy},\ c_{12,3}=\eta_{xxxy},\ c_{12,4}=\xi_y\xi_{xxx}+3\xi_x\xi_{xxy}+3\xi_{xx}\xi_{xy},$

$c_{12,5}=\xi_{xxx}\eta_y+3\xi_{xxy}\eta_x+3\xi_{xx}\eta_{xy}+3\xi_{xy}\eta_{xx}+3\xi_x\eta_{xxy}+\xi_y\eta_{xxx},$

$c_{12,6}=\eta_y\eta_{xxx}+3\eta_x\eta_{xxy}+3\eta_{xx}\eta_{xy},\ c_{12,7}=3\xi_x(\xi_y\xi_{xx}+\xi_x\xi_{xy}),$

$c_{12,8}=3\{(\xi_x\eta_y+\xi_y\eta_x)\xi_{xx}+2\xi_x\eta_x\xi_{xy}+\xi_x\xi_y\eta_{xx}+\xi_x^2\eta_{xy}\},$

$c_{12,9}=3\{\eta_x\eta_y\xi_{xx}+\eta_x^2\xi_{xy}+(\xi_x\eta_y+\xi_y\eta_x)\eta_{xx}+2\xi_x\eta_x\eta_{xy}\},$

$c_{12,10}=3\eta_x(\eta_y\eta_{xx}+\eta_x\eta_{xy}),\ c_{12,11}=\xi_x^3\xi_y,\ c_{12,12}=\xi_x^3\eta_y+3\xi_x^2\xi_y\eta_x,$

$c_{12,13}=3\xi_x\eta_x(\xi_x\eta_y+\xi_y\eta_x),\ c_{12,14}=\eta_x^2(3\xi_x\eta_y+\xi_y\eta_x),\ c_{12,15}=\eta_x^3\eta_y,$

$c_{13,2}=\xi_{xxyy},\ c_{13,3}=\eta_{xxyy},\ c_{13,4}=2\xi_x\xi_{xyy}+2\xi_y\xi_{xxy}+2\xi_{xy}^2+\xi_{xx}\xi_{yy},$

$c_{13,5}=2\eta_y\xi_{xxy}+2\eta_x\xi_{xyy}+2\xi_y\eta_{xxy}+2\xi_x\eta_{xyy}+\xi_{xx}\eta_{yy}+\xi_{yy}\eta_{xx}+4\xi_{xy}\eta_{xy},$

$c_{13,6}=2\eta_x\eta_{xyy}+2\eta_y\eta_{xxy}+2\eta_{xy}^2+\eta_{xx}\eta_{yy},\ c_{13,7}=\xi_x^2\xi_{yy}+4\xi_x\xi_y\xi_{xy}+\xi_y^2\xi_{xx},$

$c_{13,8}=2\xi_y\eta_y\xi_{xx}+4(\xi_x\eta_y+\xi_y\eta_x)\xi_{xy}+2\xi_x\eta_x\xi_{yy}+\xi_y^2\eta_{xx}+4\xi_x\xi_y\eta_{xy}+\xi_x^2\eta_{yy},$

$c_{13,9}=\eta_y^2\xi_{xx}+4\eta_x\eta_y\xi_{xy}+\eta_x^2\xi_{yy}+2\xi_y\eta_y\eta_{xx}+4(\xi_x\eta_y+\xi_y\eta_x)\eta_{xy}+2\xi_x\eta_x\eta_{yy},$

$c_{13,10}=\eta_y^2\eta_{xx}+4\eta_x\eta_y\eta_{xy}+\eta_x^2\eta_{yy},\ c_{13,11}=\xi_x^2\xi_y^2,\ c_{13,12}=2\xi_x\xi_y(\xi_x\eta_y+\xi_y\eta_x),$

$c_{13,13}=\xi_x^2\eta_y^2+4\xi_x\xi_y\eta_x\eta_y+\xi_y^2\eta_x^2,\ c_{13,14}=2\eta_x\eta_y(\xi_x\eta_y+\xi_y\eta_x),\ c_{13,15}=\eta_x^2\eta_y^2$

$c_{14,2}=\xi_{xyyy},\ c_{14,3}=\eta_{xyyy},\ c_{14,4}=\xi_x\xi_{yyy}+3\xi_y\xi_{xyy}+3\xi_{xy}\xi_{yy},$

$c_{14,5}=3\eta_y\xi_{xyy}+\eta_x\xi_{yyy}+3\xi_y\eta_{xyy}+\xi_x\eta_{yyy}+3\xi_{yy}\eta_{xy}+3\xi_{xy}\eta_{yy},$

$$c_{14,6}=\eta_x\eta_{yyy}+3\eta_{xy}\eta_{yy}+3\eta_y\eta_{xyy},\ c_{14,7}=3\xi_y(\xi_x\xi_{yy}+\xi_y\xi_{xy}),$$
$$c_{14,8}=3\{\xi_x\xi_y\eta_{yy}+(\xi_x\eta_y+\xi_y\eta_x)\xi_{yy}+2\xi_y\eta_y\xi_{xy}+\xi_y{}^2\eta_{xy}\},$$
$$c_{14,9}=3\{\eta_y{}^2\xi_{xy}+\eta_x\eta_y\xi_{yy}+2\xi_y\eta_y\eta_{xy}+(\xi_x\eta_y+\xi_y\eta_x)\eta_{yy}\},$$
$$c_{14,10}=3\eta_y(\eta_y\eta_{xy}+\eta_x\eta_{yy}),\ c_{14,11}=\xi_x\xi_y{}^3,\ c_{14,12}=\xi_y{}^2(3\xi_x\eta_y+\xi_y\eta_x),$$
$$c_{14,13}=3\xi_y\eta_y(\xi_x\eta_y+\xi_y\eta_x),\ c_{14,14}=\eta_y{}^2(\xi_x\eta_y+3\xi_y\eta_x),\ c_{14,15}=\eta_x\eta_y{}^3$$
$$c_{15,2}=\xi_{yyyy},\ c_{15,3}=\eta_{yyyy},\ c_{15,4}=4\xi_y\xi_{yyy}+3\xi_{yy}{}^2,\ c_{15,5}=4\eta_y\xi_{yyy}+6\xi_{yy}\eta_{yy}+4\xi_y\eta_{yyy},$$
$$c_{15,6}=4\eta_y\eta_{yyy}+3\eta_{yy}{}^2,\ c_{15,7}=6\xi_y{}^2\xi_{yy},\ c_{15,8}=6(2\xi_y\eta_y\xi_{yy}+\xi_y{}^2\eta_{yy}),$$
$$c_{15,9}=6(\eta_y{}^2\xi_{yy}+2\xi_y\eta_y\eta_{yy}),\ c_{15,10}=6\eta_y{}^2\eta_{yy},\ c_{15,11}=\xi_y{}^4,\ c_{15,12}=4\xi_y{}^3\eta_y,$$
$$c_{15,13}=6\xi_y{}^2\eta_y{}^2,\ c_{15,14}=4\eta\xi_{yy}{}^3,\ c_{15,15}=\eta_y{}^4$$

なお，正規直交曲線座標系では下記のように整理される．

$$c_{22}=\xi_x,\ c_{23}=-\xi_y$$
$$c_{32}=\xi_y,\ c_{33}=\xi_x$$
$$c_{42}=\xi_{xx},\ c_{43}=-\xi_{xy},\ c_{44}=\xi_x{}^2,\ c_{45}=-2\xi_x\xi_y,\ c_{46}=\xi_y{}^2$$
$$c_{52}=\xi_{xy},\ c_{53}=\xi_{xx},\ c_{54}=\xi_x\xi_y,\ c_{55}=\xi_x{}^2-\xi_y{}^2,\ c_{56}=-\xi_x\xi_y$$
$$c_{62}=-\xi_{xx},\ c_{63}=\xi_{xy},\ c_{64}=\xi_y{}^2,\ c_{65}=2\xi_x\xi_y,\ c_{66}=\xi_x{}^2$$
$$c_{7,2}=\xi_{xxx},\ c_{7,3}=-\xi_{xxy},\ c_{7,4}=3\xi_x\xi_{xx},\ c_{7,5}=-3(\xi_x\xi_{xy}+\xi_y\xi_{xx}),\ c_{7,6}=3\xi_y\xi_{xy},$$
$$c_{7,7}=\xi_x{}^3,\ c_{7,8}=-3\xi_x{}^2\xi_y,\ c_{7,9}=3\xi_x\xi_y{}^2,\ c_{7,10}=-\xi_y{}^3$$
$$c_{8,2}=\xi_{xxy},\ c_{8,3}=\xi_{xxx},\ c_{8,4}=2\xi_x\xi_{xy}+\xi_y\xi_{xx},\ c_{8,5}=3(\xi_x\xi_{xx}-\xi_y\xi_{xy}),$$
$$c_{8,6}=-(\xi_x\xi_{xy}+2\xi_y\xi_{xx}),\ c_{8,7}=\xi_x{}^2\xi_y,\ c_{8,8}=\xi_x(\xi_x{}^2-2\xi_y{}^2),$$
$$c_{8,9}=\xi_y(-2\xi_x{}^2+\xi_y{}^2),\ c_{8,10}=\xi_x\xi_y{}^2$$
$$c_{9,2}=-\xi_{xxx},\ c_{9,3}=\xi_{xxy},\ c_{9,4}=-\xi_x\xi_{xx}+2\xi_y\xi_{xy},\ c_{9,5}=3(\xi_x\xi_{xy}+\xi_y\xi_{xx}),$$
$$c_{9,6}=2\xi_x\xi_{xx}-\xi_y\xi_{xy},\ c_{9,7}=\xi_x\xi_y{}^2,\ c_{9,8}=\xi_y(2\xi_x{}^2-\xi_y{}^2),$$
$$c_{9,9}=\xi_x(\xi_x{}^2-2\xi_y{}^2),\ c_{9,10}=-\xi_x{}^2\xi_y$$
$$c_{10,2}=\xi_{xxy},\ c_{10,3}=-\xi_{xxx},\ c_{10,4}=-3\xi_y\xi_{xx},\ c_{10,5}=3(-\xi_x\xi_{xx}+\xi_y\xi_{xy}),\ c_{10,6}=3\xi_x\xi_{xy},$$
$$c_{10,7}=\xi_y{}^3,\ c_{10,8}=3\xi_x\xi_y{}^2,\ c_{10,9}=3\xi_y\xi_x{}^2,\ c_{10,10}=\xi_x{}^3$$
$$c_{11,2}=\xi_{xxxx},\ c_{11,3}=-\xi_{xxxy},\ c_{11,4}=4\xi_x\xi_{xxx}+3\xi_{xx}{}^2,$$
$$c_{11,5}=-(4\xi_x\xi_{xxy}+6\xi_{xx}\xi_{xy}+4\xi_y\xi_{xxx}),\ c_{11,6}=4\xi_y\xi_{xxy}+3\xi_{xy}{}^2,\ c_{11,7}=6\xi_x{}^2\xi_{xx},$$
$$c_{11,8}=-6(2\xi_x\xi_y\xi_{xx}+\xi_x{}^2\xi_{xy}),\ c_{11,9}=6(\xi_y{}^2\xi_{xx}+2\xi_x\xi_y\xi_{xy}),\ c_{11,10}=-6\xi_y{}^2\xi_{xy},$$
$$c_{11,11}=\xi_{x4},\ c_{11,12}=-4\xi_x{}^3\xi_y,\ c_{11,13}=6\xi_x{}^2\xi_y{}^2,\ c_{11,14}=-4\xi_x\xi_y{}^3,\ c_{11,15}=\xi_y{}^4$$
$$c_{12,2}=\xi_{xxxy},\ c_{12,3}=\eta_{xxxy},\ c_{12,4}=\xi_y\xi_{xxx}+3\xi_x\xi_{xxy}+3\xi_{xx}\xi_{xy},$$
$$c_{12,5}=4\xi_x\xi_{xxx}-4\xi_y\xi_{xxy}+3\xi_{xx}{}^2-3\xi_{xy}{}^2,\ c_{12,6}=\xi_x\xi_{xxy}-3\xi_y\xi_{xxx}-3\xi_{xx}\xi_{xy},$$
$$c_{12,7}=3\xi_x(\xi_y\xi_{xx}+\xi_x\xi_{xy}),\ c_{12,8}=3\{(2\xi_x{}^2-\xi_y{}^2)\xi_{xx}-3\xi_x\xi_y\xi_{xy}\},$$
$$c_{12,9}=3\{-3\xi_x\xi_y\xi_{xx}+(-\xi_x{}^2+2\xi_y{}^2)\xi_{xy}\},\ c_{12,10}=3\xi_y(\xi_x\xi_{xy}+\xi_y\xi_{xx}),$$
$$c_{12,11}=\xi_x{}^3\xi_y,\ c_{12,12}=\xi_x{}^4-3\xi_x{}^2\xi_y{}^2,\ c_{12,13}=3\xi_x\xi_y(-\xi_x{}^2+\xi_y{}^2),$$
$$c_{12,14}=\xi_y{}^2(3\xi_x{}^2-\xi_y{}^2),\ c_{12,15}=-\xi_x\xi_y{}^3$$
$$c_{13,2}=\xi_{xxyy},\ c_{13,3}=\xi_{xxxy},\ c_{13,4}=-2\xi_x\xi_{xxx}+2\xi_y\xi_{xxy}-\xi_{xx}{}^2+2\xi_{xy}{}^2,$$
$$c_{13,5}=4(\xi_x\xi_{xxy}+\xi_y\xi_{xxx})+5\xi_{xx}\xi_{xy}-\xi_{yy}\xi_{xy},$$
$$c_{13,6}=2\xi_x\xi_{xxx}-2\xi_y\xi_{xxy}+2\xi_{xx}{}^2-\xi_{xy}{}^2,\ c_{13,7}=-\xi_x{}^2\xi_{xx}+4\xi_x\xi_y\xi_{xy}-\xi_y{}^2\xi_{xx},$$
$$c_{13,8}=5\xi_x{}^2\xi_{xy}+8\xi_x\xi_y\xi_{xx}-5\xi_y{}^2\xi_{xy},\ c_{13,9}=5\xi_x{}^2\xi_{xx}-8\xi_x\xi_y\xi_{xy}-5\xi_y{}^2\xi_{xx},$$
$$c_{13,10}=-\xi_x{}^2\xi_{xy}-4\xi_x\xi_y\xi_{xx}+\xi_y{}^2\xi_{xy},\ c_{13,11}=\xi_x{}^2\xi_y{}^2,\ c_{13,12}=2\xi_x\xi_y(\xi_x{}^2-\xi_y{}^2),$$
$$c_{13,13}=\xi_x{}^4-4\xi_x{}^2\xi_y{}^2+\xi_y{}^4,\ c_{13,14}=2\xi_x\xi_y(-\xi_x{}^2+\xi_y{}^2),\ c_{13,15}=\xi_x{}^2\xi_y{}^2$$
$$c_{14,2}=\xi_{xyyy},\ c_{14,3}=-\xi_{xxxx},\ c_{14,4}=\xi_x\xi_{xxy}-3\xi_y\xi_{xxx}-3\xi_{xx}\xi_{xy},$$

$$c_{14,5}=-4\xi_x\xi_{xxx}+2\xi_y\xi_{xxy}+3(-\xi_{xx}{}^2+\xi_{xy}{}^2),\ c_{14,6}=3\xi_x\xi_{xxy}+\xi_y\xi_{xxx}+3\xi_{xx}\xi_{xy},$$
$$c_{14,7}=3\xi_y(-\xi_x\xi_{xx}+\xi_y\xi_{xy}),\ c_{14,8}=3(-\xi_x{}^2\xi_{xx}+3\xi_x\xi_y\xi_{xy}+2\xi_y{}^2\xi_{xx}),$$
$$c_{14,9}=3(2\xi_{x2}\xi_{xy}+3\xi_x\xi_y\xi_{xx}-\xi_y{}^2\xi_{xy}),\ c_{14,10}=3\xi_x(\xi_x\xi_{xx}-\xi_y\xi_{xy}),$$
$$c_{14,11}=\xi_x\xi_y{}^3,\ c_{14,12}=\xi_y{}^2(3\xi_x{}^2-\xi_y{}^2),\ c_{14,13}=3\xi_x\xi_y(\xi_x{}^2-\xi_y{}^2)$$
$$c_{14,14}=\xi_x{}^2(\xi_x{}^2-3\xi_y{}^2),\ c_{14,15}=-\xi_x{}^3\xi_y$$
$$c_{15,2}=\xi_{yyy},\ c_{15,3}=-\xi_{xxy},\ c_{15,4}=4\xi_y\xi_{xxy}+3\xi_{yy}{}^2,\ c_{15,5}=4\xi_x\xi_{xxy}-4\xi_y\xi_{xxx}-6\xi_{xx}\xi_{xy}$$
$$c_{15,6}=3\xi_{xy}{}^2-4\xi_x\xi_{xxx},\ c_{15,7}=6\xi_y{}^2\xi_{xy},\ c_{15,8}=6(-2\xi_x\xi_y\xi_{xx}+\xi_y{}^2\xi_{xy}),$$
$$c_{15,9}=6(-\xi_x{}^2\xi_{xx}+2\xi_x\xi_y\xi_{xy}),\ c_{15,10}=6\xi_x{}^2\xi_{xy}\ c_{15,11}=\xi_y{}^4,\ \xi_{15,12}=4\xi_x\xi_y{}^3,$$
$$\xi_{15,13}=6\xi_x{}^2\xi_y{}^2,\ c_{15,14}=4\xi_x{}^3\xi_y,\ c_{15,15}=\xi_x{}^4$$

5. 1変数，2階微分方程式の差分化

変数 $u(x,y)$ に関する2階微分方程式を

$$A_1a_1+A_2a_2+A_3a_3+A_4a_4+A_5a_5+A_6a_6=f \tag{5-1}$$

とし，上式に変換式(4-3)を行うことによって，解析平面内における微分方程式

$$B_1b_1+B_2b_2+B_3b_3+B_4b_4+B_5b_5+B_6b_6=f \tag{5-2}$$

を導く．ただし

$$B_n=\sum_{m=1}^{6}A_mc_{mn} \tag{5-3}$$

すなわち，

$$\left.\begin{aligned}
B_1&=A_1\\
B_2&=A_2c_{22}+A_3c_{32}+A_4c_{42}+A_5c_{52}+A_6c_{62}\\
B_3&=A_2c_{23}+A_3c_{33}+A_4c_{43}+A_5c_{53}+A_6c_{63}\\
B_4&=A_4c_{44}+A_5c_{54}+A_6c_{64}\\
B_5&=A_4c_{45}+A_5c_{55}+A_6c_{65}\\
B_6&=A_4c_{46}+A_5c_{56}+A_6c_{66}
\end{aligned}\right\} \tag{5-4}$$

である．

つぎに微分方程式(5-2)の差分化を行うため，図16に示す差分家族に関する差分式を示す．

	$b_2=u_\xi$	$b_3=u_\eta$	$b_4=u_{\xi\xi}$	$b_5=u_{\xi\eta}$	$b_6=u_{\eta\eta}$
(a)	$(-u_j+u_k)/2$	$(-u_l+u_m)/2$	$-2u_i+u_j+u_k$	$u_i-u_k-u_m+u_n$	$-2u_i+u_l+u_m$
(l_1)	$(-3u_i+4u_j-u_k)/2$	$(-u_l+u_m)/2$	$u_i-2u_j+u_k$	$u_i-u_j-u_m+u_n$	$-2u_i+u_l+u_m$
(l_2)	$(3u_i+u_j-4u_k)/2$	$(-u_l+u_m)/2$	$u_i+u_j-2u_k$	$-u_i+u_k+u_m-u_n$	$-2u_i+u_l+u_m$
(l_3)	$(-u_j+u_k)/2$	$(-3u_i+4u_l-u_m)/2$	$-2u_i+u_j+u_k$	$u_i-u_k-u_l+u_n$	$u_i-2u_l+u_m$
(l_4)	$(-u_j+u_k)/2$	$(3u_i+u_l-4u_m)/2$	$-2u_i+u_j+u_k$	$-u_i+u_k+u_l-u_n$	$u_i+u_l-2u_m$
(c_1)	$(-3u_i+4u_j-u_k)/2$	$(-3u_i+4u_l-u_m)/2$	$u_i-2u_j+u_k$	$u_i-u_j-u_l+u_n$	$u_i-2u_l+u_m$
(c_2)	$(3u_i+u_j-4u_k)/2$	$(-3u_i+4u_l-u_m)/2$	$u_i+u_j-2u_k$	$-u_i+u_k+u_l-u_n$	$u_i-2u_l+u_m$
(c_3)	$(-3u_i+4u_j-u_k)/2$	$(3u_i+u_l-4u_m)/2$	$u_i-2u_j+u_k$	$-u_i+u_j+u_m-u_n$	$u_i+u_l-2u_m$
(c_4)	$(3u_i+u_j-4u_k)/2$	$(3u_i+u_l-4u_m)/2$	$u_i+u_j-2u_k$	$u_i-u_k-u_m+u_n$	$u_i+u_l-2u_m$

(5-5)

つぎに，式(5-2)によって差分家族 i に関する連立方程式を導く．

$$\sum_{1}^{6}B_nb_n=\sum_j k_{ij}u_j=f_i \tag{5-6}$$

係数行列 k_{ij} を表1に示す．

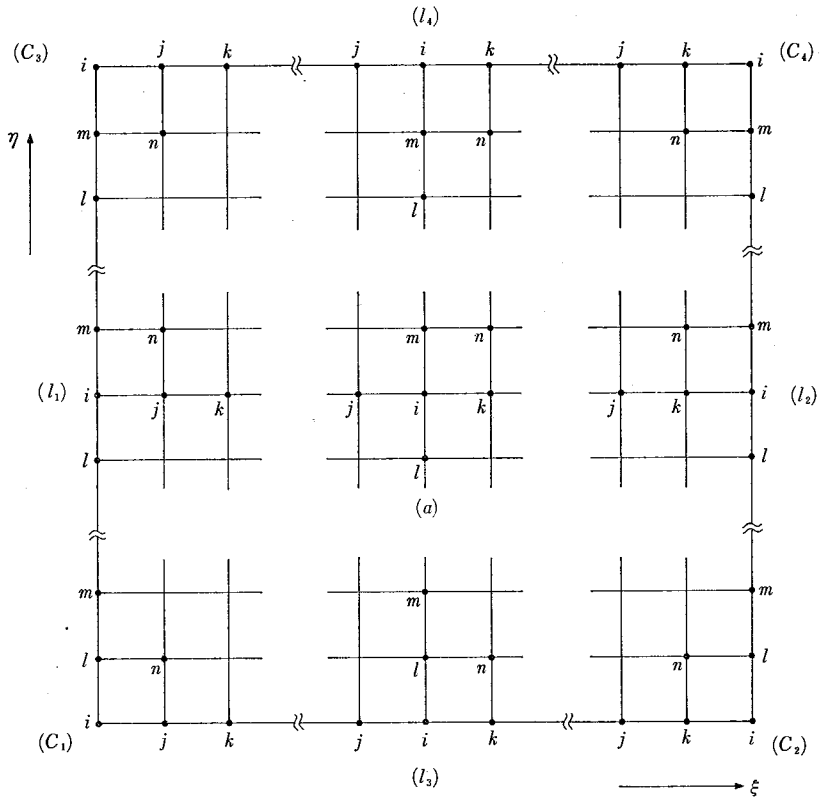


図 16 差分家族節点配置

Fig. 16 Node distribution of difference family

表 1 係数行列 k_{ij}

Table. 1 Matrix of coefficients k_{ij}

	(a)	(l_1)	(l_2)
k_{ii}	$B_1-2B_4+B_5-2B_6$	$B_1-1.5B_2+B_4+B_5-2B_6$	$B_1+1.5B_2+B_4-B_5-2B_6$
k_{ij}	$-0.5B_2+B_4$	$2B_2-2B_4-B_5$	$0.5B_2+B_4$
k_{ik}	$0.5B_2+B_4-B_5$	$-0.5B_2+B_4$	$-2B_2-2B_4+B_5$
k_{il}	$-B_3+B_6$	$-0.5B_3+B_6$	$-0.5B_3+B_5$
k_{im}	$0.5B_3-B_5+B_6$	$0.5B_3-B_5+B_6$	$0.5B_3+B_5+B_6$
k_{in}	B_5	B_5	$-B_5$

	(l_3)	(l_4)	(c_1)
k_{ii}	$B_1-1.5B_3-2B_4+B_5+B_6$	$B_1+1.5B_3-2B_4-B_5+B_6$	$B_1-1.5B_2-1.5B_3+B_4+B_5+B_6$
k_{ij}	$-0.5B_2+B_4$	$-0.5B_2+B_4$	$2B_2-2B_4-B_5$
k_{ik}	$0.5B_2+B_5-2B_6$	$0.5B_2+B_4+B_5$	$-0.5B_2+B_4$
k_{il}	$2B_3-B_5-2B_6$	$0.5B_3+B_6$	$2B_3-B_5-2B_6$
k_{im}	$-0.5B_3+B_6$	$-2B_3+B_5-2B_6$	$-0.5B_3+B_6$
k_{in}	B_5	$-B_5$	B_5

	(c_2)	(c_3)	(c_4)
k_{ii}	$B_1+1.5B_2-1.5B_3+B_4-B_5+B_6$	$B_1-1.5B_2+1.5B_3+B_4-B_5+B_6$	$B_1+1.5B_2+1.5B_3+B_4+B_5+B_6$
k_{ij}	$0.5B_2+B_4$	$2B_2-2B_4+B_5$	$0.5B_2+B_4$
k_{ik}	$-2B_2-2B_4+B_5$	$-0.5B_2+B_4$	$-(2B_2+2B_4+B_5)$
k_{il}	$2B_3+B_5-2B_6$	$-0.5B_3+B_6$	$0.5B_3+B_6$
k_{im}	$-0.5B_3+B_6$	$-2B_3+B_5-2B_6$	$-(2B_3+B_5+2B_6)$
k_{in}	$-B_5$	$-B_5$	B_5

5.1 3 次 精 度

座標変換された微分方程式について差分化の精度を向上するためには展開次数を多くとることが必要で，ここでは図 17 に示す 3 次精度の差分家族による差分化を行うこととする．領域内では図 17(a) の差分家族によって節点 i における変数と，その微係数の値は，

$$u=u_i,\ u_\xi=(-3u_i-2u_j+6u_k-u_l)/6,\ u_\eta=(-3u_i-2u_m+6u_n-u_p)/6$$

$$u_{\xi\xi}=-2u_i+u_j+u_k,\ u_{\xi\eta}=(4u_i-5u_k+u_l-5u_n+u_p+6u_q-u_r-u_s)/2,$$

$$u_{\eta\eta}=-2u_i+u_m+u_n$$

によって与えられ，下記のような置き換えによって

$$B_1u+B_2u_\xi+B_3u_\eta+B_4u_{\xi\xi}+B_5u_{\xi\eta}+B_6u_{\eta\eta}$$

$$=k_{ii}u_i+k_{ij}u_j+k_{ik}u_k+k_{il}u_l+k_{im}u_m+k_{in}u_n+k_{ip}u_p+k_{iq}u_q+k_{ir}u_r+k_{is}u_s=f$$

と変形できる．ただし，係数は次のとおりである．

$$k_{ii}=B_1-B_2/2-B_3/2-2B_4+2B_5-2B_6,\ k_{ij}=-B_2/3+B_4,\ k_{ik}=B_2+B_4-5B_5/2,$$

$$k_{il}=-B_2/6+B_5/2\ k_{im}=-B_3/3+B_6,\ k_{in}=B_3-5B_5/2+B_6,,\ k_{ip}=-B_3/6+B_5/2,$$

$$k_{iq}=3B_5,\ k_{ir}=-B_5/2,\ k_{is}=-B_5/2$$

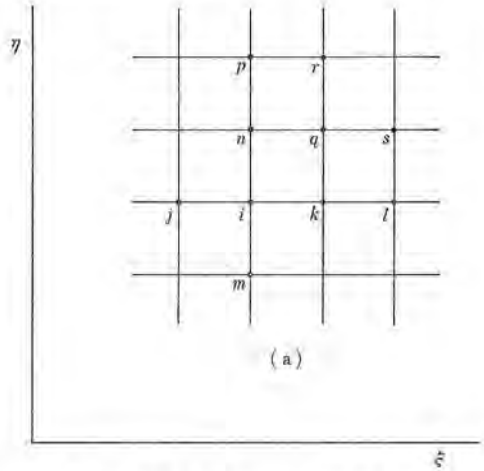


図 17 領域内 3 次精度差分家族節点配置

Fig. 17 Node distribution of difference family with third order accuracy

6. お わ り に

これまで述べたところはこの研究の糸口にすぎず，利用の範囲を拡大するためには，さらに多くの試行が必要である．たとえば多元，高階微分方程式，さらに非線形微分方程式への適用が可能となるよう，数値計算法の改善を行うことが必要と思われる．

執筆者紹介 藤 野 勉 (Tsutomu Fujino)

明治 45 年生，昭和 11 年東京帝国大学理学部物理科卒業，同年三菱重工(株)入社，主に応力・振動・流体力学等の解析法（有限要素法を含む）の研究に従事，32 年，工学博士号を取得，47 年，同社技術本部顧問となる．また，48年より東海大学工学部教授，52 年依嘱教授を歴任．51 年より日本ユニバック（株）の技術顧問となり，現在に至る．「コンピュータによる構造工学講座 II-4-B—熱伝導と熱応力」（培風館，1972）等の著書がある．

概　説

連続体解析のための新しい差分法について

On Flexible Finite Difference Method for Analysis of Continuum

藤　野　　勉

〈編集部注〉 執筆者は 1981 年からこれまでに 5 回にわたって差分法について本誌上で論じてきた．これらを概観し，今回報告する．

1. ま え が き

弾性体，流体等，連続体の挙動は微分方程式によって記述されている．しかるに連続的な表現である微分方程式をそのまま計算機により数値的に解くことはできず，なんらかの手段によって離散的な代数方程式の形に置き換えることが必要である．ここでは，このような手続きを定式化とよぶこととする．定式化の方法論としては従来，差分法，有限要素法，級数展開法，Rayleigh-Ritz 法，選点法または境界要素法等が存在するが，最近主として利用されているのは差分法と有限要素法であると思われる．歴史的には等間隔直交直線格子系による差分法がまず用いられ多くの学術的な問題の処理が行れた．しかし，この方法は複雑な幾何学的形状をもつ実用的な工学問題の処理に当たり，境界適合性が悪く，その後開発された有限要素法にその座をゆずることとなった．しかし，差分法も必ずしも直交直線格子系のみによる必要はなく，曲線格子系の採用等によって弾性的に節点配置を行うことから境界適合性を改善することができるので，有限要素法と同様に任意形状の連続体の解析に利用することができる．ここでは，このような差分法を総称して新しい差分法とした．つぎに，差分法全体について簡単に概説を行うこととする．差分法はこれを大別して，

① 直交直線格子系

② 任意節点配置 { Taylor 展開法 / 補間関数法 / アイソパラメタ法 }　③ 座標変換 { 直交曲線格子系 { 構成方程式法 / 微分方程式法 } / 非直交曲線格子系―微分方程式法 }

とすることができる．

直交直線格子系による差分式は，非常に簡単で高精度であるが境界適合性に劣っている．しかし，他のすべての差分法の基礎である．任意節点配置差分法は上表のようにさらに分類することができるが，その詳細については本技報第 0 号[1]，第 4 号[2]に述べてある．とくに 4, 6, 8 階等，高階の微分方程式はそれぞれ 2，3，4 個の境界条件式をもつので，任意節点配置差分法で解く場合境界外にそれぞれ 1，2，3 列のダミー節点を設けることが必要で，詳しくは技報第 7 号[4]に述べている．座標変換差分法は境界に適合する曲線座標系を領域内に定義することによって解析(曲線座標)空間内に直交直線格子系をつくり，変換された微分方程式を差分化するものである．曲線座標系は直交と非直交に分けられる．一般に任意形状の境界をもつ領域内に直交曲線座標系を定めることは，非常に困難で極座標系等のように既知の座標系に限られるものと思われる．この座標系による数式化（連続体の支配方程式を定める）には構成方程式と微分方程式とがあり，前者の方が微係数の次数も低く直交直線座標系における構成方程式がそのまま変換されるので容易である．ただし，微分方程式を求めるためには Green 積分を必要とする．微分方程式法はベクトル的な配慮が必要で，一般に複雑で誤りを犯しやすい．非直交曲

線座標系は，任意形状の境界についても容易につくることができるが，変換された式は前者よりも複雑である．なお，このテーマについては技報8号に述べてある．この方法は x-y 座標系で記述された微分方程式を解析平面 ξ-η 座標系に変換するもので，変数ならびに式がベクトルの場合でもその変換は行わない．

2. 任意節点配置差分法

ここでは主として技報第0，4，7号に述べられている内容について概説する．前述のように微分方程式は無限個の点に関する連続的な表現であるから，これを有限個の自由度をもつ代数方程式の形に置き換えることが必要である．そのためには，無限個の点の中から有限個の点を選出し代表させることが必要で，このようにして選ばれた点を節点とよぶことにする．直交直線格子系では格子点を置いたが任意節点配置法では，まったく自由に節点の配置を行うことができる．しかしながら，まったく無法則に節点配置を行うとき，その配置濃度にむらができるおそれがあり，さらに致命的なことは微係数の差分表示が不可能となることである．これを避けるためには領域内に境界に適合する曲線格子系をつくり，その格子点に節点を置けばよい．この格子系は非常に精度よくつくられる必要はなく，手書きの曲線程度で十分である．節点は変数の情報を代表する点で，その代表度により0，1，2次節点とよび，これらはそれぞれ0，1，2次の変数微係数値までを代表する点である．有限要素法では高次の節点まで用いられることがあるが，差分法ではすべての0次節点すなわち変数値のみを代表する節点とする．したがって，高次の微係数値は差分評価節点のみでは求めることが不可能で，その節点の近くにある数個の節点の協力をえて初めて可能となる．このようにして集められた節点の1群を差分要素または差分家族と名付ける．差分家族は差分評価の精度上なるべく近くにあることが望ましく，その構成要員の数は微分方程式の次元，階数によって定められる．たとえば2次元，2階の微分方程式の場合は，差分評価節点を含み，最低6個である．

つぎに差分家族ごとに差分評価を行う手順について考察する．ここでは，たとえば2次元 x, y 平面内変数 $u(x, y)$ に関する2階の微分方程式の差分化について考察する．簡単のため差分評価節点1は原点 $x=0$, $y=0$ に存在するものとし，その近傍における変数の分布は，

$$u(x,y)=\sum_{1}^{6} a_n f_n(x,y) \tag{2-1}$$

によって与えられるものとする．ここに，

$$a_1=u,\quad a_2=u_x,\quad a_3=u_y,\quad a_4=u_{xx},\quad a_5=u_{xy},\quad a_6=u_{yy} \tag{2-2}$$

$$f_1=1,\quad f_2=x,\quad f_3=y,\quad f_4=x^2/2,\quad f_5=xy,\quad f_6=y^2/2 \tag{2-3}$$

である．節点 m において次式が満たされる．

$$\sum_{1}^{6} f_{mn}a_n=u_m,\quad f_{mn}=f_n(x_m,y_m),\quad u_m=u(x_m,y_m) \tag{2-4}$$

上式は未知数 a_n に関する連立方程式であるから，これを解いて

$$a_m=\sum_{1}^{6} g_{mn}u_n \tag{2-5}$$

を得る．ただし g_{mn} は f_{mn} の逆行列である．このとき行列 f_{mn} が特異ならば，逆行列 g_{mn} は存在しない．したがって，差分評価は不可能となる．このように f_{mn} が特異となるのは6個の節点のすべてが一つの2次曲線または2本の直線上にのる場合に起こる．いま，ある2次式を

$$\sum_{1}^{6} a_n f_n(x,y)=0 \tag{2-6}$$

とし，すべての節点がこの上にのるならば，

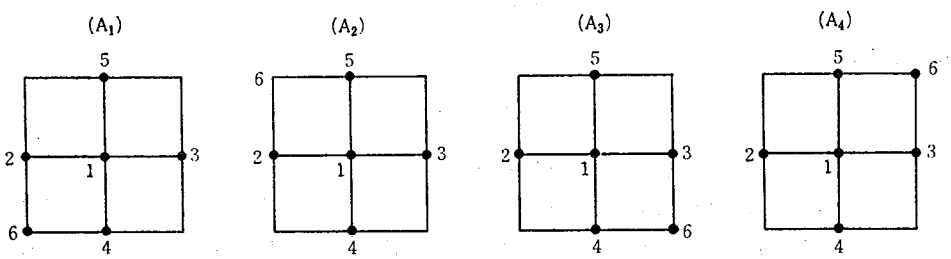
$$\sum_{1}^{6} f_{mn}a_n=0,\ (m=1\sim6) \tag{2-7}$$

が満たされる．したがって f_{mn} は特異である．

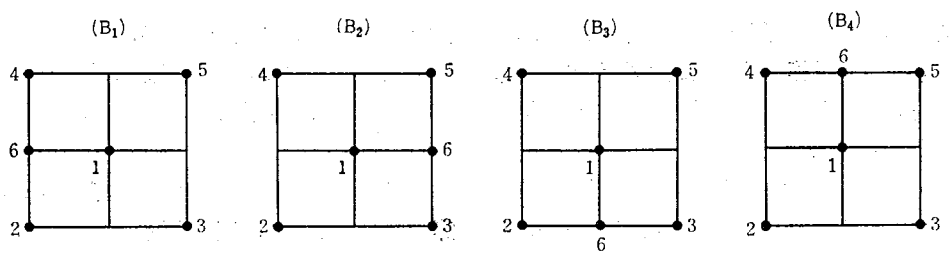
一般に6個の節点の位置が与えられたとき，その配置が正則か否かを判定することは簡単ではない．すなわち行列 f_{mn} を求め，

$$\det f_{mn}=0 \tag{2-8}$$

が満たされているか否かを，すべての差分家族について検討することはかなり労力的である．しかし，節点が曲線格子系の格子点に置かれていれば，微視的には直線格子とみなされるので，6個の節点のすべてが2本の直線上にないことを確かめれば十分である．その代表的な節点配置として十字形とX字形がある．（図1（技報第0号の図2））



（a） 十字形格子点配置



（b） X字形格子点配置

図 1 格子点配置 （技報，第0号より）

Fig. 1 Difference families of cross shape and rectangular shape

上に述べたことは差分評価節点を原点に置いた場合であるが，節点 i に置いて差分評価を行う場合は同点を原点とする i 局所座標

$$x^i=x-x_i,\quad y^i=y-y_i \tag{2-9}$$

について前述と同様のことを行えばよい．いま i 差分家族を

$$1^i=i,\ 2^i=j,\ 3^i=k,\ 4^i=l,\ 5^i=m,\ 6^i=n \tag{2-10}$$

とするとき，

$$x_1{}^i=0,\ x_2{}^i=x_j-x_i,\ x_3{}^i=x_k-x_i,\ x_4{}^i=x_l-x_i,\ x_5{}^i=x_m-x_i,\ x_6{}^i=x_n-x_i \tag{2-11}$$

である．y^i についても同様，ここで頭符 i は差分家族番号，1, 2, …は家族内節点番号を表す．

いま節点 i の近くで変数 $u(x,y)$ の分布が，

$$u(x,y)=\sum_{1}^{6} a_n{}^i f_n(x^i,y^i) \tag{2-12}$$

で表されるものとすれば，節点 m^i において次式

$$\sum_{1}^{6} f_{mn}{}^i a_n{}^i = u_m{}^i,\quad f_{mn}{}^i = f_n(x_m{}^i, y_m{}^i),\quad u_m{}^i = u(x_m{}^i, y_m{}^i) \tag{2-13}$$

が満たされる．もし，節点配置が正則ならば上式を $a_n{}^i$ について解いて，

$$a_m{}^i = \sum_{1}^{6} g_{mn}{}^i u_n{}^i \tag{2-14}$$

が得られる．ここに $g_{mn}{}^i$ は $f_{mn}{}^i$ の逆行列である．

つぎに，節点 i において満たされる微分方程式を

$$\sum_{1}^{6} A_m{}^i a_m{}^i = f_i \tag{2-15}$$

とする．上式に式(2-14)を代入し，

$$\sum_{1}^{6} A_m{}^i \sum_{1}^{6} g_{mn}{}^i u_n{}^i = \sum_{1}^{6} k_n{}^i u_n{}^i = f_i \tag{2-16}$$

が得られる．ただし，

$$k_n{}^i = \sum_{1}^{6} A_m{}^i g_{mn}{}^i \tag{2-17}$$

である．これらは，さらに差分家族構成(2-10)によって

$$u_1{}^i = u_i,\quad u_2{}^i = u_j,\quad u_3{}^i = u_k,\quad u_4{}^i = u_l,\quad u_5{}^i = u_m,\quad u_6{}^i = u_n \tag{2-18}$$

$$k_1{}^i = k_{ii},\quad k_2{}^i = k_{ij},\quad k_3{}^i = k_{ik},\quad k_4{}^i = k_{il},\quad k_5{}^i = k_{im},\quad k_6{}^i = k_{in} \tag{2-19}$$

の置き換えができるので，直に全系差分方程式

$$k_{ij} u_j = f_i \tag{2-20}$$

が導れる．この計算手順は境界条件式，領域的微分方程式について同一である．

このような計算法は多変数2階微分方程式，高階微分方程式についても同様に採用することができる．ただし，高階微分方程式の場合は前述のようにダミー節点が必要である．

2.1 補間関数法

前述の Taylor 展開法では差分家族ごとに逆行列を求めることが必要で多少計算量が多くなることが懸念され，この困難を避けるためこの計算法が提案される．図2（技報第4号の図11）に示されるような差分評価節点1を原点とし，9個の構成員からなる差分家族について変数 $u(x,y)$ の分布が，

$$u(x,y)=\sum_{1}^{9} p_m(x,y) u_m$$

によって表されるものとする．ここに $p_m(x,y)$ は補間関数で，

$$p_m(x_n,y_n)=\begin{cases} 1 & (n=m) \\ 0 & (n\neq m) \end{cases} \tag{2-21}$$

を満たす関数である

補間関数の求め方には種々の方法があると思れるが，ここでは3点を通る2次曲線4本の積によって表すこととした．実際2次曲線 $L_m=0$ は差分家族内では直線に近い式である．補間曲線 $p_m(x,y)$ が定められれば原点における変数 $u(x,y)$ の微係数は，

$$u_x=\sum p_{m,x} u_m,\quad u_y=\sum p_{m,y} u_m,\quad u_{xx}=\sum p_{m,xx} u_m,$$

$$u_{xy}=\sum p_{m,xy} u_m,\quad u_{yy}=\sum p_{m,yy} \tag{2-22}$$

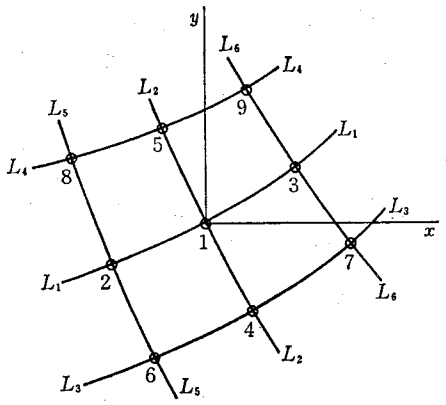


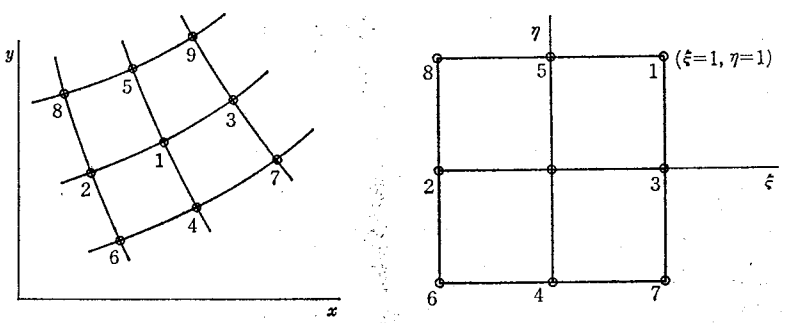
図 2　差分家族　（技報，第4号より）

Fig. 2　Difference family

によって与えられる．節点 i における微係数は Taylor 展開法と同時に，局所座標系 $x^i=x-x_i$, $y^i=y-y_i$ について同様の計算を行えばよい．

2.2　アイソパラメタ法

この計算法では，図3（技報第4号の図13）に示されるように物理 (x,y) 平面における9節点構成の差分家族をパラメタ (ξ,η) 平面における格子定数1の直交直線格子系差分家族に置き換え，変数 $u(x,y)$ も座標と同様の変換が行れることを仮定して微係数の差分評価を行うものである．



（a）物理平面　　（b）パラメタ平面

図 3　物理平面とパラメタ平面　（技報，第4号より）

Fig. 3　Physical plane and parameter plane

いま，座標 x,y は

$$x(\xi,\eta)=\sum_1^9 p_m(\xi,\eta)x_m,\quad y(\xi,\eta)=\sum_1^9 p_m(\xi,\eta)y_m \tag{2-23}$$

によって ξ,η 座標系に変換されるものとする．ここでは当然，

$$p_m(\xi_n,\eta_n)=\begin{cases}1 & (n=m)\\ 0 & (n\neq m)\end{cases} \tag{2-24}$$

が満たされる．変数も座標と同様に，

$$u(\xi,\eta)=\sum_1^9 p_m(\xi,\eta)u_m$$

により ξ,η の関数として定義され，その微係数 $u_\xi, u_\eta, u_{\xi\xi}, u_{\xi\eta}, u_{\eta\eta}$ が定められる．実際には x,y 座標による微係数 $u_x, u_y, u_{xx}, u_{xy}, u_{yy}$ が必要で，これらの値は変換式

$$a_m=\sum_1^9 c_{mn}b_n \tag{2-25}$$

によって与えられる．ただし $a_1=u, a_2=u_x, a_3=u_y, a_4=u_{xx}, a_5=u_{xy}, a_6=u_{yy}$，である．$b_1=u$, $b_2=u_\xi, b_3=u_\eta, b_4=u_{\xi\xi}, b_5=u_{\xi\eta}, b_6=u_{\eta\eta}$ で c_{mn} は，その変換行列で技報第4号82ページの式(3-9)～(3-29)に示されている．いま微分方程式を

$$\mathrm{L}(a)=\sum_1^6 A_m a_m=f \tag{2-26}$$

とするとき，式(2-25)によって

$$\sum_1^6 A_m\sum_1^6 c_{mn}b_n=\sum_1^6 B_n b_n=f,\quad B_n=\sum_1^6 A_m c_{mn} \tag{2-27}$$

解析平面における微分方程式に変換される．以上の計算は一つの差分家族について行ったものであるが実際は，すべての差分家族について同様の計算を行うことが必要である．

つぎに解析平面における変数ならびにその微係数値は，

$$b_m=\sum_1^9 d_{mn}u_n \qquad (m=1\sim6) \tag{2-28}$$

によって変換される．ただし，

$d_{11}=1$, $d_{22}=-1/2$, $d_{23}=1/2$, $d_{34}=-1/2$, $d_{35}=1/2$, $d_{41}=-2$, $d_{42}=1$, $d_{43}=1$
$d_{56}=1/4$, $d_{57}=d_{58}=-1/4$, $d_{59}=1/4$, $d_{61}=-1/2$, $d_{64}=1$, $d_{65}=1$,

である．さらに式(2-28)を式(2-27)に代入して

$$\sum_1^6 B_m b_m=\sum_1^6 B_m\sum_1^9 d_{mn}u_n=\sum_1^9 k_n u_n,\quad k_n=\sum_1^6 B_m d_{mn}$$

を得る．

3. 座標変換差分法

まえがきにおいて述べたように，座標変換差分法は直交格子系と非直交格子系によるものに大別され，さらに前者は構成方程式法と微分方程式法に分けられる．

3.1 直交曲線座標による構成方程式法

連続体の挙動は微分方程式によって記述されているが，その微視的な特性は構成方程式によって与えられている．連続体の微視的な状態は歪によって表され，これに作用する微視的な力は，たとえば弾性力学では応力とよばれている．これら二つの量は互いに独立ではなく一つの法則によって結ばれており，この関係を表す式を構成方程式とよんでいる．たとえば，一つのスカラ変数 $u(x,y)$ の微分は，

$$du=\varepsilon_x dx+\varepsilon_y dy,\quad \varepsilon_x=u_x,\quad \varepsilon_y=u_y \tag{3-1}$$

によって与えられ，$\varepsilon_x,\varepsilon_y$ は歪を表している．$\varepsilon_x,\varepsilon_y$ は1次の歪を表しているが，さらに0次の歪として $\varepsilon=u$ を加えることもある．つぎに歪に作用する量を σ_x,σ_y,σ とするとき，これらの量は構成方程式

$$\sigma_m=k_{mn}\varepsilon_n \tag{3-2}$$

によって結ばれている．ただし，$\varepsilon_1=\varepsilon_x, \varepsilon_2=\varepsilon_y, \varepsilon_3=\varepsilon, \sigma_1=\sigma_x, \sigma_2=\sigma_y, \sigma_3=\sigma$ とする．

つぎにベクトル変数を，

$$\boldsymbol{u}(x,y)=i_0u(x,y)+j_0v(x,y) \tag{3-3}$$

とするとき，その微視的状態は，

$$d\boldsymbol{u}=i_0(\varepsilon_{xx}dx+\varepsilon_{xy}dy)+j_0(\varepsilon_{yx}dx+\varepsilon_{yy}dy),$$

$$\varepsilon_{xx}=u_x,\quad \varepsilon_{xy}=u_y,\quad \varepsilon_{yx}=v_x,\quad \varepsilon_{yy}=v_y \tag{3-4}$$

によって与えられ，$\varepsilon_{xx}, \varepsilon_{xy}, \varepsilon_{yx}, \varepsilon_{yy}$ は2次の歪を表す．つぎに，これに作用する応力を σ_{xx}, σ_{xy}, σ_{yx}, σ_{yy} とするとき，これらの量は構成方程式によって関係づけられている．なお対称な応力 $\sigma_{xy}=\sigma_{yx}$ のときは，

$$\varepsilon_1=\varepsilon_{xx}=u_x,\quad \varepsilon_2=\varepsilon_{xy}=u_y+v_x,\quad \varepsilon_3=\varepsilon_{yy}=v_y,\quad \sigma_1=\sigma_{xx},\quad \sigma_2=\sigma_{xy},\quad \sigma_3=\sigma_{yy} \tag{3-5}$$

として構成方程式を次式のように表す．

$$\sigma_m=k_{mn}\varepsilon_n \tag{3-6}$$

ここに i_0, j_0 は x, y 方向の単位ベクトルを表す．さらに，支配方程式（境界条件式ならびに領域内微分方程式）は下記 Green 積分によって導れる．

（1） スカラ変数

$$\delta U=\iint\sigma_m\delta\varepsilon_m dxdy=\iint\left(\sigma_x\frac{\partial\delta u}{\partial x}+\sigma_y\frac{\partial\delta u}{\partial y}+\sigma\delta u\right)dxdy$$

$$=\int_s F\delta u ds-\iint_s L\delta u dxdy \tag{3-7}$$

$$F=l_x\sigma_x+l_y\sigma_y \tag{3-8}$$

$$L=\frac{\partial\sigma_x}{\partial x}+\frac{\partial\sigma_y}{\partial y}-\sigma \tag{3-9}$$

（2） ベクトル変数

$$\delta U=\iint\sigma_m\delta\varepsilon_m dxdy=\iint\left\{\sigma_{xx}+\frac{\partial\delta u}{\partial x}+\sigma_{xy}\left(\frac{\partial\delta u}{\partial y}+\frac{\partial\delta v}{\partial x}\right)+\sigma_{yy}\frac{\partial\delta v}{\partial y}\right\}dxdy$$

$$=\int_s(F_x\delta u+F_y\delta v)ds-\iint_s(L_x\delta u+L_y\delta v)dxdy \tag{3-10}$$

$$F_x=l_x\sigma_{xx}+l_y\sigma_{xy},\quad F_y=l_x\sigma_{xy}+l_y\sigma_{yy} \tag{3-11}$$

$$L_x=\frac{\partial\sigma_{xx}}{\partial x}+\frac{\partial\sigma_{xy}}{\partial y},\quad L_y=\frac{\partial\sigma_{xy}}{\partial x}+\frac{\partial\sigma_{yy}}{\partial y} \tag{3-12}$$

以上は x, y 座標系による数式化の手順であるが，つぎに直交曲線座標系 ξ, η について考察する．そのためには，まず歪関数を定義することが必要である．x, y 座標系では歪を評価するための微小長さは dx, dy であるが，直交曲線座標系でも微視的には直交直線座標とみなすことができるので，その標準微小長は $ds=g_1d\xi$, $dt=g_2d\eta$ とすることができる．ここに ξ, η は直交曲線座標，g_1, g_2 はその測度係数である．一般に

$$x=x(\xi,\eta),\quad y=y(\xi,\eta) \tag{3-13}$$

が与えられているとき，位置ベクトル $\boldsymbol{r}$ ならびにその微分

$$\boldsymbol{r}=i_0x(\xi,\eta)+j_0y(\xi,\eta) \tag{3-14}$$

$$d\boldsymbol{r}=ids+jdt,\quad ds=g_1d\xi,\quad dt=g_2d\eta,\quad g_1=\sqrt{x_\xi{}^2+\eta_\xi{}^2},$$

$$g_2=\sqrt{x_\eta{}^2+\eta_\eta{}^2},\quad i=(i_0x_\xi+j_0\eta_\xi)/g_1,\quad j=(i_0x_\eta+j_0\eta_\eta)/g_2 \tag{3-15}$$

によって測度係数ならびに ξ, η 方向単位ベクトル i, j が定められる．

このようにして歪関数が定義されれば，構成方程式も微視的に x, y 座標系と同じであるから応力関数も直に求められ，Green 積分によって支配方程式が導入される．

① スカラ変数，2階微分方程式

歪関数 $\varepsilon_1=\partial u/g_1\partial\xi$, $\varepsilon_2=\partial u/g_2\partial\eta$, $\varepsilon_3=u$ (3-16)

応力関数は式(3-2)によって定められる.

Green 積分 $$\delta U=\iint\sigma_m\delta\varepsilon_m dsdt=\int_l F\delta u dl-\iint_s L\delta u dsdt \tag{3-17}$$

微分式 $$L=\frac{\partial\sigma_1}{g_1\partial\xi}+\frac{\partial\sigma_2}{g_2\partial\eta}+\frac{\sigma_1}{\rho_2}-\frac{\sigma_2}{\rho_1}-\sigma \quad (\rho_1,\rho_2 \text{ は曲線 } \xi,\eta \text{ の曲率半径}) \tag{3-18}$$

② ベクトル変数, 2階微分方程式 $\boldsymbol{u}=iu+jv$

歪関数 $$\varepsilon_{11}=\frac{\partial u}{g_1\partial\xi}-\frac{v}{\rho_1},\quad \varepsilon_{12}=\frac{\partial u}{g_2\partial\eta}+\frac{\partial v}{g_1\partial\xi}+\frac{u}{\rho_1}-\frac{v}{\rho_2},\quad \varepsilon_{22}=\frac{\partial v}{g_2\partial\eta}+\frac{u}{\rho_2} \tag{3-19}$$

Green 積分 $$\delta U=\int_l(F_1\delta u+F_2\delta v)dl-\iint_s(L_1\delta u+L_2\delta v)dsdt \tag{3-20}$$

微分式 $$L_1=\frac{\partial\sigma_{11}}{g_1\partial\xi}+\frac{\partial\sigma_{12}}{g_2\partial\eta}-2\frac{\sigma_{12}}{\rho_1}+\frac{\sigma_{11}-\sigma_{22}}{\rho_2},$$

$$L_2=\frac{\partial\sigma_{12}}{g_1\partial\xi}+\frac{\partial\sigma_{22}}{g_2\partial\eta}+\frac{\sigma_{11}-\sigma_{22}}{\rho_1}+2\frac{\sigma_{12}}{\rho_2} \tag{3-21}$$

これらの微分方程式は解析平面における直交直線格子系により差分化される.

3.2 非直交曲線座標系による微分方程式法

構成方程式は一般に直交座標系に関し記述されているので, 非直交の場合はこの方法を用いることは不適当である. いま境界に適合するように曲線座標系 $\xi(x,y)$, $\eta(x,y)$ が定義されているものとする. この曲線は一般に非直交で, その x,y に関する微係数

$$\xi_x,\xi_y,\xi_{xx},\xi_{xy},\xi_{yy},\cdots,\eta_x,\eta_y,\eta_{xx},\eta_{xy},\eta_{yy},\cdots \tag{3-22}$$

も定められる. つぎに変数 $u(x,y)$ は微分方程式(3-23)

$$A_m a_m=f \tag{3-23}$$

を満たすものとする. ここに

$$a_1=u,\quad a_2=u_x,\quad a_3=u_y,\quad a_4=u_{xx},\quad a_5=u_{xy},\quad a_6=u_{yy},\quad \cdots \tag{3-24}$$

で A_m はその係数である. 同様にして ξ,η に関する微係数を

$$b_1=u,\quad b_2=u_\xi,\quad b_3=u_\eta,\quad b_4=u_{\xi\xi},\quad b_5=u_{\xi\eta},\quad b_6=u_{\eta\eta},\quad \cdots$$

とするとき, a_m は

$$a_m=c_{mn}b_m \tag{3-25}$$

によって変換される. ここで変換行列 c_{mn} は式(3-22)の関数として, 次のように与えられる.

$$c_{11}1,\quad c_{22}=\xi_x,\quad c_{23}=\eta_x,\quad c_{32}=\xi_y,\quad c_{33}=\eta_y,\quad c_{42}=\xi_{xx},\quad c_{43}=\eta_{xx},\quad c_{44}=\xi_x{}^2,$$
$$c_{45}=2\xi_x\eta_x,\quad c_{46}=\eta_x{}^2,\quad c_{52}=x_y,\quad c_{53}=\eta_{xy},\quad c_{54}=\xi_x\xi_y,\quad c_{55}=\xi_x\eta_y+\xi_y\eta_x,$$
$$c_{56}=\eta_x\eta_y,\quad c_{62}=\xi_{yy},\quad c_{63}=\eta_{yy},\quad c_{64}=\xi_y{}^2,\quad c_{65}=2\xi_y\eta_y,\quad c_{66}=\eta_y{}^2,\quad \cdots \tag{3-26}$$

つぎに, 式(3-25)を式(3-23)に代入し

$$A_m a_n=A_m c_{mn}b_n=B_n b_n=f,\quad B_n=A_m c_{mn} \tag{3-27}$$

をうる. 上式は解析 $(\xi\text{-}\eta)$ 平面における微分方程式である. いま ξ,η を整数とするとき, 物理 $(x\text{-}y)$ 平面における $\xi\text{-}\eta$ 曲線は解析平面では格子定数1の直交直線格子系に置き換えられる. したがって, 微分方程式(3-27)は直に差分化される.

以上は四つの境界によって囲まれた単純な領域について述べたものであるが, さらに複雑な形の領域でも適当に分割することによって4境界領域の集りに置き換えることができる. さらに多変数微分方程式の場合も変数ならびに式がスカラ, ベクトルにかかわりなく, 単に微係数の座標変換のみで差分化することができる. このことは前述の 3.1 節とは異なるものである.

4. 高階微分方程式の解析法

4, 6, 8階等，高階の微分方程式は複数個の境界条件式をもつので，その差分化に際しても特別の考慮が必要である．たとえば変数，$u(x, y)$ に関する4階微分方程式は下記2次の歪関数

$$\varepsilon_1=u_{xx},\quad \varepsilon_2=2u_{xy},\quad \varepsilon_3=u_{yy},\quad \varepsilon_4=u_x,\quad \varepsilon_5=u_y,\quad \varepsilon_6=u \tag{4-1}$$

これに対応する作用関数を $M_1, M_2, \cdots, M_6$ とするとき Green 積分

$$\delta U=\iint M_m\delta\varepsilon_m dxdy=\int_s (F_x\delta u_x+F_y\delta u+F\delta u)ds+\iint L\delta u dxdy \tag{4-2}$$

によって導入される．上の積分式にみられるように変分の対象となるパラメタは領域内では1個，境界上では3個である．したがって，微分方程式も領域内では1個，境界上では3個存在しうる．いま境界 s の内方法線を t とするとき，

$$F_x\delta u_x+F_y\delta u_y=F_s\delta u_s+F_t\delta u_t \tag{4-3}$$

が満たされる．したがって，式(4-2)の境界積分において

$$\int_s F_s\delta u_s ds=\int_s Fs\frac{\partial\delta u}{\partial s}ds=-\int_s F_{s,s}\delta u ds \tag{4-4}$$

の置き換えを行うことによって，

$$\int_s (F_x\delta u_x+F_y\delta u_y+F\delta u)ds=\int_s (F_t\delta u_t+F^*\delta u)ds,\ F^*=F-F_{s,s} \tag{4-5}$$

に変形することができる．したがって，パラメタは u と u_t の2個となり，境界条件式も2個に低減される．この2個の条件式を差分化するためには1個のダミー節点を境界外に設置することが必要である．同様にして6階，8階の微分方程式ではそれぞれ2，3個のダミー節点を必要とする．

5. おわりに

以上で新しい差分法に関する技報第0, 4, 5, 7, 8号の内容についてその概要を述べた．ここでは主として線形微分方程式を対象とした議論であるが，実際には Navier-Stokes 方程式のように非線形項が重要な役割を行う場合があり，とくにその数値計算法が問題となるので，今後の研究が必要である．非直交曲線格子系による座標変換差分法については引き続き報告する予定である．

参考文献 [1] 藤野 勉，渡部義維，“新しい差分法の提案と数値実験”，技報，日本ユニバック(株)，No. 0, FEB. 1981, pp. 63-78.
[2] 藤野 勉，“任意節点配置差分法による連続体解析”，技報，日本ユニバック(株)，No. 4, FEB. 1983, pp. 1-19.
[3] 藤野 勉，“座標変換差分法による連続体解析”，技報，日本ユニバック(株)，No. 5, AUG. 1983, pp. 82-96.
[4] 藤野 勉，“高階微分方程式の解析法”，技報，日本ユニバック(株)，No. 7, AUG. 1984, pp. 1-19.
[5] 藤野 勉，“非直交曲線格子系による座標変換差分法” 技報，日本ユニバック(株)，No. 8, FEB. 1985, pp. 60-70.

ビデオテックス
Videotex

當麻悦三

1. サービス，テストの推移

ビデオテックスの発祥の地は英国，創案者は Sam Fedida 氏といわれる．当時の British Post Office (現在のBritish Telecommunications) はこのアイデアを育て，1974年には Viewdata という名の雛型を実際に動かしてみせるに至った．1979年に始められた商用の Prestel は Viewdata が発展したものである．

Prestel は，西ドイツ，スイス，スウェーデン，スペイン，オランダ，フィンランド，デンマークに導入されている．フランスは1981年に Teletel の実験サービスを始めた．カナダでは1978年に Telidon の開発を終り引き続いて実験とシステムの拡大に取り組んでいる．日本はさらに遅く，1979年末に Captain (Character And Pattern Telephone Access Information Network System) を開発した．米国においては，これまでのところ動きが緩慢であったが，1984年に至り Videotex (CBS 放送，ATT) が実験開始となっている．

2. 利用対象としてのビデオテックス

ビデオテックスはニューメディア，すなわち新たな通信の媒体の一つとして脚光を浴びているが，その特質は，①文字による情報，図形による情報を広く伝達するシステムであること．②電気通信を専ら手段とすること，③低価格のディスプレイ端末装置を使うこと，④双方向の通信が行えること，⑤情報を選ぶ制御は受信者が行うこと，⑥制御手続きは特に訓練を受けていない人でもすぐに会得できること，にある．

ビデオテックスの登場は，新聞，テレビ，ラジオ等の情報通信媒体にさらに一つの公共の大量通信の手段をつけ加えることとなる．とくに，情報伝達が双方向に行われることが，従来の媒体にない属性で，その利用に新局面をもたらす鍵の一つである．ビデオテックスを用いて私的な通信もできる．公共に提供されるビデオテックスを会員制とでもいうべき様態 (CUG, Closed User Group) で使う，またはシステム全体を私有し用いればよい．すでにパーソナル・コンピュータにも，ビデオテックス用機能を取り付けられるものが出はじめている．

3. ビデオテックスの表現するシンボル

ラジオは音を表現し，テレビジョンは音と画を表現する．ビデオテックスも音と画の表現を行うが，その表現内容は総じて精密である一方，動きは単純である．表現の対象を列挙すると Captain の場合，次のとおりである．

①アルファベット，数字，平仮名，片仮名，漢字や記号等（テキスト情報），②点による絵（フォトグラフィック図形情報）③四辺形等の約150種の素片を組合せて表す絵（モザイク図形情報，フォトグラフィック図形より粗い絵となるが，情報量の節約ができる），④線分，円弧等で表す輪郭図(ジオメトリック図形情報)，⑤臨時に用いる絵文字等 (DRCS 情報，Dynamically Redefinable Character Set)，⑥フォトグラフィック図形のスクリーン上の移動(簡易動画コマンド情報)，⑦音の高さ・長さ・音色等（メロディ情報），⑧ソフトウェア（テレソフトウェア情報等)．

このような表現の枠組みは，国際標準化機構の国際標準「開放型システム間相互接続の基本参照モデル」によりプレゼンテーション層に位置づけられていると同時に，その内容は別の国際標準として審議中である．この分野で影響の大きなものとしては欧州の CEPT (Conference of European Posts and Telecommunications) の定めるものと，北米の NAPLPS (North American Presentation-Level-Protocol Syntax) がある．

4. ビデオテックスの利用分野と影響

適用分野としてビデオテックスに期待しうるものには，天気予報情報やスポーツの試合結果等を知るためなどといった情報検索をはじめとして，トランザクション処理，メッセージ伝達，コンピュータ処理，が考えられる．とくに時とともに刻々変化する情報，たとえば株価や為替相場等を個別の需要に応じて提供することはビデオテックスの特徴を発揮するものであろう．入金，支払い，買物が在宅のまま可能となること，ゲームやクイズを楽しむこと，通信教育を会話的に行うこと，手紙や葉書のかわりに使って通信文を伝えること，電話帳として使うこと，百科事典として使うこと等，ビデオテックスの利用対象分野は大きな広がりを潜在させている．

この利便の反面，今の社会環境との懸隔（ギャッ

表 1 世界の主要ビデオテックス
Table 1 Major videotex systems in the world
なお，表中のキャプションは，1948年11月30日に商用開始，米国のビデオテックス（CBS 放送，ATT）は1984年に実験を開始している．

国名	システム名	機関名	実験開始	端末数	蓄積情報量	備考
日本	キャプテン	郵政省・電電公社	1979年12月	2000	画面19万2500	1981年8月より第Ⅱ期実験
英国	プレステル	電気通信公社	1978年6月(79.9.商用化)	1万8675	23万	商用サービス中
	国際プレステル	電気通信公社	1980年3月(81.7.商用化)	326	2万	国際ビジネス用，オーストラリア，アメリカなど7カ国にサービス中
フランス	テレテル	郵電庁	1981年6月	3000	2万5000(除く外部センター分)	実験期間18カ月
	電子電話帳	郵電庁	1981年4月			1982年に30万端末設置，10年計画で全国普及化
西ドイツ	ビルトシルムテキスト	郵電省	1980年6月	7000	14万(除く外部センター分)	外部センター15の利用可
カナダ	アイーバ	マニトバ電話会社	1980年6月	33		南ヘディングリー マニトバ州 (同軸ケーブル)
	マーキュリー	ニューブランズウイック電話会社	1981年4月	45		セントジョン ニューブランズウイック州
	ビスタ	ベルカナダ	1981年5月	500		トロント，モントリオール，ケベック
	グラスルーツ	マニトバ電話会社	1981年4月	25		ウイニペグ マニトバ州 農業情報
	AGT/テリドン	アルバータ州政府電話会社	1981年7月	30		カルガリー アルバータ州 (同軸ケーブル)
	ノバテックス	テレグローブカナダ	1982年3月	500		国際データベース 衛星で在外大使館を結ぶ（公館，大使館）13対地
	ビデオテックス	ブリティッシュ・コロンビア電話会社	1981年	150		バンクーバー ブリティッシュコロンビア州
	イライ	マニトバ電話会社	1981年11月	150		イライ マニトバ州 (光ファイバ)
	テリドンⅡ	テレケーブル・ヴィオトロン社	1982年初	250		モントリオール (同軸ケーブル)
米国	グリーンサム	農務省	1980年12月	200		ケンタッキー州 農業情報
	ビュートロン	ナイトリッダー新聞社 ATT	1980年7月	200		マイアミ郊外
	タイムズミラー	タイムズミラー新聞社	1981年	200		ロスアンゼルス
	チャンネル2000	バンクワン銀行 OCLC	1980年10月	200		コロンバス（オハイオ州） 情報検索 バンキング
	シティーバンク	シティーバンク	1980年12月	100		ニューヨーク バンキング
	ケミカルバンク	ケミカル銀行	1981年1月	200		ニューヨーク バンキング
	エクスプレス・インフォメーション	ユナイテッド・アメリカン銀行	1980年12月	400		ノックスビル（テネシー州） バンキング
	ビデオテックス	CBS放送 ATT	1982年(予定)	200		リッジウッド（ニュージャージー州） 家庭情報
オランダ	ビディテル	郵電総局	1980年8月	4000	14万	1983年から商用化
スイス	ビデオテックス	郵電庁	1980年7月	120	6万	1983年2000端末予定
フィンランド	テルセット	ヘルシンキ電話会社	1978年6月(80.4.商用化)	500	2万	商用サービス中
スウェーデン	データビジョン	電気通信庁	1979年3月	40	1万	1982年から商用化
スペイン	フネスコ	スペイン電話会社	1980年2月	200		インハウスの実験
ベルギー		電信電話公社	1981年7月	600	10万	家庭50% 事業所50%
ノルウェー	テレデータ	電気通信庁	1980年1月	100	1万	プレステル方式

プ）や危険がある．標準化はどんな対象についてどんな時機に行うべきか，情報の版権をどうするか，知る権利とプライバシー保護との相克等である．ビデオテックスは個人だけではなく社会の仕組みにも影響を与える．個人への影響は，プライバシー保護，安全保護，アクセス権，消費者保護といった利用者としての個人にかかわるものである．個人用の新聞の制作・教程の設定・買物案内の作成といった利便があると同時に，政治的または経済的目的による情報操作を受ける危険がある．家庭に入る情報が増えるだけでなく，家庭から意識的，あるいは無意識的に出力される情報も増えて第三者が個人の行動の傾向，好みの傾向，消費性向等を当の本人以上に知ることも十分考えられるのである．

表 2 米国における通信媒体成長の足どり (Sterling and Haight, 1978 [1] から引用)
Table 2 Historical growth of media in the United States

通信媒体	発明の年	商用開始の年	下記普及率（米国）に達するまでの所要年数						
			1%	5%	10%	25%	50%	75%	90%
電話	1876	1877	20	—	—	—	69	80	93
中波ラジオ	1895	1920	3	4	5	8	11	18	27
テレビ	1927	1946	3	4	5	6	8	11	16
ケーブルTV	1949	1949	11	19	23	32	—	—	—
カラーTV		1955	7	10	12	14	17	22	—
ペイTV		1974	2	4	6	—	—	—	—
ビデオカセット		1978	2	4	—	—	—	—	—

ビデオテックスの情報伝達能力を使って，業務として情報伝達を行うことは誰にでも参入可能なことである．利用者にとっては情報源の多様性は歓迎すべきことであるが，これを既存の出版・新聞・放送・小売・宣伝等の大企業が寡占して規模の経済の実現を図る見込みがやはり大きいようである．また，利用の態様に応じてはばらばらの法的規制，実態のわかりにくい規制ということも考えられる．

つぎに社会の仕組みへの影響として，産業構造や雇用・国際貿易・通信等の変化が考えられる．たとえば，従来は別々であった二つの産業がビデオテックスによって境界がはっきりしなくなるとか物流や金融において仲介が減少するということが考えられる．

5. ビデオテックスの将来

英国等の先行例によると，ビデオテックスの契約の大多数（80パーセント以上）は企業等の業務上の利用を目的とするものである．しかし，このことはビデオテックスが個人にとって興味の対象になりにくいことを意味しているわけではない．たとえば，企業の契約している端末装置からも個人を対象とする情報へのアクセスがかなり起こるという例が知られている．これは端末装置を契約していない個人の潜在的利用者が，勤め先等の企業設置の端末装置によりアクセスしているものと解釈できる．

また，英国の実験で最も成功したのは旅行情報を扱うもので，電車・飛行機の発着時刻，運賃ホテルの空室の有無・宿泊料等の情報を検索し提供するものである．これも利用する対象者は企業もあろうがやはり個人が主体となるはずのものである．他にも個人の利用が有望なものとして，入金・支払い等のトランザクション（商取引）処理などがある．このような個人対象の媒体としての潜在能力を秘めたビデオテックスではあるが，その個人に対する普及の速さとなると電話，ラジオ，テレビといった先行媒体に事例を求めることが妥当と思われる．

表2によると，(中波)ラジオとテレビは普及率が90パーセントになるまで，それぞれ27年，16年であるのに対して，電話では93年を要していることがわかる．電話とラジオ，テレビの普及速度を左右する要因としてはその媒体がどれだけ娯楽志向かということがきいていると思われるが，ビデオテックスの娯楽志向性はそれほど強いとは考えられない．

このように考えると，近い将来のビデオテックスは主として企業等で用いられると予測される．この場合，受け入れやすいと思われる利用形態は商品の説明，操作手順案内，教育訓練，職員名簿案内，メッセージ伝達，機器の無人運転に伴って必要となる機器の監視等であり，必要に応じ適宜 CUG (Closed User Group) を設定して，情報交換・提供の範囲に制限を加えた利用となるであろう．

参考文献

[1] J. Tydeman, H. Lipinski, R. Adler, M. Nyhan, L. Zwimpfer, *Teletext and Videotex in the United States*, McGraw-Hill.

[2] Videotex, *Computer Weekly*, June 9, 1983.

[3] 始まったニューメディア革命，電波新聞，1983, 8.14～11.8.

[4] いよいよ来たビデオテックス時代，ビジネスコミュニケーション，Vol.21, No.2,.

[5] 実用段階を迎えたキャプテンシステム，日本の科学と技術，Vol.24, No.219,.

[6] 明日の地域社会とテレコミュニケーション，郵政省，参考資料.

[7] NAPLPS:A New Standard for Text and Graphics", *BYTE*, Vol.8, No.2～5, BYTE Publications

[8] Preliminary Standard *T 500-1982*, Videotex/Teletext Presentation Level Protocol Syntax, Canadian Standards Association.

[9] Captain Presentation Level Protocol Syntax.
[10] *ISO/TC 97/SC 16 Subcommittee Report to TC 97 Plenary*, May 1984, N 1812 (TC 97 N 1358).

(通信システム開発部)

コンピュータ・アニメーション・システム ANTICS

福士祐治郎

1. はじめに

映像は，文字や音に比べて一度に多量の情報をビジブルに伝達できる．

1980年初頭，コンピュータによる映像化の技術は欧米では，すでに実用化されつつあったが，日本ユニバック(株)でも映像化技術の一分野コンピュータ，アニメーションのターン・キー・システムを開発することになった．

コンピュータ・アニメーションについて10年以上のもの間，研究を続け経験のある英国人 Mr. Alan Kitching の GPSA 社（英国）と1981年になって共同開発に踏み切った．

ANTICS ターン・キー・システムでは次の目標が設定された．

1) コンピュータの知識なしにアーティストが使用できるマン・マシン・インタフェースを確立すること
2) システムの価格および運用コストが低いこと
3) 出力映像はテレビ放送に使用できる品質であること
4) 生産性のよいこと

1)～4)の目標を達成するために中央演算装置として32ビット・スーパ・ミニコンピュータを，また映像出力媒体としては1インチ VTR (Video Tape Recorder) を使用することとし，1コマずつ VTR に自動録画できる装置，ビデオ・インタフェースの開発を放送業界の大手（株）東通と共同開発することとなった．

また，ANTICS のソフトウェアについては年々コスト・パフォーマンスのよくなるコンピュータの動向を考え，ポータビリティ（移植性）の高いものを目指した．

こうして1982年10月末にはバージョン1.0を(株)東通ビデオ・センタにリリーズし，1984年6月末にはバージョン2.0をリリーズし，現在では国内6社のユーザをもつに至っている．海外では，オランダとベルギーに各々1か所の ANTICS スタジオが開かれている．

2. アニメーションの制作過程

従来のアニメーションの制作過程のどの部分が ANTICS によりコンピュータ化されるのかを，図1で説明する．

1) アニメーション設計の段階

この段階ではアニメーション全体の構想をねり，登場人物および各場面（シーン）を設定し，各々のシーンにおいて登場人物をどのように演じさせるか具体的に時間軸にそって設定していく段階であり，コンピュータ・アニメーションでも従来のアニメーション手法でも同じ作業を要する．すなわち，この段階は制作者や監督が創造性を発揮する段階で，人

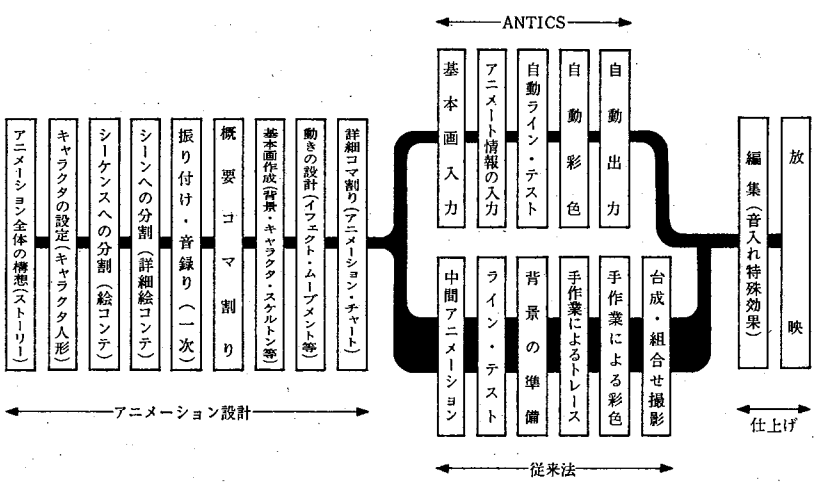


図1 従来のアニメーション制作過程と ANCITS システム
Fig. 1 Former Process of making animation film and ANTICS system

の感性に負う部分なので機械化はむずかしい.

2) 実作業段階

従来手法での，この段階は設計段階で作成された基本画，動きを時間軸にそって示した「アニメーション・チャート」に従って，多数の人々が基本画と基本画との間のコマ割り，そのコマ割り画の1枚1枚のトレース（セル画としてセルロイドの版に焼きつけるための作業），およびでき上がったセル画への彩色，これらの一連の作業とは別にかかれた場面（背景画）と先のセル画とを組み合わせ，これらの1コマ1コマをカメラで撮影する等の作業段階である．すなわち，アニメーション制作過程の中でもっとも労働集約的な作業である．

かりに，現在テレビで放映されている22分間のアニメーション・フィルム1本には，8(コマ/秒)×22(分)×60(秒)＝10,560(コマ)が，必要であり，それだけセル画が必要となる．しかし実際には，撮影の時のカメラによるZOOM，PANING等の効果によるシーンもあり，すべてセル画にしないでもよいため平均的には，約3,500枚程度のセル画を作れば済むようである．

それにしても，この3,500枚のセル画を1枚1枚手作業で作るのは大変な仕事である．

ANTICSシステムでは，基本画をタブレットから入力し，アニメーション・チャートに従って，その情報を入力すれば，コマ割り，彩色，1コマごとの撮影に当たるコマ録画をすべてコンピュータによって処理する．

3) 仕上げの段階

これは，2)において作成されたフィルムの編集(カットしたり，つなぎ合せたり)を行い，その後に声や音楽を入れて最終フィルムとする段階である．

ANTICS使用の場合は，フィルムでなくビデオ編集，ビデオへの音入れとなる．

3. ハードウェア構成

ANTICSシステムのハードウェア構成は図2に示すとおりである．バージョン2.0出荷時には，ビデオ・コントロールの部分が大幅に機能アップされており，この部分だけを，ビデオ・コントロール・サブシステム（VC-2）として商品化している（図2）.

3.1 ワークステーション

アーティスト（ANTICSオペレータ）が基本画を入力したり，アニメーション情報を入力したりする作業場がワークステーションであり，その場を形成するのは次の機器群である．

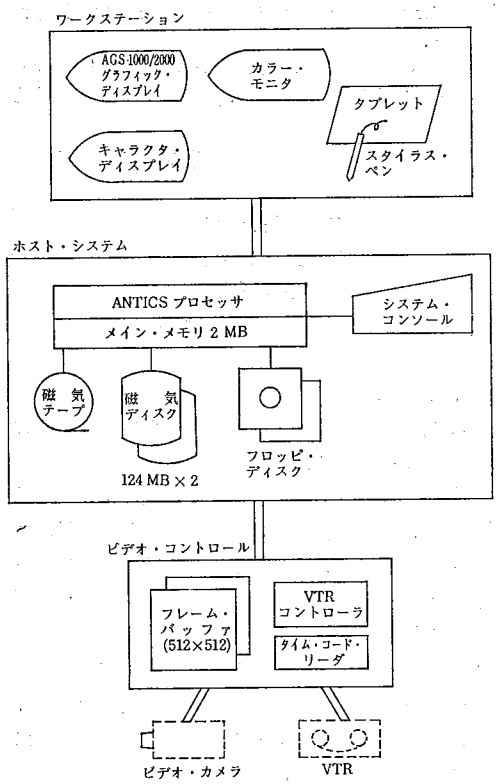


図 2 ANTICS のハードウェア構成
Fig. 2 Hardware configulation of ANTICS

1) タブレット/スタイラス・ペン

ANTICSシステムでの情報の入力は，すべてディジタイザ・タブレットに付いているスタイラス・ペンを通して行われる．タブレットには操作に必要なメニュがあり，メニュをスタイラス・ペンでヒットすることにより，ANTICSの操作モードが選択できるようになっている．メニュ内には英数字キー・ボード部分も組み込まれており，文字・数字の入力もその位置をヒットすることにより行われる．

アーティストは，キー・ボードの操作はにがてであり，普段，鉛筆・絵筆を使って仕事をしているので，スタイラス・ペンだけですべての操作ができるよう考慮してある．

このペンを使って基本画を描くことはもちろん，アニメーションの情報を与えること，録画の指示をすることもできる．

2) AGS 1000/2000 グラフィック・ディスプレイ

基本画入力時には，現在入力している絵の部分を線画として表示するものであり，アーティストはこの画面をみながら入力を進めていく．また，一度入力した基本画の部分修正，レイアウト，追加・削除等もこの画面を見ながら行う．

3) キャラクタ・ディスプレイ

アーティストはANTICSと会話をしながら操作するわけであるが，アーティストに対して次の操作を指示したり，選択メニュを表示する．また，アーティストがタブレットから入力した文字・数字の表示も行う．

4) カラー・モニタ

色づけされた基本画の表示，アニメーションにより変形された後の基本画をカラーで表示する．ここで使用されているカラー・モニタはNTSC (National Television System Committee) ビデオ信号，すなわち通常の家庭用 TV モニタと同様の信号で接続できる．

しかし，アニメーション映像の一つの重要なファクタはその色にあり，ここで使用しているカラー・モニタはプロフェショナルの仕様となっており，細かく色の調整ができるものである．このモニタは，VC-2（後述）と BNC 端子を通して同軸ケーブルにより接続される．

3.2 ホスト・システム

ホスト・システムの中核 ANTICS プロセッサは，32ビットのスーパ・ミニコンで，ANTICS ソフトウェアのローダブル・モジュール（約 20 MB）がメイン・メモリの容量を考慮せずに作動するようバーチュアル OS を採用している．

標準の ANTICS ターン・キー・システムではコスト・パフォーマンスを考慮して，メイン・メモリ 2 MB を実装している．

磁気ディスク装置は，456 MB の容量をもつウィンチェスタ・ディスクを1台採用している．また，システムのバック・アップ用としての磁気テープ・ユニットを1台必要とする．

3.3 ビデオ・コントロール

ビデオ・コントロール部分は，1984年6月に商品発表したビデオ・コントロール・サブシステム (VC-2) を中心に形成される．

1) フレーム・バッファ

ANTICS プロセッサで計算処理された画像データ (RGB のピクセル・データ各々8ビット 1600 万色同時発色) を蓄積し，DA 変換器およびエンコーダ・ユニットを通して，VTR で録画可能な NTSC コンポジット信号 (TV 信号) に変換する．

また，この逆に VTR 上の映像信号，あるいはビデオ・カメラから入力された映像信号をエンコーダ・ユニットおよび AD 変換器を通して画像データとして蓄積する．

ホスト・システムとフレーム・バッファ間は，16ビット・パラレル転送の高速インタフエース（最大0.5 MB/秒の転送レート）にて接続，READ/WRITE される．

また，VC-2 に内蔵されたシンク・ジェネレーション・ユニットは，ANTICS システム系の内部と外部のビデオ信号の同期をとることができるようになっている．

2) VTR コントローラ

VTR コントローラ部は，RS 232 C 回線を介してホストと接続され，ホストからの制御コマンドの指示により VTR をコントロールして，フレーム・バッファに蓄積された画像の自動録画および VTR，あるいはビデオ・カメラからの映像のフレーム・バッファへの自動入力を制御する．

ビデオ・テープ上のどの位置の映像を入力するのか，あるいはどの位置に録画するのかは，ビデオ・テープ上にあらかじめ記録されている SMPTE (Society of Motion Picture and Television Engineerings) タイム・コード hh : mm : ss : ff (時分，秒，フレーム) を指示して行う．

この位置を正確にポイントするタイミングをつかむことができるように，VC-2 内にはタイム・コード・リーダ・ユニットが内蔵されている．

4. 機　能

ANTICS システムには，アニメーションを作成していくのに必要な色々の機能が用意されている．

4.1 基本画の入力機能

基本画をはじめから入力することを原始入力という．ANTICS では，タブレット上のスタイラス・ペンを使用して線画で入力を進めていく．このとき，同時にその線の種類 (LINE, DOT, AREA) およびその線の色属性を指定する．

線の種類，LINE, DOT に関しては，太さ，大きさを1〜10段階で選択できるようになっている．AREA は塗りつぶしである．

次に色に関しては，カラー・モニタ上に表示されるカラー・パレット上を，カーソルでヒットすることによって選択する．色については不透明絵の具による採色 (opeque)，透明絵の具による彩色 (transparent)，ハイライト (luminous) の各々が選択できる．

4.1.1 原始入力

原始入力による3種類の方法が用意されている．

1) CAPTION

使用頻度の高いデザイン文字は，あらかじめ文字ファイルとしてディスクに登録されており，タブレ

ットからいちいち図形を入力する必要はない.

現在7種類の英数デザイン文字が使用でき，図3はFLASH BOLD, HELVETICA MEDIUMの例である.

また，漢字，カナ文字については，3種類（ナールD，ナールE，ゴチE）の常用漢字レベル（種類ごとに2240文字）を現在開発中である.

2) SHAPES

円・多角形・星形・扇形等の幾何学的図形および楕円・スーパ楕円・アルキメデスの渦線・LISSAJOUS等の関数曲線を正確に自動作成する機能である.

3) HAND DRAW

タブレットからスタイラス・ペンを使用して手書き入力する機能であり，通常，基本画入力の90パーセント以上はこの方法を使用する.

4.1.2 修正・追加・削除・編集機能

原始入力で入力された基本画の修正，追加，削除，あるいはレイアウトの変更を行うために，つぎに列挙する機能が用意されている.

ZOOM……図形の一部，あるいは全体を拡大・縮小する.

TURN……図形の一部，あるいは全体を回転する.

SHIFT……図形の一部，あるいは全体の位置の移動を行う.

RUBOUT……図形の一部，あるいは全体を削除する.

ALTER……図形の一部を変更する.

JOINTUP……図形と図形を結合して一つの図形とする.

SPLIT……図形を分割して複数の図形とする.

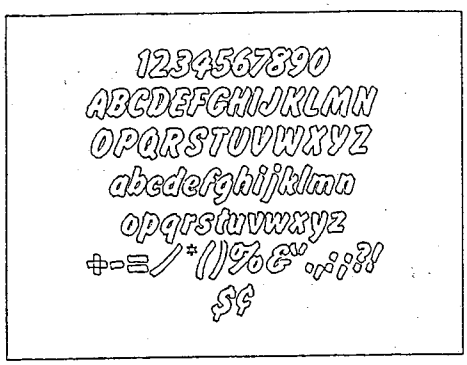
REPEAT……図形をコピーして複数個つくる.

4.2 ANIMATE 情報の入力

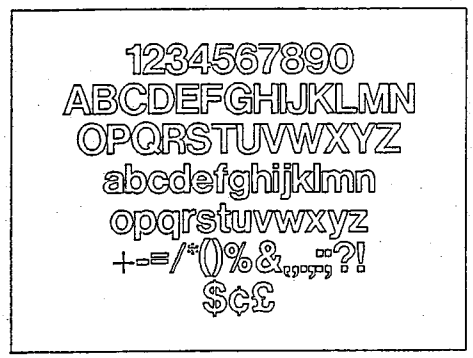
4.1節で基本画が入力されて採色のチェックも終ると，次はそれをどう動かすのかの指示，ANIMATE 情報の入力となる.

ANIMATE 情報は大きく分けて二つのタイプからなる．一つは，ANIMATION であり「動き」そのものを指定するものである．ある時間内にどう変化させるか（一般的に，初めの状態と終りの状態と時間をパラメタで表す）である.

他は MOVEMENT であり，「変化量」の制御に使用する．すなわち，同一の ANIMATION が与えてあっても，初めのうちゆっくり変化させ終りに近づいたら急激に変化させるのと，この逆とでは，できあがったアニメーションは大きく異なる.



(a) FLASH BOLD



(b) HELVETICA MEDIUM

図 3 CAPTION の文字の例

Fig. 3 Example of designed character of CAPTION

4.2.1 ANIMATION

ANIMATION には，3分類あり EFFECT（効果），INBETWEEN, SKELETON である（図4）．EFFECT は24種類あり，これらすべての ANIMATION を250個まで組み合せて使用することができるためアーティストの描いている動き，あるいは想像もつかないようなコンピュータまかせの動きまでを可能にしている.

1) EFFECT

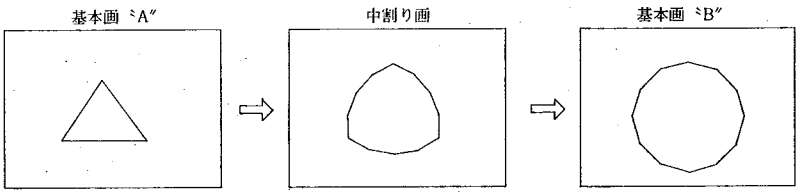
EFFECT の名称を表1に記す.

2) INBETWEEN

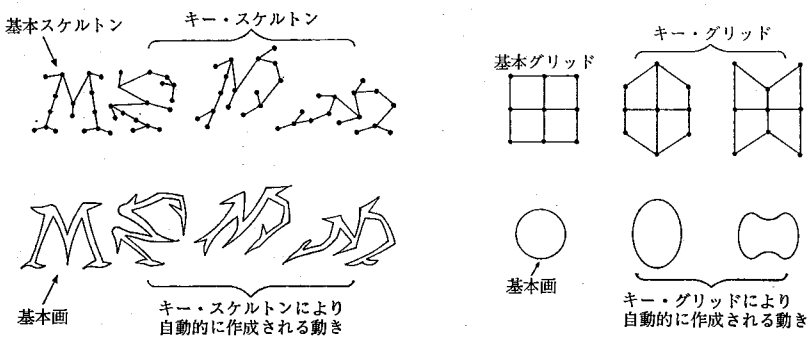
二つの基本画の間を線形補間することにより，メタモルフィックな変形効果をねらう，また，INBETWEEN は色についても線形的に変化させることができる.

特殊な機能は，INBETWEEN の 3D といって擬似的な3次元効果を与えることができる．すなわち，二つの基本画のうち，一つに平面図を入力し，もう一つに側面図を入力することにより，3次元凸多面体の基本的なものを回転・拡大・縮小・平行移動することができる[9].

3) SKELETON



(a) INBETWEEN



(b) SKELETON

図 4 INBETWEEN と SKELETON の例

Fig. 4 Examples of INBETWEEN and SKELETON function

表 1 EFFECT の名称[8]

Table 1 Name of EFFECT

HOLD (静止)	PATH (軌跡移動)	WOBBLE (伸縮1)
PAN (横移動)	FOLLOW (追随移動)	TUMBLE (巻きつけ)
TILT (縦移動)	FLIP (透視回転)	SCREW (うず巻移動)
ZOOM (拡大縮小)	PINCH (つまみ)	SPHERE (球体巻きつけ)
SPIN (平面回転)	WAVE (移動波)	TWIST (ねじり)
FADE (フェード)	BENDY (定常波)	SQUASH (伸縮2)
FREAK (フリーク)	LIMIT (部分表示)	CYCLE (繰返し)
MASK (マスキング)	MIRROR (鏡面)	LEVELS (奥行き制御)

基本画に SKELETON (骨組み) を与える. これを基本 SKELETON といい, この骨組みを動かすことにより基本画を変形させる. すなわち, コンピュータにより骨に対する肉付けを計算させるわけである.

また, SKELETON 機能の中の GRID というオプションは, 骨ではなく, GRID (格子) を基本画に対して与え, その GRID の変形をかくことにより, 基本画を変化させる (投影変換) 機能もある.

4.2.2 MOVEMENT

10種類の MOVEMENT があり, それら名称は STEADY, SMOOTH, DOUBLE, TAPER, SWING, RANDOM, KEY-FRAMES, TAGGED, SCALED, HAND-DRAWN である.

この名称の中のほとんどのものは, 横軸を時間とし縦軸を変化量とした関数が, あらかじめ ANTICS 内で設定されているものである. HAND-DRAWN だけは, 関数のかわりにタブレットからグラフを入力して MOVEMENT を決める方法であり, 人間の思考の中の ANALOG 量を入力する一つの方法であり, ANTICS のマン・マシン・インタフェースの優れた部分である.

4.3 彩 色

入力された基本画を彩色して, カラー・モニタに表示して確認する. この彩色は, ベクタ情報からラスタ情報 (ピクセルごとの RGB 値) への変換であり, ANTICS はこれをソフトウェアで行っており, 斜めの線を描いた時のギザギザ (aliasing) の除去, および採色, ハイライト色の計算もこの段階で行う.

展開されたラスタ情報は, VC-2 に転送されカラー・モニタに絵として表示される.

4.4 自動録画

色の確認・動きの確認が終了すると, VTR に1コマずつ録画することになる. ハードウェアの項で述べたように VC-2 の機能を使って録画開始のタイム・コードを指定し, 録画開始の指示をすれば, 後は正確に1コマずつ自動的に録画される.

こうして，ビデオ・テープに録画されたアニメーションは，そのまま放送局から放送されることもあり，ビデオ編集機を使用し，さまざまな編集効果を与えた後，音入れをすませて放送されることもある．

5. おわりに

動画化の応用として，シミュレーションの結果を正確な時間軸にそって映像化する．すなわち，アニメーション（動画）化してビジュアルにすることはよいプレゼンテーション・ツールとなるばかりでなく，マクロ的に現象を把握する強力な手助けになる．

また，CAD/CAM の分野においても，従来意匠デザインの部分が必ずしもコンピュータ化されてはいなかった．コンピュータ・アニメーションを利用した意匠デザインは，今後 CAD/CAM と連動して発展しよう．このようなコンピュータ処理における画像処理・映像化技術の応用分野はますます広がりをみせてきたが，これらのニーズを満たすべく ANTICS の改良を今後も進めたい．

参考文献

[1] W. M. Newman, R. F. Sproull, *Principles of Interactive Computer Graphics*, McGRAW-HILL, 1979.

[2] D. F. Rogers, J. A. Adams 著，山口富士夫訳，コンピュータ・グラフィックス，日刊工業新聞社，1980.

[3] 山口富士夫，コンピュータディスプレイによる図形処理工学，日刊工業新聞社，1981.

[4] テレビジョン学会編，テレビジョン画像工学ハンドブック，オーム社，1980.

[5] 日本放送協会編，高橋良監修，VTR 技術，日本放送出版協会，1982.

[6] 坂井滋和，“日本アニメーションが導入したアンティクス”，放送技術，1982，8.

[7] Hidemaru Sato, "Standardization of Animation Command for Computer Animation System", *Computer Graphics TOKYO '84 Proceedings*, 1984.

[8] ANTICS システム概説書，日本ユニバック情報システム，1982.

[9] ANTICS 操作解説書，日本ユニバック情報システム，1982.

（日本ユニバック情報システム
CG 事業部 ANTICS 推進室）

HAROLD ABELSON, GERALD J. SUSSMAN 他著

The MIT Electrical Engineering and Computer Science Series
"Structure and Interpretation of Computer Programs"

The MIT Press, X X+542 pp, 1984.

本書は，The MIT Press が1984年から上梓始めた The MIT Electrical Engineering and Computer Science Series の第1冊目に当たる．なお，同シリーズは，MIT の電気工学／コンピュータ・サイエンス学科 (department of electrical engineering and computer science) の教科書の公刊である．

著者の一人 G. J. Sussman は AI の専門家であり，学習システム HACKER や電子回路解析システム EL, AI 言語 (MICRO-PLANNER, CONNIVER, SCHEME) 等のインプリメンテーションや開発で知られる同学科の教授である．

一方の H. Abelson は，LOGO の実験や Turtle Geometry の著書で知られ，同学科の助教授としてコンピュータ・サイエンスの教育に情熱を傾けている人物である．

本書は，当初同学科のコンピュータ・サイエンスの入門コースの教材として作成されたもので，1978年から著述が始められ，1980年の秋までに本書と同じ内容のものが完成した．そして，それ以来毎年600～700人の学生の教育に使用されている．なお，これらの学生の大部分は，過去にコンピュータの教育を正式に受けていないとのことである．

さて，本書の狙いは，次の二つの主張の実現にある．第1にコンピュータ言語は，単にコンピュータを動かす手段として止まるわけでなく，人々が方法についてのアイデアを表現する新しい形式的なメディア (novel formal medium for expressing ideas about methodology) として捉えられるべきであること．つまり，プログラムを書く目的は，人間に読ませるためであり，コンピュータで実行させるというのは副次的なことである．

第2に，この段階で教えるべき内容は，特定のプログラミング言語の構文や特定の関数を効率よく計算するうまいアルゴリズムではなく，ましてやアルゴリズムの数学的分析や計算に関する基礎理論でもない．むしろ，大規模なソフトウェア・システムの知的複雑性を制御する技術 (the techniques used to control the intellectual complexity of large software systems) をテーマとすべきである．

さて，これらの主張は，構造的プログラミングのそれと類似しており，何ら新鮮味はない．

しかし，ただ2点を敷衍し，著者の主張の背景を明らかにしておきたい．

第一点は，第1の主張にかかわるものである．

著者は，コンピュータ・サイエンスを人間の思考方法および思考の表現を研究する新興の学問，言うならば procedural epistemology と名付けるべきものとして規定している．そして，その重要性は，コンピュータとほとんど無関係であると主張している．つまり，コンピュータ・サイエンスを，知識の構造を imperative な面から研究し，"how to" の概念を正確に取り扱うフレーム・ワークとして捉えており，"what is" の概念を正確に扱う，より declarative な性格をもつ古典的数学と対比している．

第二は，彼等の主張が LISP の世界では比較的耳新しいということである．

従来，LISP プログラマは，構造的プログラミング等に代表される主張（理解しやすく，より構造的なプログラムを作る）とは無縁であった．その理由は LISP が会話型言語であり，AI に代表される分野の prototyping に主として用いられてきたからである．

しかし，1970年代になって LISP で大規模かつ永続的に使用されるプログラムが書かれるようになると，問題が生じてきた．たとえば，1980 年の E. Charniak, C. K. Riesbeck, D. V. McDermott の "Artificial Intelligence Programming" (Lawrence Erlbaum Associates, 1980, 本年中に邦訳刊行予定) に，その例を見ることができる．同書には，McDermott と Sussman が Conniver の開発 (1972年) をしている際に，CDADR のような非ニーモニックなコードの多用によって保守不可能なプログラムを作ってしまったこと，およびその反省からマクロによるユーザ・データ型の概念を導入したことが述べられている．

そこで，従来の方法に対する反省からプログラムの可読性と変更の容易性を保証するために，FOR ループやユーザ・データ型を取り入れた新しいプログラミング技法が AI の分野でも模索され，いくつかの方法が提案された．

一方，LISP 自身にも改良が加えられ，read macro

やユーザ・データ型等の機能がLISPの標準的機能として含まれるようになった.

なお，本書で使用される言語は，1975年に G.L. Steele Jr. と Sussman によって開発された LISP 系言語 SCHEME である.

SCHEME は，static scoping を採用した tail-recursive interpreter であり，LISP のメタ記述能力と ALOGOL のブロック構造を取り入れた言語である．なお，最近 LISP の標準化案として関心を集めている COMMON LISP (Steele Jr. が中心となって仕様を定めた) にも大きな影響を与えている.

本書は，従来の LISP の教科書とは異なり，LISP 系言語によるプログラミングの概念や方法論について述べたもので，LISP の構文や AI のプログラミング技法を解説したものではない.

その題材としては，単純な数値計算から数式処理，論理型言語 (PROLOG) のインタプリタ，SCHEME のインタプリタとコンパイラの作り方まで幅広くカバーしている．そして，入門書ではあるが，imperative, applicative, object oriented, logic-based という現代のプログラミングにおける四つのスタイルが取り上げられている.

また，入門書という性格のため，理論を説くというよりも，プログラミングのスタイルの原理 (Element of Style) とプログラミングの美的感覚を肌で感じさせるために数多くのプログラム例が掲載されている.

そして，これらを通じて理論的研究へと誘っており，コンピュータ・サイエンスへのよき導入となっている.

また，書き方としては，平明であり分りやすい．ただ，何分にも500ページを越える大部なのが難点である.

プログラミングは discipline なので，オン・マシンで問題をやりながら気長にコツコツやることをお勧めする．SCHEME の構文は，LISP 1.5 と大分異なるが，本書の第4章に SCHEME で書かれた SCHEME インタプリタが掲載されているのでLISP で書き換えて使用するとよいだろう.

本書は，5章で構成されているが，内容としては三つの部分から成り立っている．まず，最初の三つの章で一般的なプログラミングの方法論について述べ，つぎに第4章では SCHEME と論理型言語のインタプリタ．さらには，第5章では仮想的な register machine 上の SCHEME インタプリタとコンパイラ等について解説している.

各章の内容を個別にみると次のようになる.

1章: Building Abstractions with Procedures

プログラミングの基本概念および pocedure abstraction について述べている.

1.1 では，SCHEME の構文および式の評価に必要なモデルとして，代入モデル (sabstitution model) を導入し，プログラミングの基本事項について説明する.

1.2 では，procedure と process との関係に触れ，SCHEME は tail recursive procedure が iterative process として計算されることを述べる．ここでは，ベキ乗計算・最大公約数の計算等の例が取り上げられている.

1.3 では，高水準プロシージャによる抽象化，つまりプロシージャを引数や値として使用すると，任意の関数に関する総和・定積分・微係数・Newton 法による求根等が一般的に取り扱えることを示す.

2章: Building Abstractions with Data

data abstraction の概念および具体的応用例を紹介する.

まず，2.1で有理数のデータ表現を例として data abstraction の概念を紹介し，つぎに誤差計算に有用な区間演算 (interval arithmetic) をモデル化する.

2.2 では，sequence (リスト) の表記法に始まり，木，さらには抽象データを用いた記号微分・集合・Huffman 付号化木 (Huffman encoding tree) 等の表現について解説する.

そして，2.3 では，直交座標系・極座標系による複素数の generic な表現や LISP でよく利用される data-directed programming, さらにはメッセージ・パッシングについて述べる．最後に 2.4 で，data-directed な方法によって四則演算の generic operator を構成し，さらに多項式と有理関数の代数計算プログラムを例として取り上げる.

3章: Modularity, Objects, and State

大規模プログラムを構成するのに有用な二つの構成法を説明する．まず，最初の方法は，1章と2章の内容を集大成したもので，プログラムをローカル・ステートをもつオブジェクトとして捉える object oriented programming である．第2の方法は，計算の信号処理モデルとも言えるストリームを用いるアプローチである.

3.1 では，object oriented programming に必要なローカル・ステートをセットする代入文 set! (SETQ に相当) を導入する.

3.2 では，代入文によって破綻をきたした sub-

stitution model の代りに，environment model を導入する．

3.3 では，リストに対する変更命令（mutator）として，set-car!（RPLACA に相当）と set-cdr!（RPLACD に相当）等を導入して，キューやテーブル等の基本的 object を定義する．そして，つぎに object oriented programming の手法によって論理回路のシミュレータを作成する．つぎに，最近注目を浴びている constraints oriented programming の例としてセ氏とカ氏の変換を取り上げる．

3.4 では，代入文を排除した形でのモジュールの結合法であるストリームおよびdelayed evaluation を導入する．そして，ストリームを用い定積分の計算や，ローカル・ステートをもつobjectの例として 3.1 で取り上げた Monte Carlo シミュレーションをストリームを用いモデル化できることを示す．最後にストリームと object oriented の両アプローチの利害得失を論じる．

4章：Metalinguistic Abstraction

4.1 では，SCHEMEのインタプリタをSCHEME を用い実現する．そして，4.2と4.3では，SCHEME インタプリタを変更し normal-order evaluator や dynamic binding evaluator を実現できることを示し，つぎに multiple name space を実現する package について述べる．

また，4.4 と 4.5 では，論理型プログラミング言語のインタプリタを実現する諸技術として unifier, rule database の保守用ユーティリティ等の作り方を紹介する．

5章：Computing with Register Machines

5.1 では，レジスタとスタックを備えた仮想機械のシュミレータを説明する．そして，5.2 では，この register machine 上で，SCHEME の explicit control evaluator を稼動させる．そして，5.3 では SCHEME コンパイラ，5.4 では storage allocation と garbage collection 等の技法を説明する．

LISP 系のプログラミングの概念を理解するにはこの一冊で十分だろうが，LISP を使ってプログラムを書く際には，P. H. Winston, B. K. P. Hornの "LISP (2nd edtion)" (Addison-Wesley, 1984) 等を読むのがよいだろう．Winston の 1981 年の初版（邦訳あり）は PROG の多用等があり，プログラミングのスタイルが古かったが，最近 COMMON LISP で書き換えられモダナイズされた．

また，前述の Charniak, Riesbeck, McDermott の "Artificial Intelligence Programming" は，やや高度であるが，AI 分野における比較的最近のプログラミング技法を修得するのに最適である．

このほか，P. Henderson の "Functional Programming—Application and Implementation" (Prentice-Hall, 1980) も LISP の良い教科書である．Henderson の本では，SECD マシン上に LISPKIT LISP と呼ぶ言語のインタプリタとコンパイラを構成している．

Avron Barr/Edward A. Feigenbaum 編
田中幸吉／淵　一博 監訳

人工知能ハンドブック（全3巻）

共立出版(株)，第1巻：xviii＋512 pp., 1983年
第2巻：xiv＋566 pp., 1983年
第3巻：xx＋826 pp., 1984年

人間の知的活動をモデル化する人工知能（AI）の研究は，1961年に MIT の Minsky 教授の著名な論文「Steps toward Artificial Intelligence」を契機とする比較的新しい分野の科学である．1965年には Stanford 大学の学際的プロジェクト Heuristic Programming Project (HPP) が Feigenbaum によって創設された．さっそく彼は，Nobel賞受賞の遺伝学者 J. Lederberg と共同で，質量分析と核磁気共鳴分析のデータから有機化合物の構造を推定するDENDRALの開発に着手し，1967年に完成している．このシステムは最初の成功した知識準拠システム（Knowledge-based System）であり，これによって人工知能システム実用化の道が開かれた．

1970 年代の前半になると，人工知能の研究は，Stanford 大学，Carnegie-Mellon 大学，MIT, Stanford 国際研究所 (SRI) だけでなく，BBN, Rand Corp., Naval Research Laboratory, South California 大学の ISI (USC/ISI), Xerox PARC 等でも行われはじめた．

さらに 1970 年代中頃になると，Feigenbaum らはエキスパート・システム構築のボトルネックとなっていた知識獲得の問題を解決するために Meta-DENDRAL を開発した．Meta-DENDRAL は，大量の質量スペクトル・データから，DENDRAL 用の経験則を抽出するもので，実際に専門家の評価に耐える法則が発見されている．さらに1974年には，同プロジェクトの E. Shortliffe が医学部の協力を得て血液伝染病と髄膜炎の診断・治療用の助言システム MYCIN を開発している．

Stanford 大学の HPP 等における知識準拠システムの実用化の成功に刺激され，1980 年には米国

人工知能学会 (AAAI, American Association for Artificial Intelligence) が組織された.

その後, コンピュータの発展とともに格段の進歩をとげ, とくにここ数年はその事業化を志向して, 多数の研究が進められている.

今日わが国でも, 第5世代コンピュータ・プロジェクトを始めとして AI 関係の研究が活発になっていることは衆知のとおりである.

本書は, このような AI の過去25年間の主な研究の集大成として, 先の Feigenbaum を編者の1人にすえて作られた. 3巻, 約2000ページからなり, 第Ⅰ巻は AI の基礎技術を紹介し, 第Ⅱ巻は AI の応用を中心に解説している. 第Ⅲ巻はこれからのテーマを主に取り上げている.

以下, 各巻を順にみてみよう.

第1巻は五つの章から構成されている. 第Ⅰ章「序論」は,「人工知能とは何か」を論じるとともに, 学習のためのガイドと参考文献を提示している. 第Ⅱ章「探索」では, 問題解決における探索全般について概観するとともに, 探索技法の基礎となる"問題表現"や, それらさまざまの問題表現を用いたアルゴリズムを扱い, さらによく知られている初期のプログラムのいくつかを考察している.

第Ⅲ章「知識表現」では, まず初めに, AI システムで表現する必要がある何種類かの"知識"を簡単に述べ, つぎに異種の知識表現法を論じたり比較する場合の用語として役立つ課題をいくつか紹介している. さらに, 重要な表現形式をいくつか概観し, 最後に種々の表現形式のメカニズム, 研究開発の歴史, 現在の開発上の問題点等を説明している.

第Ⅳ章「自然言語理解」は, 自然言語で表れされている意味を広い範囲にわたって扱える計算機プログラムの開発を目的とする章である. まず, 自然言語処理研究の歴史と自然言語における処理技法, そして機械翻訳の歴史について簡単に述べている. つぎに AI の研究者がプログラムの中で用いている文法と解析手法のいくつかを述べている. また, 文章生成のシステムや個々の NL プログラム自体についても述べている.

第Ⅴ章「話し言葉の理解」は, 音声理解すなわち話された言語を理解する計算機インタフェースの構成を目的としたものである. 他と同様, まず歴史・現状を述べ, さらに音声理解システムのアーキテクチャの解説をしている. そして, 米国国防省 ARPA の援助プロジェクトの成果を紹介している.

第Ⅱ巻も, 五つの章から構成され, AI の三つの分野 (プログラミング言語, エキスパート・システム, 自動プログラミング) を取り扱っている.

第Ⅵ章「人工知能研究用プログラミング言語」は文字通り言語とプログラミング環境について述べたものである. とくに AI 分野で作られた重要言語 LISP を解説し, さらにこの他の AI の主要言語 PLANNER, CONNIVER, QLISP, POP-2, SAIL, FUZZY の六つと比較し, AI におけるプログラミング言語の研究で探究された問題点を検討している.

第Ⅶ章「応用指向の人工知能研究：科学」, 第Ⅷ章「応用指向の人工知能研究：医療」, 第Ⅸ章「応用指向の人工知能研究：教育」は, それぞれの分野のエキスパート・システムについて述べている. これらエキスパート・システムの研究意図は, 必要な量と種類の知識を人間専門家から AI システムへ移植する方法を見出すことにあった. そして, この技術は数年の間に経済的に引き合うようなところまで進むであろう.

第Ⅹ章「自動プログラミング」では, まず, ユーザの求めるプログラムを自動プログラミング (AP) システムを伝達する手段, すなわち仕様を記述する方法について, その長所短所を含めて, 詳しい議論をしている. その後に AP システムの基本的アプローチについての説明がある. 最後にこれまでいろいろなプロジェクトで開発された AP システムを紹介している. これらのプロジェクトは AP 研究の四つの基本問題 (変換規則, 効率の探求, 部分情報の取扱い, 明示された理解) を扱ったものである.

第Ⅲ巻は, 現実世界においてコンピュータが知能的に振る舞うような機能をもつための基本的な処理過程を論じたもので, 六つの章から成っている.

第Ⅺ章「認知のモデル」では, AI と認知心理学の境界領域の紹介をしている. すなわち, 計算機科学者と心理学者との交流の成果といえる八つのモデルをとりあげたものである. 八つのモデルとは, 五つの人間の記憶のモデル (EPAM, HAM, ACT, MEMOD, Quillian のモデル) と二つの問題解決モデル (一般問題解決プログラム, 臨機応変的問題解決) と信念システムについてのものである.

第Ⅻ章「自動証明」では, 各種推論のモデルについて議論している. すなわち, 自然演繹法, 導出原理に基づく演繹法, 帰納推論, 非単調論理の形式的解析が示されている.

第ⅩⅢ章「コンピュータ・ビジョン」では, 物理世界の表現方法と, 一つの表現から他の表現への推論的変換方法との開発・研究を紹介している. 積木世界の研究を年代を追ってサーベイし, つぎに画像の特徴の初期処理技術について議論している. さらに

シーンの性質の表現の問題，特徴の照合と推論に対するアルゴリズム的道具について議論している．最後に，種々のビジョン法が全ビジョン・システムにいかにして統合されるかを紹介している．

第XIV章「学習と帰納的推論」では，四つの学習状況（暗記学習，言われるままの学習，例題からの学習，類推による学習）を詳細に分析し，各状況における学習問題の典型例を与えている．さらに，これら四つの状況のそれぞれに関連する個別の話題を解説している．

第XV章「計画と問題解決」では，計画に対しての四つのアプローチを紹介している．すなわち，STRIPS と HACKER によって実行された非階層的計画，ABSTRIPS, NOAH, MOLGEN によって用いられた階層的計画である．このテーマは，先の学習とともに現在もっとも研究されている分野である．

AIは，“知能の本質は何か”という科学上の重要な問題に貢献する分野ではあるが，かならずしもその諸概念，研究方法，技法等，広く他の分野に理解されているとはいえない．本書は人工知能の研究者だけでなく，実務家および学生にとってもすすめたい書である．

<Sperry 関係の論文・講演等の要約>

●オンライン・ユーザ・マニュアル——最近のソフトウェア設計では，主機能の充実よりも使いやすさ(ease of use)やユーザ・フレンドリネスが強調されることが多い．これらの機能を実現する方法としてディフォールト値，メニュ選択，キーワード・リスト等を使用し，選択の範囲を狭め複雑さを軽減することがよく行われていた．しかし，プログラミング言語のように複雑なシステムではこの方法を採用できないことが多い．

第2の方法としては，ヘルプ機能のようなオンライン・アシスタンスの開発が考えられる．しかし，オンライン・アシスタンスは，開発・保守コストの負担が大きいため，その普及は遅れている．

本稿のオンライン・アシスタンスは，シリーズ1100のLISPを対象として開発されたシステムである．このシステムの開発の背景としては，LISPが200を越える組込関数を有し，しかもその中にはEditor関数等のような数ダースのコマンドを備えているものもある等，LISPが膨大な機能を擁しており，ユーザがそれをマスターするのが大変な点にある．

同システム(@HELPと呼ばれる)は，CTSのHELP機能に参考にして開発されたもので，マニュアルの各章名や組込関数名等によって内容を検索でき，しかも出力形式として利用者のレベルにあわせて内容が選択できる．

本システムの開発に当たっては既存のCOMADS形式のプログラマ・リファレンス・マニュアル，EDプロセッサ，PDPプロセッサ，等の諸ツールを最大限に利用し，開発の負担を軽減したことが挙げられる．このため，従来のプログラマ・リファレンス・マニュアルにEDのマクロを使って会話の制御情報を付与し，オンライン・アシスタントのデータベースとして使用し，別途にオンライン・アシスタンスのデータベースを作成しなかったことも大きな特徴である．これによって保守が容易化し，長期的メリットが得られるようになった．なお，本システムは，LISPだけでなく他の言語用のオンライン・マニュアル・システムとしても利用可能であり，サブルーチン形式およびプロセッサ形式の両方が用意されている．(R. M. Firestone, "Online PRM: Assistance Technique", Sperry, Spring 1984)

●PASCALコンパイラのテスト法を報告——UCS(Universal Compiling System)で記述されたPASCALコンパイラの開発は1981年に溯る．同年，初期設計と同時にテスト計画が策定された．計画にあたっては，①外部からのテスト・プログラムの入手，②他言語のテスト・プログラムからの変換，③テスト・プログラムの新規開発をミックスする方針で進んだ．

まず，第1にオーストラリアのTasmania大学と英国のNational Physical LaboratoryのWichmannによるPASCAL Validation Suite，Arizona大学のRiployらのテスト・プログラム等を入手した．なお，PASCAL Validation SuiteはISOの標準案を支援するために設計されたものである．第2に自社のFORTRANコンパイラのテスト・プログラムやArgonne National LaboratoryのFORTRANテスト・プログラムELEFUNT等をPASCAL用に変換した．

このほか，これらで欠けていると思われるテストとして，外部プロシージャ，ファイル関連のテストや，スコープ・ルール，型合せ等，実現法によって異なる機能のテスト・プログラムがあり，それらを新規に作成した．

そして，テストを自動化するためにTCS(Test Controller System)と呼ばれるテスト管理システムを開発した．このシステムは，ユーザとのインタフェースとテストの実行制御を行うコントローラ・ルーチン，テスト結果の評価，ループのチェック，報告書作成等を行うステータス・ルーチンで構成される．そして，各テストに対してジョブ制御エレメント，テスト・プログラム，ベース・エレメント(正しい実行結果)の1組が用意されており，テストの実行値がベース・エレメントと自動的に比較されるようになっている．

このほか，本稿では，PASCALのテストを通じて得られた経験をもとに，社外テストの重要性，テスト・サイズの決定，テスト盲点の存在等の問題を考察している．(H. K. Seyfer, "The Testing of Compilers", Sperry, Spring 1984)

●ノンインパクト方式の印書技術を展望——オフィスでの紙の使用は，将来的にはなくなるかも知れないが，現在はそれと反対の動きを見せている．実際プリンタの売上げは，10年前の2倍の24億ドルの規模にまで拡大している．また，従来のタイプライタのかわりに感熱式，レーザ方式，インク・ジェット方式のプリンタが使用される見込みである．そ

して，その中でインパクト・プリンタの売上げは，1988年には25億ドルとなろう．これは現在の水準よりやや高い程度であり，横ばい状況を呈すると予想される．一方，ノンインパクト・プリンタは，昨年の440万ドルから1988年には26億ドルに達すると見られている．そして，ノンインパクト・プリンタの中では，レーザ方式が主流になる可能性が高い．

本稿は，ノンインパクト・プリンタの諸方式をサーベイしており，現状での問題点やその技術予測等についても述べられている．なお，取り上げられている技術は，電子写真（レーザ・プリンタ），磁力記録（磁気記録），イオン・デポジット，感熱印書，インクジェットの5方式である．(B. J. McDevitt, "A Current Look at Non-Impact Technology", Proceeding of the USE Inc., April Conference, 1984)

●非手続的言語をサーベイ——コンピュータ言語は，機械語からアセンブラ，コンパイラと進歩してきたが，次のステップとして手続的言語から非手続的言語への発展しようとしている．非手続的言語は，PROLOG, SETL, SNOBOL 等に代表される言語で，表示的言語あるいは宣言型言語と呼ばれることもある．非手続的言語は，求むべき結果(What)を記述する点で，結果を得るためのプロセス(How)を逐一記述する手続的言語と異なる．

非手続的言語の特徴としては，①アプリケーションへの依存，②データの値による連想的な参照，③データ構造に対する操作のインプリメンテーション独立性，④パターンによる実行順序の決定，⑤代入文の欠除，⑥命令文の順序とプログラムの実行順序との独立，⑦プログラムの非決定的実行等が挙げられる．

また，非手続的言語の長所は，①アプリケーションに依存した非手続的言語では仕様のチェックが行える，②並列計算が可能，③プログラム・サイズが小さい，④可読性が高く開発期間が短かい点にある．一方，短所としては，①所要記憶容量が大きい，②モジュラリティの欠除，③他言語とのインタフェースが貧弱，④実行速度が遅い，⑤適用分野が限定されている，⑥経験者が少なく，教育の必要性がある等が挙げられる．

非手続的言語は，10年くらい前から使用されているが，あまり浸透していない．しかし，コンピュータ・ネットワークの普及によって非手続的言語を使用できる環境が整いつつある．このほか，非手続的言語の普及を妨げていた理由に実行効率の悪さがあったが，コンパイラ等の開発によってその問題も解決されるようとしている．将来，非手続的言語が，FORTRAN や COBOL のように広く使用されるとは考えられないが，プロトタイピング，情報検索，ドキュメント作成，言語認識，人工知能における知識表現，数学・財務・計量モデル作成等の分野で使用されよう．

このほか，プログラムの理論のチェックや検証にも多用される見込みである．(T. N. Turba, "An Introduction to Nonprocedural Languages", Sperry, Spring 1984)

■ ■

●情報処理会議 '84 (Convention Informatique)——Paris, 1984年9月17日～21日

同会議は，欧州最大級の情報処理関連見本市SICOB (International d'Informatique, Telematique, Communication, Organization du Beureau et Beureatique, 情報処理・通信・事務機器国際見本市)に併設されたもので，今回で15年目を迎えた．

本年は市場動向，技術開発，応用，マイクロコンピュータ，経済的・社会的側面等の各セッションに分かれて，講演と次のようなパネル討論が開かれた．

市場動向では，オフィス・オートメーション，ソフトウェア・パッケージ，マイクロコンピュータ等であった．

技術開発では，シストリック・アーキテクチャ，データフロー・アーキテクチャ，垂直記録・磁気・光システム，メインフレームの将来，科学用パーソナル・コンピュータのネットワークにおける分散OS ACCENT，オブジェクト指向システム，電子ドキュメンテーション・システムとしてのメイル・ボックス，ドキュメント分析システムDIVA，署名の動的認識，分散型データベースと衛星通信，フランスにおける通信インフラストラクチャ，ビデオ・ディスクを利用した会話型プログラム，人工知能とエキスパート・システムの現状と企業における適用，第4世代ソフトウェア・システム，形式的仕様言語の現状等．

応用では，企業と銀行との将来の関係，保険業における意思決定支援，フランスにおける電子支払いと Videotex，農協におけるコンピュータ利用，現代のコミュニケーション手段としての Videotex の位置付け，Videotex による高度セキュリティ電子メール，産業用ローカル・ネットワーク等．

マイクロコンピュータでは，ミニコンピュータ用のデータベース(SHAKO, DIALOGUE)，ポータブル・コンピュータ，マイクロコンピュータによるコンピュータ援用教育 CAT (Computer Aided Teaching)，多機能・多重ウインドウ・パーソナル・コンピュータ，スプレッド・シート・ソフトウェアの動向，データ通信におけるマイクロコンピュータの利用等．

経済的・社会的側面では，EDP 投資の計画と統制，EDP 監査，EDP 教育，EDP 労働市場，EDP プロフェショナルの意見調査，コンピュータ部門の役割等であった．

●情報処理学会第29回全国大会（昭和59年後期）——仙台，1984年9月11日～13日

今回の特別講演と招待講演は，西澤潤一氏（東北大）と田崎京二氏（東北大）によって，それぞれ「21世紀の技術に向って」および「脳研究とポジトロンCT」と題して行われた．このほか，「知的プログラミング環境の諸問題」および「新しい情報処理デバイスへの展望——新しい計算機システム構築のために」をテーマにパネル討論が開かれた．

また，一般講演は，推論マシン(2)，アーキテクチャ(4)，データ駆動マシン，ベクトル・プロセッサ，並列プロセッサ，データベース・マシン(2)，周辺装置，ハードウェア，ディスプレイ，論理回路設計，レイアウト設計(2)，検証，論理シミュレーション，ハードウェア記述，マイクロコンピュータ(2)，基礎(2)，OS(6)，プログラミング言語処理，言語処理系コンパイラ生成系，プログラミング・ツール(4)，システムの実現技術(2)，ソフトウェア開発管理保守(2)，ソフト開発環境(2)，ソフト保守支援，ソフトウェア設計(3)，ソフトウェア品質，ソフトウェア信頼性，データ・モデル，分散データベース，情報検索(2)，統計データベース，応用データベース，情報センタ・システム，性能評価(2)，オフィス・システム(3)，ローカル・ネットワーク，広域網，ネットワーク管理，通信ソフトウェア，ネットワーク・サービス・システム(2)，ネットワーク性能評価，日本語／文解析，日本語／翻訳，日本語／入出力，日本語／辞書・単語分析，日本語／仮名漢字変換，日本語／処理システム，形状処理，CAD システム，画像処理(2)，形状認識，文字認識，パターン処理，文書画像処理，画像処理装置，自然言語処理(2)，言語理解，人工知能，機械翻訳，知的設計支援システム，エキスパート・システム，知識獲得，知識ベース，人工知能向き言語，PROLOG(2)，オブジェクト指向言語，音声認識，ロボット，数値計算(2)，教育，社会システムの各セッションに分かれて行われた．なお，括弧内は同一テーマのセッション数である．

今回は，ICOT 関連（推論マシン等），日本語，人工知能，プログラミング・ツール，オブジェクト指向言語等が注目された．

●COMPCON Fall '84——Arlington (Virginia州)，1984年9月16日～20日

今回は「小型コンピュータ革命」を基調テーマとして，マイクロコンピュータの新しい応用に焦点を当てて開かれた．そして，会議に先き立ち各パソコ

ンのユーザ・グループ，たとえばCapitol PC(IBM PC), FOCUS (Commodore 64), TI (TI99/4A), Apple (Apple) の意見の交換会が開催された．また，これに続くチュートリアル・セミナでは，コンピュータ応用部門，1980年代のエンド・ユーザ・ファシリティ，コンピュータ入門，実務家のためのソフトウェア品質保証，ローカル・ネットワーク技術，分散型データベース管理システム，小型コンピュータ用のADA，小型コンピュータ用C言語等がテーマとして取り上げられた．

このほか，今回のパネル討論では，ソフトウェアの所有権，軍用システムにおけるソフトウェアの安全性とセキュリティ，VHSIC用の統合設計自動化システム，日本と米国におけるソフトウェア生産方法，空母搭載の非戦略ミニコンピュータ・ネットワーク，ソフトウェア開発工程における小型コンピュータの影響予測，国防省のSTARS・ADA・SEI計画，小型コンピュータ・ユーザ・グループ等をテーマとして行われた．

また，一般講演の主なテーマは，パーソナル・コンピュータにおける人工知能，ソフトウェアの移植(Renosting)，小型コンピュータの注目すべき応用事例，米国における小型コンピュータの応用，ソフトウェア開発ツールとワークステーション，専用言語，小型コンピュータによるプログラム管理，マイクロコンピュータのネットワーキング，高度アーキテクチャ・プロセッサ，パーソナル・コンピュータによるCAI，ソフトウェア開発環境，LAN技術等であった．

●COMPSAC '84 (Computer Software & Applications Conference) ——Chicago, 1984年11月5日～9日

今回のチュートリアル・セミナのテーマは，構造的テスティング，プログラミング言語ADA，自動推論の入門と応用，コミュニケーション・ネットワーク・プロトコルの基本をテーマとして行われた．また，今回のパネル討論のテーマは，通信ソフトウェア開発環境，データベース・システムの新傾向，電子交換用ソフトウェアのアーキテクチャ，ソフトウェア工学による品質向上，応用ソフトウェア開発におけるソフトウェア工学の課題，エキスパート・システムの開発ツール，統合サービス・ディジタル・ネットワーク(CCITTの見解)，ライフサイクルにおけるソフトウェア尺度，ソフトウェアの米国標準，ソフトウェア品質保証，技術計算ソフトウェアの検証法，国防省のソフトウェア計画，通信ソフトウェア開発におけるエキスパート・システム，ソフトウェアの再利用，ソフトウェア品質評価，IEEEのソフトウェア工学標準活動の現状等であった．

このほか，一般講演では，ソフトウェア・ツール，ソフトウェア工学環境，並列性制御，ソフトウェア工学管理，データベース応用，分散データベース処理，分散型プログラミング，ソフトウェア開発用言語，ソフトウェア設計技術，人工知能応用，新ソフトウェア方法論，ソフトウェア信頼性，プロセス制御システムにおける妥当性チェック・検証等，開発環境用のOS等が話題として取り上げられた．

●自然言語処理シンポジュウム——東京，1984年11月6日～7日

今回は，吉田 将氏(九大)の基調講演「言語工学の推論」，木下是雄氏(学習院大)の特別講演「理科系の作文技術」と四つのセッション，検索・対話・支援システム，意味理解と表現，機械翻訳，言語解析で合計22論文が発表された．最初のパネル討論では，「自然言語の知能的処理」(司会・武蔵野通研野村浩郷氏)をテーマに開かれた．

基調講演において吉田氏は以下のように述べた．自然言語処理は日本語ワードプロセッサの出現により多くの人々の理解を得ることができたこと，(カナ漢字変換を例に掲げ)当初難題と思っていた同音異義語処理はそれほどたいしたことではなかったこと(実際，自然言語処理の上では未解決であるが)，むしろ人間の特性を生かすことで解決していること，さらに最近とくに盛んとなっている機械翻訳の研究は，時代の要求を反映したものであり，しかもコンピュータの歴史と同程度に古く，むずかしいテーマであること，control languageについては今後急激に広まるであろうと述べている．また，いくつかの提言もあり，一つに構築された自然言語処理システムについては，何ができて何ができないかを明確にする必要がある．そのためには，評価方式を確立する必要があり，評価のための文(章)のセットの開発が必要であること，また，Toy modelでの研究に留まることなく研究を進めていく上では，計算機可読な形になっている辞書等は，共通財産とし共有化を図る必要があること，さらに人工知能的言語処理やアルゴリズムの研究の重要性を忘れずに，より高度な自然言語処理を目指す必要があると主張していた．

シンポジウム2日目の木下氏の特別講演は次のとおり．米国における英語教育で，表現力の涵養が具体的な教程として小学校から実行されている．こ

れは，文学鑑賞を主とする日本の国語教育と様相を異にする．たとえば，そこでは事実と判断の峻別が教えられる．これは一見やさしそうに見えるけれども，実はそれほど簡単ではない．「AはBである」という肯定文が，事実を述べることをはっきり示すためには，たとえば「AがBであるという事実がある」という形にすることが必要である．「…と思われる」という形の文は皆がそう思う公知の事柄なのか叙述者だけがそのように思うという判断なのかわからない．事の本質は，事実と判断の区別にだけあるのではなく，多岐にわたる．また，このような教育は大人になってからでは手遅れである．したがって，現在教科書を作っている，と述べていた．

セッション1（検索・対話・支援システム）では，①高順位単語の生起特性を利用した情報検索法，②自然言語の分析による知識データの収集，③日本語索引自動生成システム，④対話型英日機械翻訳システムのための曖昧性解消フロントエンド，⑤J-TUTOR：A Kanji Oriented Japanese Language Education System，⑥定形文章コンパイラとその文章作成支援システムへの応用の論文が提出された．

セッション2（意味理解と表現）では，①はい・いいえ質問における情報付加の枠組について，②日本語の算数文章題の理解・解決システム，③文脈解析用意味表現 I-MOP の構造と機能，④文脈の表現と学習の一方式，⑤テキスト処理へのアプローチの5論文が発表された．

セッション3（機械翻訳）では，①外国語手紙文の自動作成について，②語法規則変換方式による機械翻訳：翻訳の表現式と翻訳文主成関数，③より自然な翻訳へのアプローチ[I]：英日翻訳における表現の対応，④Mu プロジェクトにおける辞書の運用方式：日英変換辞書と英語生成辞書であった．

その他セッション4（言語解析）では，以下の7論文が発表された．①属性文法評価システムによる自然言語処理，②アフター指向型を導入した解析制御，③An English Sublanguage Parser Based on DCG，④自然言語の認識過程の心理的なモデル，⑤日本語の文法情報の処理について，⑥日本語技術文における並列構造と文の概形，⑦文書理解システムの試作．

最後のパネル討論は，武蔵野通研／野村，埼大／荻野，長崎大／鶴丸，京大／辻井，武蔵野通研／島津，電総研／石崎らの各氏により行われた．パネラの中に言語学者である荻野氏を加えたことで学際的興味が刺激される雰囲気であった．シソーラスの作成が主要論題の一つであった．既刊のシソーラスには Roget のものを初めとしていくつかあるが，これには人間が使用することを前提としたものである．機械による自然言語処理用のシソーラスとなると機械には常識がないので，どんな特徴によって二つの言葉が類語であるかといったことの明記が必要となる．ところでシソーラスの作成には多大の作業と期間を要するが，現状ではこつこつやればそのうち機械処理に使えるようなものができるかというと答は否定的で，まず言語学者がシソーラスを作る環境にいないこと．工学系の人が語るシソーラスもどんな情報をシソーラスに盛り込むことが必要十分かが既知となっているわけではないので，できたとしてもただちに機械による目的言語処理に使用できるものとはならないこと．したがって，シソーラスも使用目的に応じて作ることが必要となりそうであること等が指摘された．

●第5世代コンピュータ国際会議――東京，1984年11月6日～9日

同会議は，通産省をバックに官民を挙げて次世代コンピュータの研究開発に当たっている「新世代コンピュータの研究技術開発機構（ICOT）」の主催で，開催された．日本で第5世代コンピュータに関する国際会議が開かれるのは1981年10月に次いで2度目だが，実際に同コンピュータの研究開発が始まってからは初の国際会議であり，参加人員約1,200名のうち海外からは30か国約300名の研究者の参加があり，第1回会議と比べて増加しており，第5世代コンピュータに対する各国の関心の高さを浮き彫りにした．4日間の会期中，前半2日間の全体会議では，基調講演，招待講演，および ICOT の研究開発状況の成果発表と同時に，各国の第5世代関連プロジェクトについてパネル討論が行われた．後半2日間ではテクニカル・セッションとして招待論文（3件）と一般論文（62件）の発表ならびに将来の並列処理に関する展望について専門家のパネル討論が行われた．

初日の基調講演でICOTの淵一博研究所長は「プロジェクトがスタートしてから2年半の間，さまざまな研究体験と国内外からの反響でICOT発足後初に打ち出した基調理念を，より深く確信できるようになった」と述べ，同プロジェクトが研究開発の基本として取り組んでいる新しいコンピュータ技術の枠組，すなわち「論理系をベースにして，コンピュータのハードウェアとソフトウェアの体系を再構築すること」に自信を示した．また，ICOTの川野辺一清研究所次長による「研究開発の現状と今後の計画」

の発表では，「研究開発は前期3年，中期4年，後期3年の10年計画になっている．現在，前期計画の最終年度にあり，研究開発は順調な進捗を示している．前期の重点は知識情報処理の分野で，これまで培ってきた研究成果を収集し，その評価と再構成を行い，中期に向けての基本技術開発を行うことである．中期は前期の成果をもとに，ソフトウェア，ハードウェアの基礎となる計算モデル，実現アルゴリズム，基本アーキテクチャを設定し，それにそって，小・中規模のサブシステムを試作すること．後期ではソフトウェア・システムとハードウェア・システムとの間の機能配分を行い，トータルシステム(プロトタイプ)を試作することが目標である．」と述べ，ICOTの研究開発の現状と中期計画(案)について紹介を行った．

パネル討論「各国の研究開発事情と社会への影響」は，日本，米国，英国，フランス，西ドイツ，ECの各国の第5世代プロジェクト関連代表によって行われ，各国の研究開発事情が紹介された．欧米では日本の第5世代プロジェクトに刺激されて，最近，各国で政府主導の先端的コンピュータ開発プロジェクトが推進されており，英国のALVEYプログラム，ECによるESPRIT計画，および米国の国防総省ベースのプロジェクトと民間企業による共同研究プロジェクトMCC等がある．

テクニカル・セッションは会場を三つに分けて，四つのテーマ，①論理型プログラムの基礎理論，②論理プログラム言語とその方法論，③新世代コンピュータのアーキテクチャ，④新世代コンピュータの応用，について各国の研究者から発表があった．この中で，ICOTが開発した逐次型推論マシン(SIM)のハードウェア「PSI」，および，その基本ソフトウェア「SIMPOS」とそのシステム記述言語である「ESP」について，また並列処理に関して日本電信電話公社の武蔵野電気通信研究所が開発したデータフロー型の演算処理装置「DFM」が内外の参加者の注目を集めた．

第5世代コンピュータの研究開発は，わが国が世界に先駆けて着手したもので，現時点では世界の先頭を走っているが，わが国が第5世代コンピュータにある程度の技術的メドをつけたことで，米国を中心に，各国の研究開発競争は一層激しくなるものと思われる．

●IEEE第1回OA国際会議(1st International Conference on Office Automation)——New Orleans, 1984年12月17日～19日

この会議は，IEEEのコンピュータ学会の主催によるもので，システム・ユーザ・インタフェース，電子郵便，事務手続き，組織との関連，ドキュメント処理の各部門に分かれて20編を越える論文が発表された．このほか，ユーザ・インタフェースの使用経験，オフィス・システムのユーザ体験と題して二つのパネル討論が開かれた．

なお，一般講演の主なテーマは，システム関連ではオフィス・ワークステーション・ファミリNGEN，パーソナル・ネットワークApplebus，分散環境におけるサーバの設計，統合オフィス・システムLumiere等．

ユーザ・インタフェース関連では，Waterlooポート・ユーザ・インタフェース，2次元編集，グラフィックスを利用した会話技術等．

電子郵便関連では，オフィス・オートメーション・システムにおける遠隔音声アクセス，電子郵便システム実現に関して人的コミュニケーションの経済的分析等．

事務手続き関連では，オフィス手続きを支援する実験的ツールXCP，オフィス情報システムにおける承認，分散型オフィス・システムの抽象モデル等．

組織との関連では，OAシステムにおけるセキュリティ，コンピュータによって作成された証拠情報の法的有効性，オフィス情報システムの費用対効果分析(快感度による価値付け)，非EDP要員におけるコンピュータ文盲プロファイル等．

ドキュメント処理関連では，多重インテリジェント・ノード・ドキュメント・サーバーMINDS，文書の蓄積と対策ツールであった．

●ACM POPL (Principles of Programming Languages) Symposium——New Orleans, 1985年1月14日～16日

POPLは今回で12回目を迎えた．主なテーマは，並行処理プログラムの仕様記述，有限状態並行プログラムの仕様チェック，関数型プログラミング言語の表示的意味論と書き換え規則，プロセス・ネットワークのモデルと時制論理証明システム，Hoare的論理の論理的完全性，言語準拠エディタにおけるメタ言語とそのシステム，最適化コードの増分的コンパイル，増分的コード生成へのグリーディな並行アプローチ，並列コードの線形化，パス表現のVLSI回路へのコンパイル，アリアス化と型付けの宣言，低水準要求用の高水準言語機能，分散処理用のインタフェース仕様記述言語Matchmaker，知的再コンパイル，クロージャ化と集中化，関数的プログラミ

ングと論理変数，形式パラメタのアリアスの解析，条件分岐をもつ定数の波及，関数的言語と論理的言語におけるコピーの回避，意味指向のコード生成，記憶アクセスと並行な算術命令の最適スケジューリング，コード生成を支援する木の高効率パターンマッチング等であった．

NEW PRODUCTS

昨年夏期以降発表の製品の中から主なものを選んでご紹介します．各製品の詳細についてはマニュアル等をご参照ください．

●シリーズ 1100 FAS 1100

カタログ・ファイルの維持・管理のために，従来SECURE が提供されているが，データの互換性を保ちながら SECURE より使いやすく，SECUREの機能をすべて含み，なおかつ豊富な機能をもつ新しいソフトウェアとして FAS 1100 (File Administration System 1100) がある．

FAS の最も重要な機能は，あらかじめファイルのコピーを磁気テープ上に作り，必要なときにファイルを回復することである．第2の機能は，使用頻度の低いファイルや有効期限の切れたファイルを磁気ディスク装置のような高価で限りのある記憶媒体から，比較的安価で容量を簡単に拡張できる磁気テープへ移し，必要になるまで保存すること，これによりマスタ・ファイル登録簿が小さくなり，通常のファイル処理の効率が向上する．FASの第3の機能は，作成されたファイルの属性および容量について報告書を作ることである．(資料コード：481205132)

●シリーズ 1100 ファイル・サポート・プログラム

ファイル・サポート・プログラムは，システムの開発や運用で必要となるデータ・ファイルの作成，印書，複写，照合（比較）を効果的に処理するためのソフトウェア群であり4種類のプログラムから構成されている．

それらは，①テスト・データ・ファイルをレコード単位に作成する GFGEN（生成），②ファイルの内容をレコード単位に印書するGFPRNT, FPRNT (印書)，③ファイルの内容をレコード単位に複写し，複写の際レコード内の項目変更，レコードの追加・削除ができる GFCOPY, FCOPY (複写)，④二つのファイルをレコード単位に照合するGFCOMP, FCOMP（照合）プログラムである．ファイルの対象は，ASCII COBOL または，日本語 COBOL で処理可能な順書込み/順読出しファイルである．

(資料コード：481205480)

●シリーズ 1100 会話型文字ファイル保守プログラム/漢字パターン

会話型文字ファイル保守プログラム/漢字パターン (IKPGEN) は，会話形式で大容量記憶装置上の文字ファイルおよびパターン・ファイルの保守を行うプログラムである．

機能としては，端末装置の画面を見ながら，文字パターンの修正，新規作成が行え，文字パターンを文字ファイルまたは文字パターン・ファイルに登録することができる．また，確認リストの出力，文字ファイル保守プログラム (KLMTN) に引き渡す指示文を作成することができる．

(資料コード：481205401)

●シリーズ 1100 カナ漢字変換システム（住所）

事務処理を行う上で，氏名および住所は非常に多くの場合に使用される．カナ漢字変換システム（住所）は，現在多くのユーザにおいて英数カナのこれらデータを日本語へ移行する際に，また氏名や住所を日本語で扱う新規事務を開発する際に，使用することを目的に開発されたソフトウェアである．カナ漢字変換システム（住所）は，住所カナ漢字変換ライブラリ（ADRLIB）と住所辞書作成プログラム（ADRGEN)，姓・名・地名辞書作成プログラム（ADSGEN）および各種辞書等より構成されている．ADRLIB は，使用者プログラムより呼ばれるサブルーチン形式のプログラムであり，使用者プログラムより渡された英数カナの住所データを，各種辞書を検索して漢字に変換するプログラムである．ADRGEN は住所ファイルの全国版，分県版を入力して住所辞書を作成するプログラムである．ADSGEN は姓・名ファイルの更新，住所ファイルの更新により，姓・名・地名辞書を再作成するためのプログラムである．また，変換のために使用する主な辞書は，①全国の都道府県，市区町村名等19万件を収容した住所辞書，②日本人の姓・名・地名等18万件を収容した姓・名・地名辞書，③企業，事業所，組織，法人，団体等の名称6万件を収容した企業名辞書，④他に住所略号辞書，使用者追加住所辞書等がある．(資料コード：481205476)

●シリーズ 1100 SCHIP

SCHIP は，論理回路図に代表される配線図面とその図面が表しているシンボル相互間の結線情報を会話型グラフィック端末を用いて目視確証しながら計算機に入力するソフトウェアである．

機能として任意形状シンボルの登録機能，図面間

の論理的接続関係の定義機能等があり，多くの類似した図面を取り扱う分野への応用が可能である．また，入力された図面情報の保存，再生，製図，削除，修正等の管理機能もある．さらに，後続アプリケーション・プログラムとして論理回路解折(DIANA)，プリント基板設計（PPLS-Ⅱ）への標準インタフェースをサポートする．

（資料コード：483205414）

●シリーズ 1100 SUFICS 1100 (GRAPH)

SUFICS 1100 グラフは，SUFICS 1100 により構築したモデルの実行結果を各種ビジネス・グラフ（折線グラフ，棒グラフ，円グラフ，スター・チャート，バブル・チャート等）が作成できる会話型経営計画プログラムである．サポート端末としてUTS 50（Ⅰ型〜Ⅳ型），UTS 40，AGS 2000で，使いやすさを考慮しており，利用部門の担当者が2時間程度のトレーニングで使いこなすことができる．プロット・コマンドで指示されていない場合，システムにより仮定値が設定されるので数行のコマンドで作図できる．また，線の太さや活字等の変更も容易であり，一画面に複数のグラフを出すこともできる．

（資料コード：483205158）

●シリーズ 1100 VIDEOTEX 1100

ビデオテックスは家庭のテレビ等を編集機として，通信網経由でビデオテックス・センタと接続し，図形や文字で種々の情報を提供するサービスである．VIDEOTEX 1100 は，シリーズ 1100 を直接型情報センタとして接続する場合のサポート・ソフトウェアである．特徴として，画像データベースの構築，画像情報の検索，使用者プログラミングが不要で出力が容易であること。さらに，豊富で強力なトランザクション処理支援ライブラリを用いて使用者 TPS を作成することにより，リアルタイム処理結果や使用者データベース内のデータを画像データベース内の画像情報と合成し，利用者端末へ出力することができること，また，キャプテン PLPS（出力画像）を知らなくても，使用者プログラミングが容易にできる等がある．（資料コード：481203706）

●シリーズ 1100 FSP 1100

FSP 1100 (Facsimile Support Package) は，ファクシミリ通信網にシリーズ 1100 を接続したセンタ・エンド形ファクシミリ通信を利用したイメージ情報蓄積/検索システムの構築を容易にするサポート・ソフトウェアである．

特徴として，ファクシミリ通信網に接続されたファクシミリ端末を使用した集信，配信，問合せのいずれの形態の業務処理システムでも容易に構築できること，CORE 1100 等を使用して作成したグラフ・図形の DSP ファイルを入力とし，そのままファクシミリ端末へ出力可能なこと，さらに日本語プリンタ等へ出力するために作成したシンビオント・ファイルを入力とし，内容をそのままファクシミリ端末へ出力可能なこと等がある．

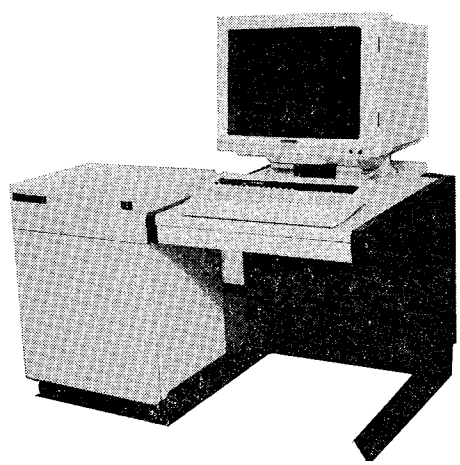
（資料コード：481203705）

●シリーズ 1100 Housing CAD

Housing CADは，ホストコンピュータとしてシリーズ 1100 を使用し，2次元ラスタ・スキャン型グラフィック・ディスプレイ AGS 2000 シリーズを使用した3次元住宅設計システムである．特徴としては，3次元家情報をもとに作業の進行に応じて必要な図面を出力することができること，設計・見積期間を大幅に短縮，設計変更にも即時に対応でき，顧客の予算を意識したプランの作成が可能，さらに概算見積・詳細見積が設計と同時に得られる等が挙げられる．すなわち，住宅設計における引合いから施工までをサポートする一貫したシステムである．

（本誌 p.22〜36 参照，資料コード：083201301）

●AGS 2000Fシリーズ グラフィック・ディスプレイ



高度なインテリジェント機能を備えた高速・高精度グラフィック・ディスプレイであり，設計者・技術者のための CAD/CAM をはじめ，ビジネス・グラフィックスの分野に利用される．

AGS 2000 F シリーズは，多様化する客先アプリケーションに応えるため，現在提供中のAGS 2400 F

に機能拡張を行うと共に，ホスト接続のバリエーションを大幅に強化し，また従来のペデスタル型にセパレート型を加え5モデル10機種とした．

AGS 2000 F シリーズ共通の特徴として，①60 Hz ノンインタレース方式による安定した表示，②1024色同時表示による多彩な画面，③分解能 1280×1024 ピクセルによる高品質表示，④大容量セグメント・バッファを利用した高速の図形編集機能等，豊富なインテリジェント機能がある．

拡張された AGS 2000 F の機能・特徴は以下のとおりである．

①図形処理機能の強化と画像処理機能の拡充……ヒットした図形要素を指定した色に変えるヒットエコー機能，ユーザ定義可能な中塗りパターン，拡大，縮小，回転の倍率定義，ピクセル・イメージ・データの高速表示・読取り等13種のコマンドの追加，さらにカラー・ルックアップ・テーブルの拡張（1678万色）等が追加されている．

②ホスト接続の豊富なバリエーションと高速転送……従来の ASYNC/フリーランに加え，高速シリアル・インタフェース，BSC 3270 エミュレート，16ビット・パラレル・インタフェース等があり，さらにバイナリ転送方式モデルを加え，ホストとの高速データ転送を可能にしている．

③設置環境の多様化への対応……設置面積および操作性重視のペデスタル型（床置型）に加え，ダイナミックなシステム構築を可能にするセパレート型（モニタ/コントローラ分離型）も提供する．

（資料コード：081841020）

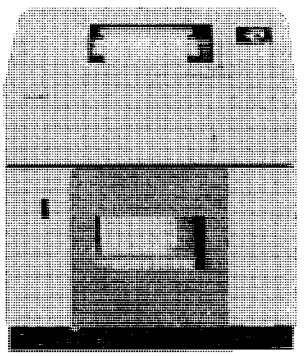
●UTS 4000 ターミナル・システム ファミリ TIMS

TIMS は，UTS 50 モデル Ⅲ/Ⅳ の CP/M のもとで稼動するリレーショナル型データベース・ソフトウェアである．メニュ選択方式とガイダンス機能により，操作はプログラム知識を必要としない．主な特徴として32,767件のデータを扱える大容量ファイル，10個までの条件を指定できる強力な検索機能等がある．また，Micro REPO および MAPPER とのデータの授受が可能である．

●0469 型印書装置

0469型印書装置は，UTS 50-Ⅲ/Ⅳ 型にのみ接続可能な低価格，中速度の日本語ラインプリンタである．仕様として，印字方法はインパクト・ラインドット方式を採用，印字速度は ANK で毎分159行，漢字で毎分140行，一行最大印字は ANK で136字/行，漢字で90字/行である．印字文字種は ANK は英小文字を含み160文字，漢字は標準で JIS 第1水準，JIS 非漢字，オプションで JIS-2 漢字フォント ROM フィーチャがある．

●0455 型印書装置



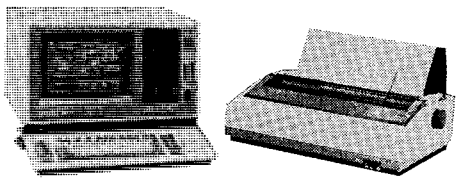
0455型印書装置は1100/60, 70, 80および90システムのバイト・マルチプレクサ・チャネル，またはブロック・マルチプレクサ・チャネルに接続されるオンサイトの高速ライン・プリンタである．特徴として筐体寸法1100×915×1.250mm（幅×奥行×高）とコンパクトで，消費電力も 2.0 KVA と省エネルギー・タイプであり，騒音を抑える工夫が施されている．操作員の介入を必要とするような警告およびエラーは一目でわかるように絵で表示し，マイクロ・プロセッサを用いて用紙走行速度の制御を行う等の信頼性をアップするために各種技術が取り入れられている．仕様としては，プリント・バンドを使用したインパクト・タイプのプリンタで，110 字種の標準プリント・バンドを装塡して1インチ当たり16行で印字すると毎分950行，また16字種（数字と特殊記号6文字）のみ印字を行うと毎分2,000行の印字速度が得られる．（資料コード：081821905）

●漢字複合プリンタ装置

漢字複合プリンタ装置は，多機能オフィス・ステーション MODEL 800 に接続可能な卓上型のプリンタ装置である．印字方式にワイヤ・ドット・マトリックス方式を採用，印字速度は漢字モード（24×24ドット）で60字/秒，英数カナ・モードで150字/秒である．その他連続帳表および単票やはがきが取り扱える水平インサータを装備，伝票マーク検出機能があり用紙裏面に印字された伝票マークによって，1,000 種類の伝票をプログラムで認識できる．

●日本語ワードプロセッサ UW 25

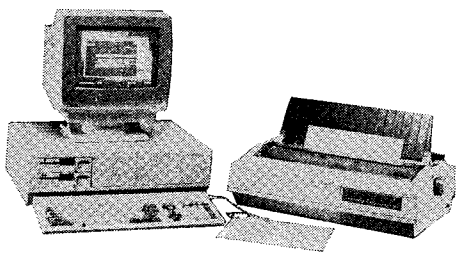
複合語を含む約8万語の辞書により，複文節変換が可能なカナ漢字変換方式を採用，高度な図形処理



機能とヒストグラム，パレート分析等の豊富なグラフ機能をもっている．

マルチフォント・システムにより15種類と豊富な文字サイズの印刷が可能である．また，重ね表示方式の採用により，ディスプレイ表示と印刷が完全に一致し，文書作成中に印刷イメージが確認できる．さらに，ラベル印刷，罫線ぬき印刷，領域指定印刷によるフォーマット用紙への印刷も可能である．　（資料コード：472815218）

●UP 10 E モデル 70

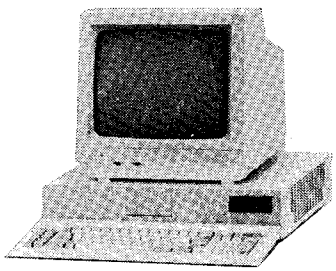


パーソナル・コンピュータ UP10E モデル 70 は，既存 UP 10 E モデル 40/50/60 の上位機種であり，マルチタスクが可能なコンカレント CP/M-86 を標準搭載，マルチウインドウ機能，文節単位でのローマ字・カナ漢字変換が可能で文法解析および学習機能により効率のよい日本語入力ができる．また，16色カラー，モノクロ 16 階調，最大 4 画面 VRAM，ドットスクロール，パレット機能等，強力なグラフ機能もある．

本体，キーボード，CRT ディスプレイ装置がそれぞれ分離したセパレート・タイプのパーソナル・コンピュータで，16 ビット CPU を採用，512 K バイトの主記憶装置を装備，メモリ ECC 機能付最大 1 M バイトまで，ディスク容量最大 40 MB と大きな拡張性をもっている．

（資料コード：481775415, 472775222）

●シリーズ 8 OA ワークステーション/新ワークステーション



（写真：新ワークステーション）

ＯＡワークステーションは，パーソナル・コンピュータ機能がサポートされ，豊富な汎用系（CP/M 系）ソフトが利用でき，ＯＡ機能が強化されている．また，ワード・プロセシング，ビジネス・グラフ等のパーソナル処理は，ＯＡワークステーション側で独立に処理可能なため，負荷分散ができ，使いやすさと高速処理が図れる．ワークステーション機能とパーソナル処理機能，ワークステーション機能と日本語ワード・プロセッサ機能の組合せで二つのジョブが同時に実行可能である．

新ワークステーションは，ＯＡワークステーションおよび MODEL 800 のワークステーションと整合性，統一性がとれ，かつエルゴノミクス・デザイン，JIS 第 1 水準漢字の ROM 化，小型軽量化が図られている．制御キーの位置および機能が改良され，TAB，BACKTAB，次項，レジューム・キーの位置および機能を改良し，ＯＡ機能として最新の流れにそったものとなっている．

ISSN 0289-6257

技　　報

UNIVAC TECHNOLOGY REVIEW

No. 8

発　行　日	昭和60年2月28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（大代表）
頒布価格	1,500 円
印　刷　所	三美印刷株式会社

本書の特色

①姓氏の収録数は約十三万数千項目、本邦最多。
②「表音編」「表記編」「解説編」の三冊セット。読みからでも、漢字からでも、検索はたやすい。
③姓氏の由来を33種の型に分類して全姓氏に示した。
④姓氏の読み方は、従来の辞書では調べられなかったが、本書で、姓氏特有の漢字や読み方がすべてわかる。
例 姓氏特有の漢字(十・丁・才・七など)や、古語(郡・評・私部・正月一日など)や熟字訓(栗花落・月見里など)などのほか、方言や音転など国語研究の資料となるものが山とある。
⑤「解説編」は、平易な記述で、だれにも興味ぶかい。
⑥古代の氏から現代の苗字まで総合的に解説。
⑦地名・家紋との関係も述べ、姓氏の本質を解明。

日本人の誇るべき文化財——姓氏

○名前は一代限りであるが、姓氏は子々孫々に伝承されるもので、祖先からうけ、子孫につぐ、姓氏にはだれもが関心がある。
○古代からの姓、明治になってできたものにもその由来がある。その由来をすべての姓氏について明らかにし、祖先をたどる手がかりとした。
○十年間の全国調査と集録により、はじめて日本姓氏の全貌が明らかになった。丹羽基二氏と日本ユニバック㈱との共同研究の成果。
○姓氏・系譜研究にはもちろん、国語学、言語学、民俗学、歴史学、地理学、紋章学、その他多くの分野に重要な基礎資料を提供する。
○学校・図書館、研究所のほか電話局、役所、会社、銀行、保険やDM業界、印鑑屋、広告社のネーミング・カード等まで、本書の利用は無限。

日本姓氏大辞典

丹羽基二／日本ユニバック㈱編

昭和60年3月20日刊行

表音編・表記編・解説編 全三冊

●816ページ ●832ページ ●416ページ

造本体裁＝菊判・セット函入・上製

発売記念特別定価 18,000円
（昭和60年12月末日まで）

定　価＝22,000円

角川書店

〒102 東京都千代田区富士見2-13
電話 03(238)8521営業部

東京大学名誉教授 **坂本太郎**

丹羽氏の不退転の研究心には頭のさがる思いがするが、氏の研究の強みは、ひとり苗字に限らない。地名と家紋とにも及んでいることである。当然、この三者は相互に密接な関係をもつものだから、三者に通暁する人によって、おのおのの研究は完全となるのである。

誰しも、この書をみれば、まず自分の苗字が何に由来するのか、どこに分布するのかを知って、先祖をしのぶよすがにすることであろう。そしてまた、難語難読の苗字のいかに多いかを知って、漢字と深くかかわり合った日本文化の融通無碍な性格に感慨を催すであろう。研究者はもとより、一般読書人の座右に欠くことのできない書物であると私は信ずる。

元東京大学教授 史料編纂所所長 **竹内理三**

今日までの日本人の姓氏は、本来は発音をもととしながらも、それを漢字で表示することになっているために、同じ漢字を用いながらも、人によってその読みがちがうことが多い。漢字を音よみ、訓よみ、意義よみ、組み合わせよみなど、自由自在に駆使して万葉集を編集した日本人の才能はここにも発揮されている。たとえば、「神」の漢字のよみは40とおりほどにも使われていて、姓氏をよみわけることを困難にしている。

著者の指摘するように、姓氏は、国語学・民俗学・地理学・歴史学の分野にも深くかかわりをもつ。姓氏は、これらの分野にも、重要な資料であるとする著者の主張に心から賛意を表したい。

国学院大学名誉教授
国学院大学栃木短期大学学長 **樋口清之**

本書は13万種もの姓氏を集めることに成功したから、まさに世界的な成果だといえる。さらにそれを30数種の「型」に分類して、分類姓氏辞典になっている。その中で私らの大いに注目したい点は、その姓氏の80パーセントが時代と関わりながら地名と結びついている、という事実にある。根拠をもってこれを確定し得た業績は、大いに評価すべき点だと私らは著者の労を多としたい。

また著者は、日本人が持つ姓氏を日本人の誇るべき文化財として長い伝統生活の中に生まれて来たものと把えたい、と主張する。たしかに珍名、難名と各自が思うのは勝手だが、その評価には今の人の教養の偏りや低下もあることを反省しなければならないし、一々の由緒を解き進めてその存在の必然性にたどりついたとき、確実な歴史の証人に遭い、証拠を把えた悦びと安心を得るのである。姓氏・苗字とはそんな意義を持っているものだと、私らは改めて本書によって教えられる。