技報

UNIVAC TECHNOLOGY REVIEW

1983 年 8 月　第 5 号

日本ユニバック

R.C. Heinselman の**分散システムにおける待ち行列モデルのネットワーク**は，分散処理システムのパフォーマンスを評価するモデルについて述べている．このモデルを使うと，システム設計や分析をする際にシステム設計上の決め手となる調査が応答を見ながら行える利点がある．このモデルで用いている待ち行列ネットワークを評価する分析技法は新しいもので，モデル化の結果得られるものは，利用者が記述した分散処理システムに関するパフォーマンス推定値（の集合）である。計算されるパラメタには，トランザクションのエンド・ツー・エンドの平均応答時間，ハードウェア装置の利用率がある．

A. Rosenthal, D. Reiner らの**ネットワーク・モデル上の関係型ビューの問い合せ**は，CODA-SYL モデル，あるいは，実体-関連モデルにもとづいて生成されたデータベースに対する関係型問い合せを支援する設計について述べている．ここに述べるアーキテクチャは通常の方法とは異なり，純粋な関係型システムのビュー機能によってモデル間の問い合せ翻訳を処理するものである．このシステムでは，CODASYL データベースの意味を表現する関係型スキーマは一貫性制約を持った関係型ビューの集まりとされる．また，生成した関係型スキーマに対する各 SQL 問い合せは，本質的には CODASYL レコードである“アクセス・リレーション”に対する SQL 問い合せに翻訳される．

J.J. Hart の**デザインとコードにおけるウォークスルーの効果**は，Sperry 社における Programmers Advanced Debugging System (PADS 1100) の開発時に行った設計とコードのウォークスルーにおいて，観察された効果について報告している．とくに，ウォークスルーによって認められたいくつかの非統計的な効果を論じ，一つの非常に有意義な定量的効果が示されている．

T.N. Turba の**高水準言語の下方拡張機能**は，使用者が定義したコード列をコンパイラの生成コードに統合する機能について，一般的な考察を行っている．この機能は PLUS (Programming Language for Univac Systems) において実現された機能にもとづくもので，このインライン・システムの経験を通して，高水準言語を下位方向に拡張する機能が実用的に，さらに効果的かつ効率よくコンパイラ生成コードに統合できると報告している．

斎藤哲郎の**ソフトウェア開発環境“CASE”**は，ソフトウェアのライフ・サイクルを全般にわたって使用できるソフトウェア開発保守のための抱括的な環境である，CASE とこれに含まれるツールに焦点を当てて，その概要と効果および日本語処理について報告している．

これらのツールは，Sperry 社で開発され，同社ならびに日本ユニバック社内での EXCE を含めた OS 1100 ソフトウェアの開発・保守に使用され，生産性および品質の向上に寄与している．

C.H. Chen の**テスタビリティを目的としたVLSI設計論**は，論理設計の方法論の立場から研究したものである．これには，レベル・センシティブ・スキャン設計 (LSSD) への構造的アプローチ，スキャン・セット技法のさまざまな適用，アド・ホックな設計規則が含まれている．すなわち本稿の力点は，動的なテストを可能にするより，一般的な基本的設計規則および設計原理を明らかにすることにおいている．そして，論理回路のテストのライフ・サイクルについても触れつつ，VLSI のテストの特徴を分析している．

なお，テスト可能な論理に対する設計上の制約条件はブール式で表現してある．

LSI チップ上のゲート，モジュール上の LSI チップ，基板上のモジュールなどの位置決め問題はすべて基本的に同型である．

A.M. Patel の**配置改良アルゴリズムにおける配線可能性の測度**は，配置アルゴリズムの概要と，このアルゴリズムのテスト結果を報告している．

全配線長を最小化する従来のアルゴリズムでは，必ずしも配線可能な配置が得られない．ここで述べているアルゴリズムでは，初期配置を構成的に生成し，全配線長と配線可能性指標の両方を目的関数として，配置状況を反復改良し最適化している．

藤野　勉の**座標変換差分法による連続体解析**は，筆者が日本ユニバック(株)に入社以来研究した結果をまとめたものである．任意形状の境界を持つ連続体の解析問題において極座標系あるいは楕円座標系のような直交曲線座標系が利用できる場合，座標変換による差分法が便利であると報告している．また，直交曲線格子系が確立していない他の系では，任意節点配置差分法か有限要素法などにより，この系を設定することが必要であると述べている．

報告 分散システムにおける待ち行列モデルのネットワーク

A Network of Queues Model for Distributed Systems

R. C. Heinselman

要 約 本稿は，分散処理システムのパフォーマンスを評価するモデルについて述べたものである．このモデルを使うとシステム設計や分析をする際，システム設計上の決め手となる調査が応答を見ながら行えることとなる．このモデルは，モデル分析の用語に不案内な利用者にも容易に使えるよう，わかりやすいインタフェースを備えている．このモデルで用いている待ち行列ネットワークを評価する分析の技法は，新しいものである．

このモデルでは，分散処理システムすなわち（ネットワーク）を通信網で相互接続されたコンピュータ・システム(すなわち節点の集まり)とみなしている．ネットワークの中でデータベースは分割され，複数の節点に貯えられている．個々の節点でのトランザクション処理は，トランザクションを多数のクラスに分けたセントラル・サーバ待ち行列モデルとしてモデル化されている．ネットワーク中の節点間におけるトランザクションの流れは，データベース駆動型である．すなわち，個々のトランザクションは，始節点から必要とするデータベース・セグメントのある節点へ流れる．ネットワーク内のトランザクションの流れにつれて，トランザクションはオペレーティング・システムの支援，データベース管理システムの支援，通信ハンドラ，業務プログラムおよびトランザクション管理といった種類の資源を使用する．利用者が分散処理システムを記述するには対話型の一連の質問に答えるだけでよく，これでネットワークの構造や負荷特性を決めるために必要な情報が集められる．ネットワークの構造の指定は，あらかじめ基本的な構成節点を記述しておくことによって行う．トランザクションの定義は，トランザクションが低水準で一般的なものであるか，または高水準の DDP トランザクションであるかを見て行う．利用者が与えるシステムの記述を基本として多重クラスの待ち行列のネットワークを作り，このネットワークを積形の分析手法により分析する．モデル化の結果得られるものは，利用者が記述した分散処理システムに関するパフォーマンス推定値（の集合）である．計算されるパラメタには，トランザクションのエンド・ツー・エンドでの平均応答時間，ハードウェア装置の利用率がある．

Abstract This paper describes a model that estimates performance for distributed computer systems. The model provides the systems designer or systems analyst with a responsive tool for performing high level system design studies. The model has a user-friendly interface designed for the user who is not familiar with the terminology of analytic modeling. The model is implemented using state-of-the-art analytic techniques for evaluating queueing networks.

The model views a distributed computer system (network) as a collection of computer systems (nodes) interconnected by a communications subnetwork. The databases in the network are segmented and the segments are stored at nodes throughout the network. The processing of transactions at each node is modeled as a central-server queueing model with many transaction classes. The flow of transactions between nodes in the network is database-driven with each transaction flowing from its origin node in the network to the nodes having the database segments it requires. During its travels in the network a transaction consumes resources attributable to sources such as operating system support, database management system support, communications handlers, applications code, and transaction management. A user describes a distributed computer system by responding to a series of interactive solicitations which gather the information needed to establish the network structure and workload characteristics. The network structure is specified using predefined descrip-

tions for the basic node components; transactions are defined in terms of either low level general transactions or high level DDP transactions. The system description provided by the user is used as the basis for constructing a multiclass network of queues that is analyzed using product form analytic techniques. The end product of a modeling session is a set of performance predictions for the distributed computer system described by the user. The parameters computed include average end-to-end response times for transactions and utilizations for hardware devices.

1. はじめに

本稿では，分散処理システムのパフォーマンスを評価する分析モデルについて述べる．分散処理とは，複数の離れて存在するコンピュータの間に利用者の業務を機能的に分散し，協力して処理を行うことと定義する．この定義は非常に一般的で，通常行われている他の定義の趣旨を取り込んだものとなっている[8]．この定義は，利用者業務が2台以上のコンピュータ資源を必要とすることを暗に意味している．したがって，数台のコンピュータにまたがる業務内通信を実現する必要がある．さらに，データベースの概念についても同様である．データベースとは，ファイル管理システムかデータベース管理システムを介してアクセスされるデータの論理的な集まりとみてよい．分散処理システムのコンピュータの中には物理的にデータベースを持たないものがあってもよく，単一の論理的データベースを物理的に分割したセグメントは，複数のコンピュータにまたがってそれぞれ分散されたり，あるいは複製されてもよい．

このモデルを使うとシステム設計やシステム分析を行うに際し，システムが構造や負荷の変化に対し，どのくらい敏感かを容易に評価できる．これによって，設計者は設計上の要件変更の影響を視覚化して評価でき，設計をやり直さずにすむ．このモデルによって，任意の分散処理システムは合理的に表現できる．また，このモデルではトランザクションは独立であって，データベース・セグメントの複製は作らない．以下ではモデルをいくつかの視点から論じる．第1に根底にある概念モデル，第2にネットワークの分析を可能とする計算モデル，第3に計算モデルに対する検証と妥当性確認の手続きについて論じ，最後に批判的見地から概念モデルと計算モデルを見て，分散処理システムのモデル化に適したものであるかどうかを評価する．

2. 概念モデル

分散処理システムは，ここでは**ネットワーク**とも言い，コンピュータ・システムが通信網によって相互接続された集まりを指し，個々のコンピュータ・システムを**節点**と言う．ネットワークをわたり歩く処理対象を**トランザクション**と言う．トランザクションはネットワークにはいって，アクセスすべきデータベース・セグメントを持つ節点に至り，ネットワークから抜け出す．トランザクションは，どの節点からでもネットワークへはいることができる．すべてのトランザクションはただ一つの決まった節点に到着する必要はなく，またトランザクションがすべての節点に到着する必要もない．トランザクションが節点に至ると，CPU 命令と入出力操作と言う形で資源を使用する．この分散処理システムの概念モデルの根底を形作る定義，制約および仮定は次のとおりである(図1)．

1) ネットワーク・データベースは，データベース・セグメントを分割したり，複製して分散してよい．(後者は前者の拡張とみなされる．)
2) **始節点**とは，あるトランザクションがネットワークにはいる節点である．
3) **終節点**とは，トランザクションがデータベース・セグメントをアクセスする節点である．始節点が同時に終節点である場合もある．トランザクションの使用できる終節

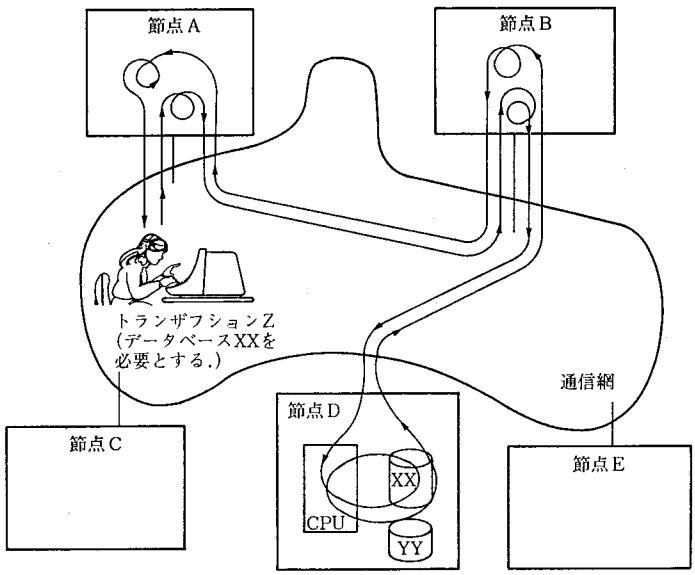


図 1 分散処理システムの概念モデル

Fig. 1 Conceptual model of a distributed computer system

点は，ただ1個である．

4) **中間節点**とは，トランザクションがその始節点から終節点へ至る途上で経由する節点である．

5) 複数の節点を経由するトランザクションの経路は，始節点から中間節点を経て終節点へ至る路であり，その後に同じ路を中間節点を経て始節点へさかのぼる（バックトラック）経路があってもよい．

6) トランザクションは，訪れる任意のあるいはすべての節点において，CPU 資源と入出力資源を使用してよい．

7) すべてのトランザクションの優先順位は等しい．

8) どのトランザクションも他のトランザクションから独立である．トランザクションの間に依存関係はない．

9) 一つのトランザクションは，複数の資源を同時に占有できない．

10) “メッセージ”とは，トランザクションが節点から節点へ移るとき，節点間で転送される情報の全体である．節点間通信の調整をするために転送される情報は，通常はメッセージに含めない．

2.1 トランザクション

トランザクションは，外部からネットワークにくる仕事の一つの単位であり，トランザクションの例としては，一括処理の一つのラン，タイム・シェアリングにおける命令，データベースの問い合せや更新がある．トランザクションを用いて負荷を定義する．負荷とは，最小の負荷の単位である．また，負荷はこのモデルの出力報告で，ネットワーク・パフォーマンスを求める対象である．トランザクションは，節点にある利用者の機器からネットワークにはいり，数々の節点をめぐり，ある節点にある利用者の機器を通ってネットワークを抜け出す．トランザクションの終節点は，そのトランザクションが必要とするデータベース・セグメントの位置によって決まる．トランザクションがある節点から次のトランザクションは，データベースの必要な項目をすべて読み，データベースの項目を

節点へいく路は，通信網の構造によって決まる．トランザクションがネットワークをめぐり，ネットワークの機能が実行される際には，トランザクションは CPU 資源と入出力資源を使用する．トランザクションには資源使用の構造化した様式（図2）があり，また CPU 資源と入出力資源を使って，データベース関連以外の業務機能と必要とされるオペレーティング・システムの支援機能を実行する．

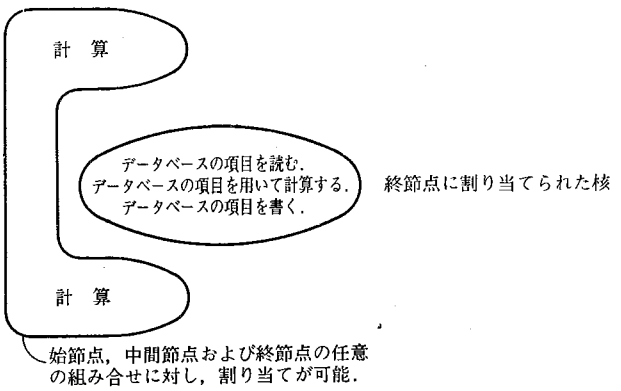


図 2　トランザクションにおける事象の構造

Fig. 2　Structure of events in a transaction

用いて再度計算し，データベースのある項目を書き，次いでデータベース関連以外のあらゆる事後処理としての計算を行う．データベースの項目を読み，参照し，書くトランザクションの部分を**核**（kernel）と言う．核の処理は，トランザクションの終節点でのみ実行される．計算の領域のうち核の両側の部分は，トランザクションの始節点，中間節点，または終節点で任意の組み合せで実行される．**トランザクション型**は，似た特性を持つトランザクションを記述するのに用いる抽象的対象である．任意のトランザクションは，トランザクション型の一つの**具体例**（instance）である．　トランザクション型を決めるには，次の事項を指定する．

1)　始節点
2)　個々の始節点への到着率
3)　計算特性（すなわち，利用者業務および必要なオペレーティング・システムの支援に必要な CPU 命令数と入出力操作の数とサイズ．ただし，入出力ハンドラ，通信インタフェースおよび立ち上がりのディスパッチは除く．）
   - 始節点における特性
   - 中間節点における特性
   - 終節点における特性（核の計算を含む）
4)　核のデータベース操作（データ操作言語の命令数と入出力操作数）
5)　核のデータベース操作を行うために必要なデータベース・セグメントの名前
6)　必要とするデータベース・セグメントに対する入出力操作の振り分けとサイズ
7)　メッセージ特性(トランザクションが流れるときに節点間で受け渡される情報の量)

2.2 節　　点

節点は，ハードウェア複合体とソフトウェア複合体から構成される．ハードウェア複合体の要素は，CPU 複合体と入出力複合体である．　ソフトウェア複合体は，その節点のオペレーティング・システム，ネットワーク・オペレーティング・システムの部分，データベース管理システムおよび通信インタフェースの集合から構成される．

節点によってサービスされるトランザクションは，多重タスク方式の同期式入出力モデルに従い，その結果トランザクションの入出力操作はトランザクションの CPU 処理時間帯内に一様にばらまかれる．任意の時点において，任意のトランザクションは1台の CPU，または1台の入出力装置だけを占有できる．しかし，トランザクションが異なれば CPU と入出力装置を並行して占有してもよい．

CPU 複合体は，1台以上の CPU で構成される．CPU は密結合，または疎結合である．CPU が利用できる記憶容量は，無制限にあると仮定する．一つのトランザクションが1台の CPU に割り当てられる時間は，連続する二つの入出力操作の間にある CPU 処理時間全体である．平均的命令を実行するためにかかる時間によって，CPU が仕事を実行する能力を表現する．入出力複合体はストリングとしてまとめたチャネル，制御装置および機器群で構成される．**ストリング**とはチャネル，制御装置および機器の集まりであり，全体としてビジーまたはアイドルとなる一つの単位として動作する．入出力ストリングの仕事の能力は，一つの入出力操作に対するデータのアクセスと移送に要する時間で表される．ストリング当たりの記憶容量は，制限がないと仮定する．入出力複合体は，データベース・セグメントの貯蔵庫として働く．セグメントごとに特定の入出力ストリングが割り当てられる．一つのストリングに割り当てられるセグメント数は1とは限らないが，一つのセグメントが複数のストリングに割り当てられることはない．オペレーティング・システムとデータベースに関係のない利用者の入出力は，すべて単一のストリングに振り向けられる．

一つの節点のソフトウェア複合体は，その節点のオペレーティング・システム，その節点のネットワーク・オペレーティング・システムのセグメント，データベース管理システムおよび，通信ハンドラの集合といった複数のモジュールから構成される．このソフトウェアによって可能となる基本的なサービスにより，トランザクションは業務の機能を実行する．このソフトウェアは階層状にモデル化できる（図3）．

ソフトウェア・モデルを説明するには，図3に従って述べるのが，おそらく最も簡明であろう．トランザクションが節点に到着したものとする．このトランザクションが，その節点の利用者の機器からきたか，それとも他の節点からきたかは重要でない．到着後，トランザクションはネットワーク・アクセス処理と通信インタフェースの層を通り抜けねばならない．オペレーティング・システムはトランザクションを待ち行列に入れ，サービス

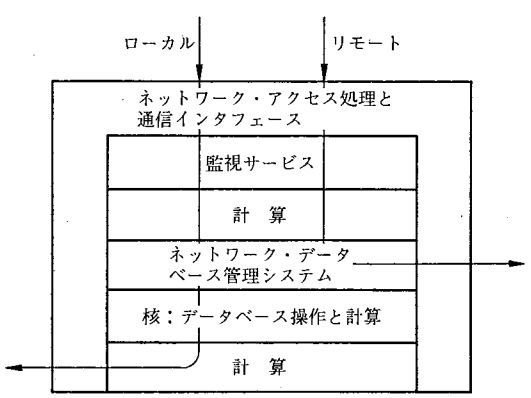


図 3 節点におけるソフトウェアの階層

Fig. 3 Software layering at a node

のスケジュールをする．トランザクションが，最初の計算処理を始めるとき，オペレーティング・システムのサービスによって入出力処理，ディスパッチ，スワッピングが可能となる．この節点での最初の計算処理が完了すると，トランザクションはネットワーク・データベース管理システムに問い合せて，状態が“ローカル”か“リモート”かを決める．“ローカル”の場合，このトランザクションの核の操作に必要なデータベース・セグメントは，その節点に存在し，核の処理を行うために別の節点へ移る必要はない．核の処理のサービスは，オペレーティング・システムとデータベース管理システムによって行われる．

核の処理が完了すると，最後の計算処理がすべて完了する．トランザクションは利用者の機器へ戻るか，あるいはネットワーク・アクセス処理と通信インタフェースの層を通って現在の節点へくる前にいたところの節点へ戻る．“リモート”の場合，このトランザクションの核の処理に必要なデータベース・セグメントはその節点になく，トランザクションは終節点に至る路の中の次の節点へ渡される．このような動作を行うには，ネットワーク・アクセス処理と通信インタフェースの層を通る必要がある．次の節点で，トランザクションはこのサイクルを再度始める．どのトランザクションもその終節点においては“ローカル”であり，そこでの処理を完了後，帰路につき，始節点への路を逆にたどる．

ディスパッチとかスワッピングといったオペレーティング・システムのサービスは，トランザクションに依存したものなので個々のトランザクションの計算処理の定義の中で説明されねばならない．一方，入出力処理や立ち上がりのディスパッチといったオペレーティング・システムのサービスはきわめて普遍的なので，この種のサービスは節点の定義に含まれる．同じ理由で，ネットワーク・アクセス処理と通信インタフェースの層のサービスとデータベース管理システムのサービスは，個々の節点の定義に含まれる．サービスを遂行するために，ソフトウェアの個々の階層が必要とする CPU と入出力の能力の大きさは異なる．この大きさは層ごとに CPU 命令数，入出力操作の数と大きさにより決められる．トランザクションの記述によって，計算モデルがトランザクションにこの種のサービスをする上で必要な情報が得られる．

2.3 通 信 網

通信網は，分散処理システムにおけるデータ転送機能を遂行する．この網は，利用者の機器とコンピュータ・システム間でデータを移動する．概念モデルでは通信網の構造は問わず，たとえばリング，ループ，バス，星型，網目状などすべて許される．通信網の定義は静的であって，相互接続の動的変更を支援していない．相互接続の仕事をする能力は，その相互接続を通して１単位の交換（データ）を転送するに要する平均時間によって表される．

3. 計 算 モ デ ル

計算モデルによって概念モデルは克明に表せると同時に，ネットワーク・パフォーマンスを素早く評価できる．このモデルは，分散処理システムの設計を支援するために作られたものである．したがって，このモデルはパフォーマンス予測を報告することに加えて，ボトルネックに対して自動的に対応する． ボトルネックとは，利用率が 99.9 パーセントより高い構成要素を指すものとする．

ボトルネックが検出されると，モデルはボトルネックとなった構成要素に仕事の能力を付与し，この結果としてのネットワークのパフォーマンスを予測しようとする．数々の能力の付与を行ってもボトルネックを取り除けないならば，モデルは処理を終了する．計算

モデルを作るには，待ち行列分析の多重クラスの積形の分析ネットワーク[3],[15]を使う．その目的は，概念モデルを表すために形成される待ち行列ネットワークのパフォーマンスを評価することにある．このモデルのプログラムは，モジュール化されていて計算アルゴリズムを容易に改変できる．主要モジュールのアクティビティについては，以下に論じる．

3.1 利用者インタフェース

利用者インタフェースによって，利用者は目標とする分散処理システムを記述でき，モデルの提供するパフォーマンス予想を検分できる．記述する情報は，すべて会話形式で利用者から引き出されたものである．個々の節点は，あらかじめ定義された表から選んだ構成要素により記述される．構成要素は，あらかじめ定義されており，利用者は，CPU，入出力ストリングおよびオペレーティング・システムの細部を知っている必要はない．利用者が通信網を定義するには，始節点として働くすべての節点を終節点として働くすべての節点へ結ぶ路を列挙する．始節点から終節点へは，ただ一つの路がありうる．計算モデルでは，トランザクションを二つの観点から見る．まず**一般トランザクション**は，概念モデルにおける低水準トランザクションで多様なトランザクションを表す．**DDP トランザクション**は，ジョブ転送とかファイル転送といった一つ以上の一般トランザクションから成る高水準トランザクションである．DDP トランザクションから一般トランザクションへの写像は自動的に行われ，利用者の介入は不要である．DDP トランザクションに対して出てくる報告のエントリは，必要となった一般トランザクションのエントリの集まりを反映したものである．

出力の報告では，ネットワーク分析から得られるパフォーマンス予想を明晰で簡潔に表示する（報告の抜粋を表1(a),(b)，および表2(a),(b),(c)に示す）．

表 1 概 要 報 告

Table 1 Summary output reports

(a) トランザクションに依存しない分析

構 成 要 素	利 用 度	トランザクション総数*
節点1 CPU	5.19%	.05
節点1 入出力機器	29.24%	.41
節点2 CPU	10.53%	.12
節点2 入出力機器	85.12%	5.72
節点3 CPU	5.19%	.05
節点3 入出力機器	29.24%	.41
節点1 〈−〉 節点2	4.19%	.04
節点2 〈−〉 節点3	4.19%	.04

* 構成要素ごとの総トランザクションの平均

(b) 構成要素に依存しない分析

トランザクション	遅 延 (秒)	装置におけるトランザクション総数*
節点1からのトランザクション1	2.54	.04
節点3からのトランザクション1	2.54	.04
節点1からのトランザクション2	1.57	.13
節点3からのトランザクション2	1.57	.13
節点1からのトランザクション3	1.57	.00
節点3からのトランザクション3	1.57	.00
節点2からのトランザクション4	9.12	.82
節点2からのトランザクション4	10.75	1.23

* トランザクションごとのすべての機器，すべてのフェーズの総トランザクションの平均

表 2　詳 細 報 告

Table 2　Detailed output reports

(a)　節点1からのトランザクション1についての分析

(始節点と中間節点での業務＋ネットワーク・データベース管理システム＋ネットワーク・インタフェース)

構 成 要 素	装置利用度	トランザクション総数	遅延 (秒)
節点1 CPU	.22%	.00	.14
節点1 入出力機器	.82%	.01	.70
節点1 端末装置	.00%	.00	.21
節点2 CPU	.02%	.00	.02
節点2 入出力機器	.08%	.01	.33
節点3 CUP	.02%	.00	.01
節点3 入出力機器	.08%	.00	.07
節点1 〈−〉 節点2	.01%	.00	.00
節点2 〈−〉 節点3	.01%	.00	.00

(b)　節点1からのトランザクション1についての分析

(データベース管理システム＋終節点での業務)

構 成 要 素	装置利用度	トランザクション総数	遅延 (秒)
節点1 CPU	.11%	.00	.07
節点1 入出力	.47%	.01	.40
節点3 CPU	.01%	.00	.01
節点3 入出力	.05%	.00	.04

(c)　節点1からのトランザクション1についての分析

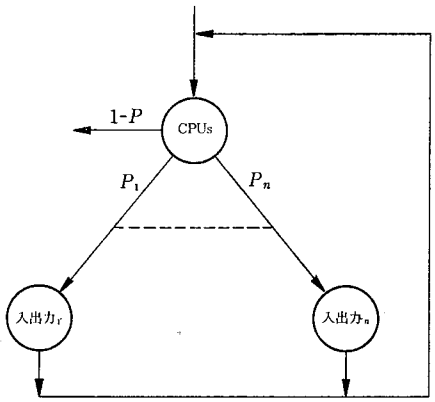
(ネットワーク・インタフェース)

構 成 要 素	装置利用度	トランザクション総数	遅延 (秒)
節点1 CPU	.01%	.00	.01
節点1 入出力機器	.06%	.00	.05
節点1 端末装置	.00%	.00	.21
節点2 CPU	.01%	.00	.01
節点2 入出力機器	.06%	.00	.24
節点1 〈−〉 節点2	.01%	.00	.00
節点2 〈−〉 節点3	.01%	.00	.00

次節で示すように待ち行列ネットワーク分析は，個々のトランザクションを別々のいくつかのフェーズを持つものとしてモデル化している．したがって，出力の報告にはトランザクション・フェーズの概念がある．報告に示されている実験値は平均である．オンラインとオフライン両方の報告が作られている．オンライン報告は概要であって，各ハードウェア装置（たとえば CPU）ごとにトランザクションに依存しない利用度およびトランザクション総数を示している．また，各トランザクションごとのエンド・ツー・エンドの応答時間と，装置の総数も示している．オフライン報告には，概要と詳細がある．概要報告の内容は，オンライン報告に類似している．詳細報告によって，トランザクション・フェーズに対する1組の実験値がわかる．この詳細報告では，各装置ごとに各フェーズの利用度，応答時間および機器におけるトランザクション総数が示されている．

3.2　待ち行列ネットワーク分析

概念モデルを実現する待ち行列ネットワークの表現は，利用者からデータ収集フェーズに得られた情報を使って作られ，分析される．待ち行列のネットワークは開放されている．つまり，トランザクションの流れが，ネットワークに出入りする際に，いつどこででも，その数に限度を設けない．個々の節点は，一般化した開放型のセントラル・サーバ・モデル



$P=\sum_{i=1}^{n} P_i$ ただし
$P_i=\sum P_{DBk}$ とし，データベース・セグメント k がストリング i にあるものとする．
また P_{DBk} は，入出力アクセスを行う場合に，データベース・セグメント k にアクセスする確率を表す．

図 4 節点の基本的表現

Fig. 4 Basic node representation

として表現できよう．通信網の相互接続マトリックスによって節点を接続できる（図4）．

トランザクションを表すには，多重のクラスの開放型チェインを使う．一つの節点に到着する各トランザクション型ごとに，一つのチェインがある．一つのトランザクションが起こすクラス（フェーズ）の変化を考える際，概念モデルのソフトウェア図を考える必要がある．どのトランザクションもクラスaで始まるが，これは始節点と中間節点における計算処理，およびそれと関連したオペレーティング・システム処理を表すものである．このクラスは，また一つのトランザクションに対するネットワーク・データベース管理システム，ネットワーク・アクセスおよび通信インタフェース処理をも表す．トランザクションが“ローカル”になると，クラスはa1に遷移し，終節点の核の計算とデータベース処理，および関連したオペレーティング・システム処理を表す．始節点と終節点が異なるトランザクションに対するネットワーク・アクセスと通信インタフェース処理を表すため，クラスa2への最後の遷移がある．一つのチェインの中でのクラスの変化を使って，一つのトランザクションのフェーズごとの資源の利用度と待ち行列の状態が報告される．

各節点のCPU複合体は，CPUに対するプロセッサ共有サーバを使って作られる．CPUが密結合多重プロセッサの場合，待ち行列ネットワークの基本的分析は，単一CPUのサービス時間を適当に高速化して実行する．しかしながら報告が作られたとき，多重サーバの待ち行列理論にもとづいた近似を行って，CPUに関連するエントリを洗練する．疎結合多重プロセッサの場合は，CPUに対し多重プロセッサ共有サーバを使い，個々のCPUを等しい確率で選ぶ．プロセッサ共有サーバを使うと，個々のクラス（トランザクション・フェーズ）のサービス時間には多様な分布を選択できる[3]（サービス時間の分布は，有理Laplace変換を持たなければならない．普通のサービスの分布は，ほとんどすべてこの条件を満たしている）．

各節点の入出力複合体は，出力ストリングごとに先着順（FCFS: First-Come-First-Served）サーバを使って作られる．入出力ストリングは，利用者がトランザクションに割り当てたデータベース・セグメントの使用法に従って選択する．しかし，入出力ストリングの内部では，入出力サーバの選択は等しい確率で行う．先着順（FCFS）サーバでは，すべ

てのクラス（トランザクション・フェーズ）に対して同一の指数分布のサービス率を仮定する[3]．しかし，個々のクラスがサーバをループする回数が違うため，各クラスに積み上げた全サービス時間は異なりうる(結果の K アーラン分布の平均は，ループ・サービス時間の平均にループした回数を乗じた値に等しい[21])．パフォーマンス予測は，分布と独立に全サービス時間の平均を使って計算するので[13]，個々のトランザクション・フェーズに与えられる入出力パフォーマンスは，その入出力操作の数とサイズに対応するものとなる．

ネットワークのレベルでは，ネットワークの節点間通信による遅延は先着順 (FCFS) サーバによって表される．遅延が0ではない各節点間リンクに対して，一つのサーバが使われる．入出力複合体の場合と同様に，トランザクション・フェーズごとに節点間通信の遅延を与えることができる．これは，その節点間メッセージのサイズに対応するものである．

利用者のワーク・ステーションと，その節点への通信リンクは遅延サーバによって表される．この型のサーバによってワーク・ステーションを必要とする任意のトランザクションは，遅延サーバを利用できる．遅延サーバによって，個々のクラス（トランザクション・フェーズ）のサービス時間には多様な分布が選択できる．

4. 検証と妥当性の確認

モデルを使う前に，利用者はその出力の正しさを確認しなければならない．本稿で考察されたようなモデルでは，システムを広範囲に考える必要があるので，これは大仕事となる．しかし，この場合でも確信を持つことはできる．

確信には二つの面があり，一つは検証，もう一つは妥当性の確認である．検証では，モデルの計算が正確であることを調べる．このプロセスでは結果が有意であることを確かめるわけではなく，アルゴリズムの正しさを調べるにとどまる．妥当性の確認においては，与えられた場合におけるモデルの出力をモデルとは独立に得られる同様の観測値との関係において，有意であることを調べる．較正，すなわち特定のシステム設計にモデルを合せるために内部パラメタを調整することは行わなかった（この種の調整はモデルのプログラムを変更することとなる）．実際に行った検証と妥当性の確認では，結果を“改良”するための負荷の調整をしていない．通常較正と言われるこの種の調整は，モデルの正確さを損うだけで改良にならない．モデルの正しさを確認するため，二つの実験をしてみよう．この実験では，分散処理システムと同様に，集中コンピュータ・システムを表すモデルを利用する．表3(a),(b)にこれらの実験の結果を示す．

最初の実験においては，実際のシステム・データと，閉鎖型の（トランザクション総数を有限と想定した）セントラル・サーバ・モデルのデータを，C. A. Rose の文献[19]からとった．多重クラスの積形を用いた待ち行列ネットワーク分析を使う開放型セントラル・サーバ・モデルのデータを発生するには，SNAP パッケージ[14],[15]を使用した．開放型と閉鎖型とで，装置の利用度数と利用率はほぼ同じである．けれども閉鎖型の場合は，トランザクション総数が小さい（4トランザクション）ので[4]，トランザクション総数と遅延は同じではない．計算モデルを使ってシステムをモデル化するときには，一つは単一節点として，もう一つは節点が異なると入出力装置が異なる多節点ネットワークとしてモデル化した．単一節点と多節点との間で表現が一致していると，節点間の遷移アルゴリズムを部分的に検証する上で役に立つ．SNAPの結果と単一節点の場合の結果が一致していることは，節点内アルゴリズムの正当性を部分的に証明する上で役に立つ．Rose は，彼の閉鎖型セントラル・サーバ・モデルが，実際のシステムに酷似していると考えた[19]．し

表 3 実験結果
Table 3 Experimental results

(a) 実験1の結果

	実システム	閉鎖型セントラル・サーバ・モデル (Rose)	開放型セントラル・サーバ・モデル (SNAP)	モデル (1節点)	モデル (5節点)
利用率 (%):					
CPU	89.30	86.50	86.45	86.51	86.51
CH1	44.90	51.70	51.70	51.68	51.68
CH2	37.00	38.00	38.02	38.02	38.02
CH3	32.00	35.50	35.50	35.50	35.50
CH4	47.80	51.80	51.96	51.96	51.96
平均応答時間 (秒):					
タイム・シェアリング	.081 (予測値)	NA	.294	.29	.29
一括処理	.091 (予測値)	NA	.144	.14	.14
システム平均トランザクション総数:					
タイム・シェアリング	.8	1	2.7	2.69	2.69
一括処理	3.8	4	7.0	7.04	7.04

(b) 実験2の結果

	A (AIET/2)*	モデル (AIET/2)	モデル (2サーバ CPU)	B (2サーバ CPU)
利用率 (%):				
CPU	50.1	50.13	50.13	50.4
CH 1-CH 12	10.9	11.15	11.15	11.5
平均応答時間 (秒):				
T1	96.15	96.32	101.40	101.11
T2	116.48	116.34	135.17	135.27
T3	23.30	23.72	27.49	27.05
T4	4.87	4.84	4.96	4.90
システム平均トランザクション総数:				
T1	.56	.56	.59	.59
T2	.16	.16	.19	.20
T3	1.72	1.75	2.02	1.99
T4	.04	.04	.04	.04

* AIET (Average Instruction Execution Time): 命令を実行する平均時間 AIET/2 は，AIET が2倍の速さで行われることを示す．

たがって，計算モデルでの実験値をこのデータと比較すると，このモデルが集中システムを表現する能力を持つことの妥当性を何がしか確かめられたことになる．2番目の実験では，密結合多重プロセッサの CPU 複合体を持つ集中システムの計算モデル表現を，二つの別の分析モデルによる表現と比べている．モデルAでは，CPU は多重プロセッサではない．したがって，このモデルでは密結合多重プロセッサのモデル化の際にサービス時間を短縮している．計算モデルによる結果と商用モデルによる結果が非常に近いので，節点内アルゴリズムは正しいと言える．

このモデルが分散処理システムの能力を持つか否かをテストするため，実際のシステムを作ったり，別の外部モデルによる実験はしていない．しかし，この計算モデルを用いて仮説的に分散処理システムをモデル化する実験を行い，得られた出力報告のエントリを手計算による値と比較した．その結果は，満足できるものであった．

5. 限界の分析

本稿で述べるモデルによって，トランザクションは互いに独立でデータベース・セグメ

ントの複製の存在しない，任意の分散処理システムを合理的に表現できる．実際の分散処理システムは，これよりもずっと複雑である．すなわち，トランザクション間には依存関係があるし，データベース・セグメントの複製も存在している．また，トランザクションがアクセスする必要のあるデータベース・セグメントは，一つの節点にあるものだけではない．トランザクションの中には，メッセージをブロード・キャストするものもある．入出力のシステムと通信網は，このモデルで表現されているものより複雑である．この節では，多様な分散処理システムを表現する能力に対するこのモデルの数々の限界が，どんな影響を持つかを吟味する．

しばらくの間，概念モデルが完璧であると仮定する．計算モデルの限界は何であろうか．計算モデルは分析モデルである．この分析モデルは，定常状態を統計量で表現したものである．つまり，分析モデルはある負荷とシステムの記述を仮定して構成し，その結果は当該負荷とシステムの記述にだけ当てはまる．この種のモデルをもってしては動的環境でのパフォーマンスを，そのまま予測することはできない．できることは，モデルごとに特定時点のパフォーマンスをスナップ・ショットするのみである．パフォーマンス予測のために計算モデルで用いる計算は，待ち行列分析技法のうちで，多重クラスで，積形のネットワーク分析を基礎としている．この技法では，トランザクションの独立性と資源を重複して持つことを仮定しており，サービス時間と到着の分布に関しては，現実の分散コンピュータ・システムとは必ずしも一致していない．A. O. Allen[1], P. J. Denning[7], K. M. Chandy[5] が示唆するところによると，このような近似を，少なくとも集中システムに適用するときには，モデルの表現の妥当性を大幅に損うことはない．分散コンピュータ・システムは非常に複雑な集中システムとみなせるので，この限界が極めて強すぎるとは思われない．

概念モデルは，通信網の相互接続マトリックスに制限を課していない．けれども計算モデルでは，制限している．計算モデルでは，一つの始節点と他の一つの終節点との間の路は，ただ一つしか許されない．この制限は，計算モデルの実現を単純化するために適用されたものであり，相互接続アルゴリズムをプログラムし直せば取り除くことができる制限である．同じように，一つのトランザクションが，その始節点から終節点へ至る路を逆にたどるという概念モデルの制約も，計算モデルで節点の相互接続アルゴリズムのプログラムを直せば取り除かれよう．

計算モデルでは，CPU，入出力ストリングおよびオペレーティング・システムの特性をあらかじめ定義した表を利用するので，利用者がネットワークの仕様を定めるプロセスを単純化できる．しかし，この技法では，あらかじめ必要な特性を集めて表を作る必要がある．この表を作る人間は，通常**モデルの監理人**である．このようなやり方の特徴は，使いやすいというよりも，モデルの監理人が利用者にある構成要素についてだけ考えるように枠をはめることができる点にある．これは，市場では役に立つ特徴かも知れないが，そうでない場合には足枷となるかもしれない．

概念モデルについてはどうであろうか．概念モデルおけるトランザクション・モデルの機能レベルには，一つの明白な制限がある．考えているのは非常に低水準のトランザクションであるから，利用者は常に高水準のトランザクションを一般トランザクションに分解しなければならない．ファイル転送とジョブ転送の場合には，DDP トランザクションを使えば自動的に計算モデルに移せた．しかし，他の場合には，利用者が必要な一般トランザクションを作り出し，適当なパフォーマンス予測値を集めなければならない．この場

合，利用者は適切に分解と収集を行うために計算の方法を理解している必要があり，モデルの使いやすさに反することとなる．利用者が，複雑なトランザクションを一連の一般トランザクションに分解できるなら，表現上の問題は発生しない．利用者がシステムにデータベース・セグメントの複製の存在を許してモデル化しようとし，更新の伝播のトラフィックを説明するために仮想の一般トランザクションを導入しても表現上の問題は起きない．むしろ，高水準トランザクションを一般トランザクションにアド・ホックに分解するのがうんざりするような仕事で，間違いが多くなることが問題である．もし，一般トランザクションが，複数の順次適用される終節点での核を持つならば，分解のプロセスは幾分簡単になろう．トランザクションのモデルをこのように拡張し，計算モデルへの影響はほとんどなしに概念モデルへ組み込むのは，今後の課題である．

概念モデルと計算モデルに共通な他の制限がある．トランザクションの同期に関する制限は，おそらく最も明白なものであろう．トランザクションが同期して，初めてデータベースの一貫性は保たれ，複数のトランザクションの同時並行処理も制御でき，またトランザクションを互いに合併することもできる．同期をモデル化するにはトランザクションの独立性を緩めて，概念モデルと計算モデルの複数資源占有制限を緩める必要がある（現存のモデルに適合するように同期を近似するには，利用者が各トランザクション，あるいは各装置ごとに同期の遅れの確率と同期の遅れ時間の分布を決める必要がある．残念ながら，これはこのモデルの潜在的利用者の大多数の知識の範囲を超えるものと思われる）．同期に関する制限は，どの程度重大であろうか．トランザクションの並行性とデータベースの一貫性を制御する同期は，今日の実際の分散処理システムにあっては，ロックによってトランザクションはめったに待つことがないように思われる[10]．トランザクションの合併については，それほど明らかでない．今日のシステムにおいてトランザクションの合併は一般的でないが，将来のシステムでデータの流れと業務の流れという概念が受け入れられるようになると一般化すると思われる．

概念モデルと計算モデルとに共通するもう一つの制限とは，CPU の記憶容量が有限なことである．概念モデルでは，CPU の記憶容量を無制限であると仮定している．計算モデルでは，開放型チェインを使ってトランザクションを表している．開放型チェインにおいてはトランザクション総数はいくらでも大きくできるので，無制限の，あるいは極めて大きな記憶容量を意味することになる．計算モデルが開放型チェインを使うのは，第1に計算を迅速に行うためであり，トランザクション総数を制限するかもしれない閉鎖型チェインで実現すると，むやみに記憶容量を要し計算時間もかかりすぎていたであろう．開放型チェインを使うと，トランザクション総数が小さく，サーバ(たとえば，CPU)の利用度が高いとき，コンピュータ・システムのトランザクション応答時間とトランザクション総数は極めて不正確な表現となる．幸い評価によって実際の実験値の上限がわかるので[4]，設計者の眼でみると，開放型システムが目標にかなうならば，いくぶん閉鎖型の実際のシステムもきっと目標にかなうものとなろう．今日のコンピュータ・ハードウェアの経済性からすると，ハードウェア装置を十分に利用することはなく，また CPU の記憶容量は大きくできる．つまり，トランザクション総数を大きくすることが実際的である．もしもトランザクションを表すのに開放型チェインのみならず閉鎖型チェインでも表す必要性があるなら，計算モデルはモジュール構造なので，そのアルゴリズムを追加挿入するのは容易である．概念モデルへの付加も直ちにできる．

両モデルに共通のもう一つの制限は，節点における入出力複合体の表現にある．両モデ

ルともに，入出力複合体はストリングから構成されている．個々のストリングは一つの先着順 (FCFS) サーバであり，チャネル，制御装置および機器の集まりに対するサービスをする．一つのストリングは，そのストリング用に決められた制御装置とチャネルによってサービスされる一つの入出力機器（たとえばディスク装置）に対応する．明らかに，これは実際とは異なる．典型例では，一つのチャネルが一つの制御装置につながれ，その制御装置は一連のディスク装置に接続されていて，その動き方は複数のディスク装置が同時に動いて，たとえばシークとシークないしデータ転送とが重複しているといったものである．制御装置に複数のチャネルがつながれ，また複数ストリングのディスク装置を制御することもよくある．論理モデルが単純にただ一つの入出力ストリングのサーバを持つというのは，このような事実に照らし，あまりに単純すぎることとなる．しかし，このように単純なものであっても実際の入出力複合体を設計する上ですぐれた洞察が可能である．入出力複合体の設計に対しては，単純な単一サーバの入出力ストリングを複合した，あるいは待ち行列と等価な入出力ストリングを使って拡張すると，反応時間とサーバ総数の評価値の点でかなり忠実な結果を得ることができる[5],[6]．このやり方では，特別の入出力モデル[2]を計算モデルと共に使って，実際の入出力複合体の振舞いを表すのに用いる単純な単一入出力ストリングのサーバの特性を決める．

節点間通信の遅延の表現についても，同じような制限が出てくる．入出力複合体の場合のように，等価な待ち行列を用いると近似操作を最小にできる．等価な待ち行列を自動的な機能として，モデルにどのように組み込むか考えるのは興味深い問題である．モデルの監理者がまず何回か試行して，その結果を用いて構成要素の集合を作り出し，利用者はそこから選択することによって入出力ストリングと節点間通信の遅延を決めるということもできる．メカニズムは，すでにモデルの中に存在している．しかし，この方法の欠点は，あらかじめ定める構成要素の数が，多数の可能な入出力複合体と節点間通信の遅延を十分に含む必要があり，したがって非常に大きなものになることである．そのため，特殊な場合を除いては，この方法はその汎用性の必然的結果としてうまくいかない．一般的な解として最もよさそうなのは，計算モデルを階層モデルに変換することである．計算モデルはモジュラ構造なので，この方法の場合モデルを完全に書き直す必要はない．

6. おわりに

本稿は，分散処理システムのパフォーマンスを予測するための利用者になじみやすいモデルについて述べたものである．このモデルによりトランザクションが独立で，データベース・セグメントに複製のない分散処理システムは，合理的に表現できる．さらに，このモデルにより，これらの条件を緩和したシステムも表現できる．このモデルによってシステム分析を行うと，ハードウェア，データベース，プログラムおよび負荷の選択や配置に対し，システムのパフォーマンスがどれだけ感応するかをうまく評価できる．このモデルは，待ち行列のネットワークを評価する分析手法に現在の新しい技術を採用している．この手法を使うと，ネットワーク全体のパフォーマンスを迅速で正確に予測できる．したがって，利用者はある節点におけるアクティビティを別の節点のアクティビティと関係づける必要がない．

将来はどうなるであろうか．妥当性の確認の実験は，さらに積み重ねられることであろう．トランザクションの同期と，トランザクションによる複数資源の占有が必要な状況を，さらに鋭く表現することも今後の研究すべき目標である．これがうまくいくと複製し

たデータベースを持ち，問い合せのブロード・キャストを行う分散コンピュータの振舞いを，モデルで克明かつ容易に表現できるようになる．

最後に，本稿の査読をしていただいた大勢の人々に謝意を表したい．

（ソフトウェア・プロダクト統括部 ソフトウェア一部 松井 節男
技術企画部 テクニカル・パブリケーション室 桑野 龍夫 共訳）

参考文献 [1] A.O. Allen, "Queueing Models of Computer Systems," *Computer*, April 1980, pp. 13-24.
[2] Y. Bard, "A Model of Shared DASD and Multipathing," *Comm. ACM*, October 1980, pp. 564-572.
[3] F. Baskett, K.M. Chandy, R.R. Muntz, J. Palacios, "Open, Closed, and Mixed Networks with Different Classes of Customers," *J. ACM*, April 1975, pp. 248-260.
[4] J.P. Buzen, P.S. Goldberg, "Guidelines for the Use of Infinite Source Queueing Models in the Analysis of Computer System Performance," *National Computer Conference*, 1974, pp. 371-374.
[5] K.M. Chandy, C.H. Sauer, "Approximate Methods for Analyzing Queueing Network Models of Computing Systems," *ACM Comput. Surveys*, September 1978, pp. 281-317.
[6] K.M. Chandy, C.H. Sauer, "Approximate Solution of Queueing Models," *Computer*, April 1980, pp. 25-32.
[7] P.J. Denning, J.P. Buzen, "The Operational Analysis of Queueing Network Models," *ACM Comput. Surveys*, September 1978, pp. 225-262.
[8] P.H. Enslow, "What is a Distributed Data Processing System?" *Computer*, January 1979, pp. 13-21.
[9] H. Garcia-Molina, "Performance of Update Algorithms for Repilcated Data in a Distributed Database," *Report STANCS-79-774*, Department of Computer Science, Stanford University, June 1979.
[10] J. Gray, P. Homan, H. Korth, R. Obermarck, "A Straw Man Analysis of the Probability of Waiting and Deadlock in a Database System," Fifth Berkeley Workshop on Distributed Data Management and Computer Networks, February 1981.
[11] A.R. Hevner, G.M. Schneider, "An Integrated Design System for Distributed Database Networks", *COMPCON 80-Fall*, September 1980, pp. 459-465.
[12] L. Kleinrock, *Queueing Systems-Volume I: Theory*, John Wiley & Sons, New York, 1975.
[13] L. Kleinrock, *Queueing Systems-Volume II: Computer Applications*, John Wiley & Sons, New York, 1976.
[14] A. Krzesinski, P. Teunissen, "An Introduction to System Modelling Using the Stochastic Network Analysis Program," *Technical Report RW 77-05*, Dept. of Computer Science, University of Stellenbosch, June 1977.
[15] A. Krzesinski, P. Teunissen, "Stochastic Network Analysis Program User's Manual," *Technical Report RW 76-02*, Dept. of Computer Science, University of Stellenbosch, March 1977.
[16] A.I. Levy, "Introduction to Practical Operational Analysis; An MVS Perspective," *CMG XI*, December 1980, pp. 208-214.
[17] F.J. Maryanski, "Backend Database Systems", *ACM Comput Surveys*, March 1970, pp. 3-25.
[18] D. Potier, Ph. Leblanc, "Analysis of Locking Policies in Database Management Systems," *Comm. ACM*, October 1980, pp. 584-593.
[19] C.A. Rose, "Measurement and Analysis for Computer Performance Evaluation," Sc. D. Dissertation, George Washington University, September 1975.
[20] C.A. Rose, "A Calibration-Prediction Technique for Estimating Computer Performance," *National Computer Conference*, 1977, pp. 813-818.
[21] S.M. Ross, *Introduction to Probability Models*, Academic Press, New York, 1972.
[22] J. Spragins, "Analytic Queueing Models," *Computer*, April 1980, pp. 9-11.

執筆者紹介 Russell C. Heinselman

1968年に数学の B. S. を，また1972年にコンピュータ・サイエンスの M. S. をいずれも Minnesota 大学で取得．1973年に Sperry 社に入社，1980年までシステム設計に従事．その後 Sperry Research Center で，分散システムのパフォーマンスおよび評価する待ち行列モデルの分析ネットワークを開発．現在は，オフィス・オートメーションの設計開発に従事し，利用者インタフェースの仕様を担当．

報告 ネットワーク・モデル上の関係型ビューの問い合せ

Querying Relational Views of Networks

A. Rosenthal, D. Reiner

要 約 CODASYL モデル，あるいは実体-関連モデルにもとづいて生成されたデータベースに対する関係型問い合せを支援する一つの設計を述べる．ここに述べるアーキテクチャは通常の方法とは異なり，純粋な関係型システムのビュー機能によってモデル間の問い合せ翻訳を処理する．CODASYL データベースの意味を表現する関係型スキーマは，一貫性制約を持った関係型ビューの集まりとして表現され，本システムでは生成した関係型スキーマに対する各 SQL 問い合せは，本質的には CODASYL レコードである“アクセス・リレーション”に対する SQL 問い合せに翻訳される．セット・リンクは，述語 join によって表現される．なお，問い合せに対するアクセス戦略を見出すために，少し修正された関係型問い合せの最適化機構が使われている．

Abstract We describe a design for supporting relational queries to databases created under a CODASYL or Entity-Relation model. The architecture is unusual in that intermodel query translation is handled by the view facilities from a pure relational system. The relational schema which captures the semantics of the CODASYL database is expressed as a collection of relational views with integrity constraints. Each SQL query against this system-generated relational schema is translated into a SQL query against "access relations" which are essentially CODASYL records. Set links are represented by join predicates. A slightly modified relational query optimizer is then used to produce an access strategy for the query.

1. はじめに

CODASYL モデル，あるいは実体-関連モデルにもとづいて生成されたデータベースに対する関係型の問い合せを支援するための一つの設計を述べる．使用者は，すべてのデータの関係型ビューを参照し，SQL のような関係型の言語で問い合せや更新を実行する．使用者は，レコードの平担なファイル（データベースのある部分は，そのような方法で実際に蓄積されるかもしれない）として蓄積されたリレーションと，いくつかの CODASYL レコード型とセット型から得られるリレーションとを区別しない．

一般に，モデル間支援システムは，まず CODASYL スキーマを処理し，意味的に同値な関係型スキーマを作り出す．ここでの主要な考えは，CODASYL レコード（“アクセス・リレーション”）を直接に原型とするリレーションに対する SQL ビュー（“ビュー・リレーション”）として，関係型スキーマを定義することにある．CODASYL のセット型（親子リンク）は，二つのアクセス・リレーションを結びつける述語として取り扱われる．ビューの置き換え機構が，入力されてくる使用者の関係型スキーマに対する問い合せを，アクセス・リレーションに対する SQL 問い合せに自動的に翻訳する．更新の翻訳については，第3章で取り扱う．

次のような，より基本的なシステムから利用できるソフトウェアの上に，このシステムを構築することを考えている．

1) 索引つきの平担ファイルとしてデータを蓄積する“純粋な関係型 DBMS”での問い合せ最適化機構[10]とビュー変換機構[11].

2) 既存 CODASYL システムのアクセス・ルーチン

これらのシステムを結合する機能を追加することにより，新しく開発するコードの量を少なくしている（図1）.

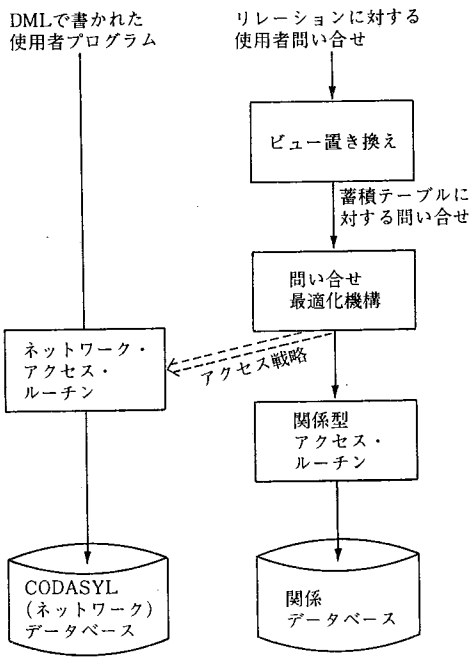


図 1 システム概観

Fig. 1 System overview

問い合せ翻訳のアプローチは，対象とする巡航型システムに関係型のインタフェースを提供する他の方法は，データベース変換，データ抽出，問い合せ解釈，あるいは巡航型 DML を含む親言語プログラムへの問い合せ翻訳に比べて大きな利点がある.

ネットワーク型データベースを関係型データベースに変換することは，アプリケーション・プログラムもまた変換しなければならないので，一般に現実的な方法とは言えない. ほとんどの関係型システムは，セットの子レコードの順序が持つ情報を捕える機能を提供していないので，データベースの変換自体もむずかしい. そのような機能が提供されれば，順序が意味を持っているセットを明らかにするために，すべてのアプリケーション・プログラムを調べなければならなくなる.

別のアプローチは，古いデータベースをマスタ・コピーに残したまま，ネットワーク型あるいは階層型データベースからデータを抽出する方法である. 抽出アプローチの問題点は，費用と更新のむずかしさにある. 抽出は（より古い情報を使うのでない限り）各処理の前に実行されなくてはならず，また処理の間はデータの二つのコピーが格納されなくてはならないので，費用がかかる. マスタ・コピーは，むずかしい低水準の巡航型更新言語を使ってしか更新できないというところに更新の問題がある. このため，ほとんどの使用者とアプリケーション・プログラマは，熟練者が準備するトランザクション・プログラムを介してのみデータベースの保守を行えるということになる.

Zaniolo[13] と Goldman[5] は，CODASYL スキーマから，(一貫性制約を含んだ) すべての論理情報を保存した使用者に判りやすい関係型スキーマを導き出す方法を示した．導き出されたスキーマに対する関係型の問い合せと更新を，対象のネットワーク型データベースに対して実行するプロトタイプのインタプリタが作られている[3]．

彼らのプロトタイプのインタプリタによるアプローチには，次の欠点がある．

1) アクセス戦略が，きわめて非能率的にしかならない．インタプリタのアプローチと，一時に1レコードを処理対象とするネットワーク型のコマンドは，最適化を図ることをむずかしくする．
2) インタプリタは，実行時に主記憶に常駐しなければならない．
3) インタプリタは，開発に相当の費用がかかる．

マルチベースのプロトタイプについて，Dayalは文献[4]で関係型の問い合せをCODADYL命令語を含む親言語プログラムに変換することができるとしている．このマルチベース方式の大きな強みは，既存の CODASYL 型 DBMS を変更することなく利用できることにある．このことは，特別な目的のシステムのすべての多様さに適合するアクセス戦略を決めるために多くの手作業が必要であることを意味している．「CODASYL の実行時ルーチン上での一時に集合を対象とする操作を含む蓄積管理システム上の制御機構がある (そして必要とされる)」．これにより，より簡単な変換スキーマを定義することができる．さらに，蓄積管理システムが一時的ファイルの創成やソートを行うと仮定できるので，組み合せの走査や小さな資源をしまうようなアクセス戦略が利用できる．マルチベースは，各蓄積システムの機能に合せてその最適化を行う．Vassiliou は，その文献[12]で同じく変換アプローチを提案しているが，最適化については触れていない．変換アプローチの別の欠点は，変換プログラムが，特定の親言語に制約されてしまうということである．

2. CODASYL データベースのための関係型スキーマの生成

2.1 定　　義

本稿では，ほとんどの場合 CODASYL の用語が使用されるが，実体-関連 (E-R) 型データベース上の関係型ビューについてわずかな変更が成されている． CODASYL レコード型のフィールドの集まりは，その型の二つのレコードがそのフィールドに同一の非空値を持つことがないとき，“キー” と呼ばれる． レコード型Rをセット型Sの子レコード型とする．Rのフィールドの集まり K(S) は，同一親レコードを持つ二つの子レコードがその K(S) に同一の非空値を持つことがないとき，Sの “セット・キー” と呼ばれる．親レコードのキーと子レコードのセット・キーを合せることにより (もし両者とも非空値なら)，子レコードの実現値を一意に識別することができる． キーとセット・キーは，上に述べた性質を持つフィールドの最少の集まりであるとする．セット型Sは，子レコード型の各実現値が，セット実現値のどれかの子レコードでなければならないとき，“永久的自動型 (automatic mandatory) と呼ばれる．フィールドは，“非空値は許されない (NNA)” あるいは “重複は許されない (DNA)” と宣言されていてもよい．

もしセット内の子レコードの順序が意味なく，各レコード型 Rec が NNA キーか，あるいは (永久的自動型のセットに対し) NNA セット・キーを持っていれば， CODASYL データベース中の情報は，関係スキーマを使って表現することができる[13]． 永久的自動型のセットは，環状になることが禁じられている．したがって，Rec からの親レコードへの鎖をたどることにより，究極的には NNA キーを持つレコード型にたどりつく．Rec の

実現値はキーとセット・キーの鎖（空かもしれない）により一意に識別される．Rec 中にない識別子の部分をそのレコードの "外部キー" と呼ぶ．複数の子レコード型を持つセットは，セット内の順序が意味のない限り，いくつかのセットに分割することができる．

2.2 一貫性制約

一貫性の制約は，スキーマの本質的な部分である．使用者が CODASYL データベースを理解する必要がないようにするため，スキーマ変換ルーチンは，CODASYL の一貫性情報を知り，それを関係型スキーマに表現しなければならない．

Zaniolo[13]は，"CODASYL スキーマ" から得なければならない一貫性情報として次のものをあげている．

1) あるレコード中のフィールドの集まりは，キー（DNA）と宣言されていなければならない．すなわち，どの二つのレコードもキー中のすべてのフィールドに対し，同一の非空値を持たない．
2) あるセットの子レコード型中のフィールドの集まりは，そのセットのセット・キーと宣言されていなければならない．
3) セット型は，永久的自動型と宣言されていなければならない．
4) フィールドは，NNA と宣言されていなければならない．フィールドの集合が，自然な集団項目（たとえばキー）となっている場合，R1．キー中の各フィールドに対する NNA 宣言は R1．キー（NNA）と略記して使われる．

使用者のために生成される "関係型スキーマ" におけるビュー・リレーションの定義に，次の三種類の一貫性の規則を設ける．

1) フィールドは，NNA と宣言されていなければならない（上述の略記法が複数の NNA 宣言に対して使われる）．
2) フィールドの集まりは，DNA と宣言されていなければならない．（各ビュー・リレーションは，使用者が参照できるフィールドから選ばれた少なくとも一つの（NNA, DNA）キーを持つ．）
3) サブセット制約が宣言されていなければならない．すなわち，フィールド R.A の各非空値は，ある他のフィールド R′.B の値として現れなければならない．このことは，R〔A〕⊆R′〔B〕と記される（〔 〕は射影を表す）．AとBがフィールドの集まりの場合にもサブセット制約は定義できる．（Zaniolo の文献[13]での "外部キー" 宣言は，B が R′ のキーである場合のサブセット制約である．）

これらの規則により，Zaniolo が CODASYL スキーマで確認したすべての一貫性情報を関係型スキーマに表現することができる．CODASYL スキーマでの NNA と DNA の宣言は，そのまま関係型スキーマに現れる．関係型スキーマにおいて，親レコードのキーとセット・キーの組を DNA と宣言することにより，各 CODASYL セット・キー宣言を得ることができる．セットが永久的自動型であるという宣言は，NNA である親レコードから分離されたキーの宣言とサブセット制約中にそのキーを表現することにより得られる．

2.3 リレーションとしてのレコード型の取り扱い

CODASYL レコードの実現値は，レコードの実現値を一意に識別する組織別フィールド "tid" を含んでいると考える．通常，物理番地で十分なので tid の特別な開発はないと仮定する．レコード型は正規化（すなわち，繰り返し項目を持たない）されており，キーとして tid フィールドを持つので，それはリレーションである．組は CODASYL の蓄積

システムから直接に利用可能であると仮定される．これらの事実を強調するために，レコード型のリレーションを“アクセス・リレーション”と呼ぶ．

レコード型をリレーションとみなすことにより，最適化機構は CODASYL レコードの操作を考えることができる．蓄積管理システムは，アクセス・リレーションからの組（すなわちレコード）を検索する．そして，使用者の関係型スキーマ中のリレーションからの組を生成するために，関係操作の結合機能（JOIN）によっていくつかのレコードからの情報が結びつけられる．

レコード型をリレーションとして形式的に取り扱うことにより，アクセス・リレーション上にシステムが生成した“ビュー・リレーション”を表現するために，関係型の言語を使用できるようになる．このために，SQL が使用される．tid は使用者に表示されないことを除いて，通常のデータ・フィールドとして関係システムで取り扱われるものとする．リレーションRの tid フィールドは‘R.@’と記述される．

セット型は，ビュー・リレーションの定義中に述語として現れる．R0, R1 をセットSの親レコード型，および子レコード型としよう． セットSの情報は，“R0.@ と R1.@ により識別される組は，セットSの実現値により結びつけられる”という述語により記述される．このことは，R0.@…Link (S)…R1. @ と略記される．(Manola[8] もこれに類似したモデル構成を使用している.） 関係型システムのビュー処理機構は，ちょうど使用者サブルーチンにより開発された述語を扱うように，Link(S) を二つのフィールド上のまだ翻訳されていない述語として取り扱う． ビュー処理機構は，CODASYL 構造の意味を理解する必要はない．

Hwang[6], Katz[7], Vassiliou[12] の文献で述べられている生成されるスキーマに類似した関係型スキーマが，ビューによりいかに表現されるかを述べる．Zaniolo の文献[13] における変換については，ここでは触れない．なぜなら，それは外部結合 (outer join) を必要とするからである．ここでは，できるだけ簡単にビューを基底とした変換を導入したい．しかし，Zaniolo のスキーマは，より良い使用者インタフェースを提供する．ここでのスキーマのかわりに，彼のスキーマが採用されるなら，彼のリレーションの一つに対し，制限（restriction）や射影（projection）が必要となる．

この後で導かれる関係スキーマは，各実体（すなわちレコード型）のためのリレーションと，実体の関連の定義で使用されるセット型を除く各関連（すなわちセット型）のためのリレーションを含んでいる．実体と関連の区別は，問い合せ処理のためには重要でないが，更新の意味を定義するには有効である[2].

2.4 ビューを基底としたスキーマ変換アルゴリズム

第一ステップは，アクセス・リレーションやセット定義，一貫性情報などの CODASYL スキーマからのすべての論理情報を抽出することである．CODASYL スキーマに対応したビュー・リレーションは，ビュー定義に各レコード型や各セット型が一度は使われるように，下記に示す三つ（独立した実体，従属した実体，関連）の規則を任意の順に適用することにより非決定的に定義される．どの規則を適用すべきか，どの外部キーを持ち込むべきかどうかについては，しばしばいくつかの選択がある．図2は，スキーマ生成中におけるビュー・リレーションの情報の流れを示している．一度ビュー・リレーションが定義されれば，それは新しいビュー・リレーションの定義に使用できることに注意したい．

≪ビュー・リレーション定義における記法≫

1) R.＊ は，“tid フィールドを含む”Rのすべてのフィールドの集合を表す．Rはビ

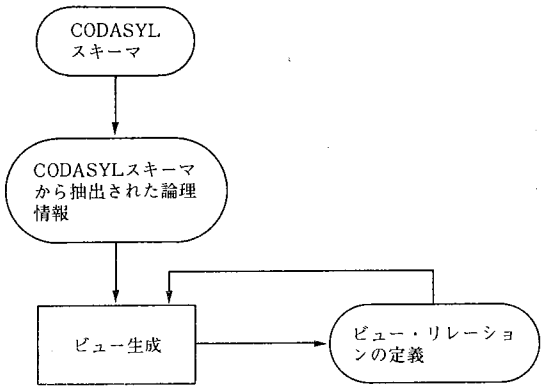


図 2 ビュー生成
Fig. 2 View generation

ュー・リレーション，あるいはアクセス・テーブルである．tid フィールドは，決して使用者には示されない．

2) R. key は，Rの一つの NNA キー中のフィールドを表す．（このキーは tid であってはならない．）

3) R. Set-key(S) は，いくつかの NNA セット・キーのフィールドを表す．Rはセット S の子レコード型である．

4) ビュー・リレーション名は，接尾語 _R. で表す．Rec に対する実体の表現であるビュー・リレーションは，Rec_R. で表す．そして，セット S の関連を表すビュー・リレーションは，S_R. で表す．

5) ビュー・リレーションのフィールド名は，対象となっているリレーション中のフィールド名に対応して同じものである．

つぎに述べる規則 1)〜3) は，ビュー・リレーション・スキーマを導出し，また各ビュー・リレーションの使用者が参照できる NNA キーを決定する（tid は使用者には見えない）．このほかの一貫性制約から導かれる規則については，付録Ⅰに述べてある．(E-R スキーマからの) n : m 関係を取り扱うために必要ないくらかの修正についても，また付録Ⅰに述べている．関係型の使用者には，ビュー・リレーション・スキーマとビュー・リレーションの一貫性制約から成るスキーマが示される．

1) 独立した実体……CODASYL レコード型 R 1 が，使用者の参照できる NNA キー (R. key で示される) を含んでいるならば，

```
Define View R 1_R as
  Select R 1. * from R 1
  key : Define R 1_R. key as R 1. key (NNA, DNA)
        (すなわち，R 1. key からの R 1_R のフィールドは，R 1_R の
         キーを構成する．)
```

2) 従属した実体……R 0 が，子レコード型 R 1 を持つセット S の親レコード型であり，S が NNA セット・キーを持ち，それが永久的自動型であり，ビュー・リレーション R 0_R がすでに定義されているならば，

```
Define View R 1_R as
  Select R 0_R. key, R 1. *
```

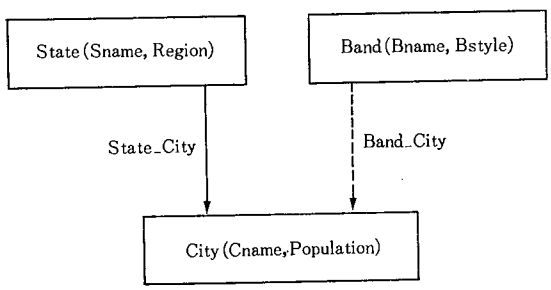
From R 0_R, R 1
Where R 0_R. @ … link (S) … R 1. @
Key : Define R 1_R. key as R 0_R. key U
R 1. Set-key (S) (NNA, DNA) ;

3) 関連……R 0_R と R 1_R がすでに定義されており，Sが親レコード型 R 0 を子レコード型 R 1 に結びつけ，Sに規則 2) がまだ適用されていないならば，

Define View S_R as
Select R 0_R. key, R 1_R. key, R 1_R.
（Sのセット・キー中のすべてのフィールド）
From R 0_R, R 1_R
Where R 0_R. @ … link (S) … R 1_R. @
Key : Define S_R. key as R 1_R. key (NNA, DNA) ;

上記の規則から，キーを持ったレコード型のビュー，あるいは永久的自動型のセットの親レコード型のキーとセット・キーから成るキーを持つビューが得られる．もしキーが得られなければ，そのレコード型を関係スキーマにおいて，重複なしに取り扱うことができない．関係型の言語は，値を基礎としており，重複している組の一つをアクセスしたり，更新したりすることはできないので，重複の存在は面倒なことになる．

図3は，ネットワークのスキーマ図の例を示している．これは，ネットワーク・スキーマ自体と，それと同等な関係スキーマから導かれている． CODASYL スキーマは，変換アルゴリズムの興味ある部分を強調するように書かれており，データ型の定義については触れていない．



（セットの意味） State_City：あるStateがそのCityを含んでいる（永久的自動型）
Band_City：そのCityにおける人気Bandを示している（手動型）

図 3 ネットワーク・スキーマ図

Fig. 3 Network schema diagram

≪CODASYL スキーマ≫

Record State
Sname (NNA, DNA), Region
Record City
Cname (NNA), Population
Set name is State_City
Owner is State, Member is City
Set-key is Cname
Membership is automatic mandatory

Record Band

Bname (NNA, DNA), Bstyle

Set name is Band_City

Owner is Band, Member is City

Membership is optional

≪同値な関係スキーマ≫

このアーキテクチャにおいて，スキーマ変換の出力は，特別な目的のデータ構造というよりは，SQL ビュー・リレーションの集まりである．導出されたビュー・リレーションは次のようになる．

Define View State_R as

Select State. * from State

Key : Define State_R. key as Sname (NNA, DNA)

Define View Band_R as

Select Band. * from Band

Key : Define Band_R. key as Bname (NNA, DNA)

Define View City_R as

Select State_R. Sname, City. *

From State_R, City

Where State_R. @ … link (State_City) … City. @

Key : Define City_R. key = (Sname, Cname) (NNA, DNA)

その他の一貫性情報：

City_R[Sname] ⊆ State_R[Sname]

上の三つのビューは，実体を表現している．

次のビューは，関連の意味を表現している．

Define View Band_City_R as

Select Band_R. Bname, City_R. (Sname, Cname)

From Band_R, City_R

Where Band_R. @ … link (Band_City) … City_R. @

Key : Define Band_City_R. key as (Sname, Cname) (NNA, DNA)

その他の一貫性情報：

Band_City_R[(Sname, Cname)] ⊆ City_R[(Sname, Cname)]

Band_City_R[Bname] ⊆ Band_R[Bname]

3. 簡単な問い合せ翻訳

3.1 ビュー置き換えによる翻訳

問い合せ翻訳は，新たな開発を必要としない．それは，通常のビュー置き換えとして取り扱うことができる．ビュー置き換えの機構は，リンクがフィールド R. @ と R′. @ 上の述語の場合のみ，リンクの意味を知る必要がない．このアーキテクチャを図 4 に示す．

SQL ビュー “V” の定義は，目的リストとリレーション，あるいは組変数のリスト (“From-list (V)”)，条件 (“Where-clause (V)”) から成る．V 1 が From-list (V 2) 中に現れるか，From-list (V 2) 中のあるビューの基礎となっているとき，ビュー V 1 はビュー V 2 の基礎となっている（あるいは V 2 は V 1 から導かれる）と言う．

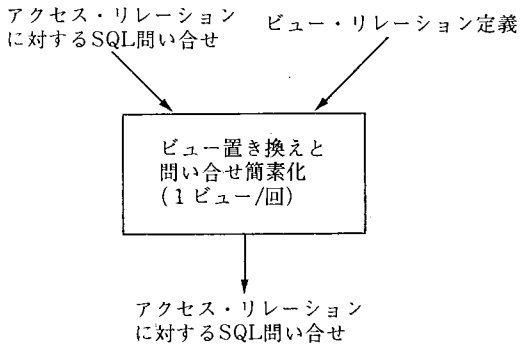


図 4　問い合せ変換アーキテクチャ
Fig. 4　Query translation architecture

(問い合せの修飾による) ビュー置き換えは，From-list (V) 中のリレーションの参照をビューVの参照に置き換えることにより問い合せQを翻訳する．この翻訳は，すべての参照がアクセス・リレーションになる問い合せを生成するまで繰り返される．ビューは，そのビューから導出されたビューが削除された後でのみ置き換えられる．このことは，ビュー置き換えを早め，つぎに述べる問い合せ簡素化規則 QS 2 を効果的なものにする．

問い合せQからビューVを削除するアルゴリズム (Stonebraker の文献[11] からの応用) は次のようである．

(VS1) From-list (Q) からVを削除する．そして，From-list (V) を From-list (Q) に付加する．(QとV中の組変数は名前で区別できなければならない．衝突がある場合には，この問い合せの処理中は，Vからの組変数Rは R′ に置き換えられる.)

(VS2) Q中の各フィールド V.A をビュー定義の目的リスト中の対応するエントリによって置き換える．(V.A が R.A に対応しており，R′ がステップ1で作られていれば，V.A は R′.A で置き換えられる.)

(VS3) Where-clause (Q) は “Where-clause (V) AND Where-clause (Q)” により置き換えられる．

例　題：次の関係型問い合せを考える．

```
Select＊from City_R
  Where Sname='Minnesota' and Cname≠'Frostbite Falls'
```

規則 VS 1-VS 3 により，問い合せは次のように変形される．

```
Select State.Sname, City.Cname, City.Population From State, City
  Where Sname='Minnesota' and
        Cname≠'Frostbite Falls' and
        State.@ ··· link (State_Cite) ··· City.@
```

3.2　問い合せ簡素化規則

複雑なビューに対して，通常のビュー置き換えは，不必要に複雑化した最適化の困難な問い合せを導き出してしまう．そのような不都合を排除するために，基本的な意味情報を利用する問い合せ簡素化規則 QS 1-QS 3 を追加する．

CODASYL 上の関係ビューは，しばしば結合 (join) を使用し，純粋の関係型システムにおける，ほとんどの使用者定義ビューよりも複雑になっている．しかし，純粋の関係データベースが再編成されると，データベース管理者は，しばしば新しいリレーションの上

に古いインタフェースを提供するためにビューの定義をすることになる．そして結果的に複雑なビューを作り出すことになる．したがって，問い合せ簡素化規則は，純粋な関係システムにおいても有効なものである．

(QS1) 代数的簡素化：次の規則を適用することにより Where-clause を簡素化する．

a）不必要な結合（join）を取り除く．（たとえば“T 1. A＝T 2. B and T 2. B＝constant”を“T 1. A＝constant and T 2. B＝contsant”により置き換える．）

b）余分な述語の削除（たとえば，“T 1. A＝T 2. B”と“T 2. B＝T 1. A”の一つを削除する．）

c）同値律の述語の削除（たとえば“T 1. A＝T 1. A”を削除する．）

(QS2) 不必要な組変数の除去：Tと T′ をリレーションT上の組変数と仮定する．また，Where-clause がTのあるキーKに対し T. K＝T′. K の結合を含んでいるとする．リストから T′ を削除し，Where-clause から (T. K＝T′K) を削除し，すべての T′ を Tに置き換える．

Kはキーであり，T. K＝T′. K なので，Tと T′ はいつも同じ組を参照しなければならない．したがって，T と T′ は重複していることになる．

ビュー処理の別の方法として，問い合せの変換時ではなく，ビューの定義時に各ビューをアクセス・リレーションに対するものに変換する方法がある．この方法は，ビューの置き換えに時間が少なくてすむが，QS 2 の開発がむずかしいものになる．なぜなら，各アクセス・リレーションの外部キーが認識できなければならないし，また QS 2 は特定の順番で組変数に適用されなければならないからである．

(QS3) 削除可能な組の除去：“R 0. A＝R. B”あるいは“R 0. @ … link (S) … R. @”の形のQ中の項は，もし R 0 が Where-clause のどの他の項にも現れず，かつ R 0 がQの目的リスト中のフィールドでもなければ，R 0 とRの“半結合 (semijoin)”が生じる．QがあるRと R 0 の半結合を生じ，各Rレコードが対応する R 0 レコードをただ一つ持つことを保証されていれば，R 0 は“削除可能”と呼ばれる．（リレーションに関しては，これは，R〔B〕⊆R 0〔A〕のサブセット従属のことである． CODASYL セットに関しては，これはSが永久的自動型であることを意味する．各Rレコードは，陰に制約 R〔親レコードの tid〕⊆R 0〔tid〕をともなった親レコードの識別子（すなわち親レコードの tid）を含んでいるので，後者はサブセット従属性の形になる．対応する R 0 レコードの数は，重複の正しい数を得るために唯一つでなければならない．）

削除可能なテーブルの内容は，問い合せの結果に影響を与えてはならない．“削除可能なテーブル R 0 を除去するために”，From-list(Q) から R 0 を削除し，Where-clause(Q) から R 0 を含んでいる項を削除する．R 0 を削除すると，Rが削除可能になることがあり得ることに注意する．

問い合せの簡素化の必要性を説明するために例を示す．QS 1 は，最適化機構の選択計算を簡素化する．QS 2 の必要性をみるために．次の問い合せ Q 2 を考えてみよう．

```
Select State_R. Region, City_R. Sname, City_R. Cname
  From State_R, City_R
  Where State_R. Sname＝City_R. Sname
```

規則 VS 1-VS 3 による City_R の除去の結果は，

```
Select State_R. Region, State_R′. Sname, City. Cname
```

```
From State_R., State_R′, City
Where State_R Sname=State_R′. Sname
  and State_R′. @ … link (State_City) … City. @
```

QS 2 は State-R′ を除去し，それを引用していたものを（たとえば，目的リスト中のもの）State_R で置き換える．簡素化された問い合せは次のようになる．

```
Select State_R. Region, State_R. Sname, City. Cname
  From State_R, City
  Where State_R. @ … link (State_City) … City. @
```

それから State_R を置き換える．

問い合せがリレーションを基礎としているすべてのレコード型を必要としないとき，QS 3 は大きな非効率を取り除く．たとえば，問い合せ Q 3：“Select Cname from City_R” において，不必要な State のアクセスの費用はきわめて大きい．問い合せのどの部分も，City レコードが結びつけられている State の実現値により影響を受けない．そして，State_City は永久的自動型なので，link 述語は，すべての City レコードを満足する．(QS 3 に類似した規則が，単一方向の外部結合を使用して定義されたビューを取り扱うために呼び出される[11])．

4. 先行の仕事との比較

4.1 関係スキーマの生成

多くの研究者が，CODASYL データベースを表現する関係スキーマを定義するためのアルゴリズムを開発している．そして，CODASYL データベースに対する関係型の問い合せや更新を変換するアルゴリズムを与えている．どの方法もその第一段階は，ネットワーク・スキーマに同等な関係スキーマを生成することである．

われわれが生成するスキーマは，Hwang[6]，Katz[7]，Vassiliou[12] らが生成するスキーマとは，少々異なっている．彼らは，各実体が一つのキーを持つ単純な E-R モデルを使用している．CODASYL データベース中の情報を正しくモデル化するためには，関係キー制約を使ってセット・キー宣言をしなければならない．われわれは Hwang, Katz, Vassiliou らが使用したのと同じ関係型スキーマを使用し，スキーマ中にはない結合に関するキー制約を果たし，実体間の関連を表現するリレーション中にセット・キーを含むようにした．セット・キー制約は，S_R に対する追加のキー制約を定義することにより得られている．Zaniolo[13] はセット・キー情報を正しくとらえていることと，原始データ項目 (1978 CODASYL 仕様) も取り扱うことを注意しておく．

われわれは，スキーマ変換の実現の基礎としてビューを使用し，これまでの実現よりもより簡素で健全な変換法を得ている．

1) 簡素化……変換が進み，“一時に一つのビュー・リレーション” を生成するにつれ，各新しいビュー・リレーションがネットワークの詳細を隠すことになる．結果として，“一時に 1 集合” 以上のことをもはや考える必要はなくなる．リレーション・キーの定義中のキー属性の結合を除いて，いくつかのレコード型からのフィールドのリストとして外部キーを操作するための機構を必要としない．あらかじめ定義されるビューの上に構築することは，すべての操作をちようど二つのリレーションに局所化する．もし，ただ二つのリレーションが各 From リスト中にあるのであれば，更新のための写像規則や，一貫性制約，トリガー，キーなどは，開発者にとって理解が容易で

かつ定義しやすいものになる.

2) 健全性……スキーマ生成においては，多くの選択がある．それは，レコード型やセット型の出現の順序により決定されるだろう．ほとんどの他の開発においては，レコードやセットが追加されたあとにスキーマの生成を行うと，結果としてのスキーマは前とは異なったものになり，古いビューのすべてを含むことはない．したがって既存の使用者プログラムは正しくないものになる．われわれのビュー生成の方法では，すでに定義されているビュー・リレーション上に追加的に生成されるので，古いビューを保存するための特別な考慮を必要としない.

レコードやセットが削除されれば，それらから導出されていたビューも削除されなければならない．これまでのアプローチでは，削除のための機構は，外部キーや鎖の中のセットを識別するラベル，変換アルゴリズムの他の部分を理解する必要があった．われわれのアーキテクチャにおいては，削除されるレコードやセットを参照している単一のビューを識別するだけでよい．この後，関係システムのビュー操作機能は，無効になったものから導出されているビューを識別し削除するために使われる.

われわれは，関係モデルを“空値”を含むように拡張する必要はなかった．(Zaniolo[13] は空値を定義したが，それらは外部キーのみに使用されているにすぎない.) R〔A〕と R′〔K〕間の外部キー宣言は，われわれのシステムにおいては“R〔A〕⊆R′〔K〕；K は (NNA, DNA)”に変換される．部分的に空であるAの値は，Kの値として現われることがないので それらはサブセット制約によって規則外となる.

4.2 問い合せ処理

Hwang[6] は，問い合せ変換の中間段階として実体-関連スキーマを使用する．問い合せは，高水準のネットワーク問い合せ言語 (NQUEL) に変換される．NQUEL に対し最適化機構を作ることはできるが，最適化について触れていない．各実体型や関連型は，われわれのビュー・リレーションの一つに同値である．「そして翻訳の結果である NQUEL 問い合せは，われわれの関係型問い合せに同値である」．したがって Hwang[6] の正しい翻訳の証明は，また，われわれの翻訳を正当化する.

関係モデルのビュー機構を使用して，すべての問い合せを変換するという，われわれのアーキテクチャには大きな利点がある.

1) われわれの定式化は概念的により経済的であるから，新しいコードの開発をほとんど必要としない．その差異はインタプリタと比較して大きい．Hwang[6] の方法に比べてさえ大きい．E-R モデルにおいては，異なった置き換え規則が実体や関連に使われているのに，われわれは対象の一つのビュー型を持っているにすぎない．すべての問い合せ変換は，拡張された関係システムで利用可能なビュー置き換え機構により処理される.

他のデータ・モデルにさらにオーバヘッドを加えることは，ほとんど利点がないようにみえる．E-R モデルの追加は，最適化のためには必要ない．われわれは，E-R モデルの最適化機構を書くよりも，アクセス・リレーションと結合の述語を処理できる関係最適化機構を設計した[10]．(たとえ既存の E-R システム上に構築されても，関係型から E-R 型への変換の多くは，関係型ビューの置き換えにより処理される.)

われわれも Hwang, Dayal[6] も，不必要な組変数の導入を発見するためにビューが削除される順番を注意深く検査する．(順番は次のとおりである．対象となる実体の

前に関連を削除する．セット・キーのチェインにおいて上向法で削除する．）われわれは，ビューの導出された逆順に削除するという非常に単純化した規則を開発した．

2) 新しい機能は，関係型とネットワーク型の変換に使われるばかりでなく，関係型の使用者にとっても利益をもたらす．Hwang[6] の文献は純粋の関係型使用者にとって有用なばかりでなく，問い合せ変換に埋め込まれる規則として有用な簡素化規則を追加した．われわれの類似した簡素化規則は，ビュー置き換えの一部を成すので，問い合せ翻訳で使われるだけでなく，すべての関係型使用者にとって有用である．

ネットワークとして蓄積されているデータへの関係型更新を提供する橋渡しシステムを構築するために，われわれは，一緒に結びつけられる対象（レコードやセット）に対応したリレーションを更新する必要がある．(Zaniolo[13], Goldman[5], Vassiliou[12], Hwang[6] はすべて更新を許している．）われわれのアーキテクチャにおいては，これらの機能は，純粋な関係型使用者に結合述語を含む多くのビューを更新することを許している．

3) 問い合せ，あるいは更新は平担なファイルとして蓄積されやすいものや，CODASYL構造で蓄積されたものを含む多くのリレーションやビューを参照することができる．同じビュー置き換えと，問い合せ最適化機構がすべてのデータに対して適用できる．蓄積構造は，使用者の問い合せや更新に対して完全に透過である．

われわれの設計は，単一の蓄積管理システムを持つ“プリズム”システム（Katz の文献[7]）であるが，われわれのソフトウェア・アーキテクチャは Katz のパラダイムとは二つのことで異なっている．すべての物理的情報は，われわれの最適化機構にとって利用可能なので，単一の大域的な物理スキーマが存在すればよい．また CODASYL レコードは，アクセス・リレーションとみなされているので，モデル間変換や，問い合せ最適化，結合処理（すなわち，親レコードと子レコードからの情報の結合）は関係モデル中で成される．

5. 物理処理——ネットワーク・データの最適化機構モデル

問い合せ翻訳は，結合の述語“R 0. @ と R 1. @ により識別される組は，セット S の実現値により結びつけられる（R 0. @ … link(S) … R 1. @ と略記）”を持った SQL 言語によってアクセス・リレーションに対する SQL 問い合せを作り出す．この節では，いかに最適化機構がこの機能を取り扱うかを述べる．最適化機構の完全な議論については Rosenthal, Reiner の文献[9],[10] を参照せよ．

まず，セット S に関する子から親への巡航の最適化モデルを考える．親レコードの tid を Tmember の各組に概念的に追加する．非永久型のセットに対し空値を持つかもしれない新しいフィールドは，owner-tid(S) と書かれる．そして拡張された子レコードであるテーブルは Tmember$^+$(S)，あるいは Tmember$^+$ と書かれる．たとえば，City$^+$(State_City) は，フィールド City. @ Cname, Population, @ State を持つ．(Tmember$^+$ Join (link) Towner) は，各Tmember レコードと，その親レコード・ポインタにより結びつけられる Towner レコードを関連づける．（この Join は，また各レコードからの十分な情報を保存する．そこで，他のリンクをたどって続く Join を準備するために，他のレコードからの tid を結果とする結合を行うことができる．）

親レコードから子レコードへの巡航には，一つの技術的な困難がある．各親レコードは，典型的に一つではなく複数の子レコードへのポインタの集合を持っている．論理的な目的

のために(すなわち，結合の結果を決定するために)，〔(data-fields, member-tid 1, member-tid 2, …)〕は，Towner^{+}=〔(data-fields, member-tid 1)，(data-fields, member-tid 2)，…〕と正規化されていることを仮定する．物理レベルでは，正規化は，ポインタ・フィールドで並べ換えが行われたとき，あるいは結合の出力が作られたときにのみなされる．(興味ある拡張が，非正規化データを処理するために考えられている．) Rosenthal と Reiner の文献[10]において，同様の正規化が，リレーションとして索引を取り扱うために，ポインタ・リストを持った索引に適用されている．

T_1 と T_2 を，セット・リンクで結びつけられた二つのレコード型とする．(どちらが親レコードであるかを指定する必要はない．われわれは一般の“外部 tid”，あるいは“ftid”で owner-tid と member-tid を参照する．)

最適化機構は，(“T_1^+ Join (link) T_2”によりモデル化される) T_1 から T_2 への巡航と(“T_1 Join(link)T_2^+”によりモデル化される) T_2 から T_1 への巡航を結合の述語 Join の選択と考える．T_1^+ が与えられると，T_2 を直接にアクセスすることができる．また，T_2^+ が与えられると，T_1 を直接にアクセスすることができる．利用可能な結合戦略の評価後，最適化機構は最も安いものを選ぶ．結合戦略は，ループ戦略（たとえば，組み合せ探索，入れ子ループ）と，結合のためのオペランドのリレーションを準備するための物理的操作 (read, sort, cache) の両方を意味している[10]．

利用できる結合戦略は，基本的に，リレーションとしてみられる索引とリレーション間の等結合のためのものと同じである．最適化機構により提供される機能を次の方法で十分に利用する．

1) 巡航の方向の最適化……最適化機構は，同じ論理的結果を生み出すいくつかの代替結合を許している．詳細な実現の費用を比較したあと，単純に安い方向が選び出される．
2) 結合戦略の表現……アクセス・リレーションは，操作子として Sort, Scan, Join, Dreate_index を持つ抽象データ型として取り扱われる．結合戦略は，アクセス・リレーション上のこれらの操作子の用語によって表現される．
3) 費用のモデル……かたまりのデータあるいは分散したデータや，並べ換え，(索引からのポインタをすでにモデル化できる費用のモデルは，CODASYL 上でのアクセス戦略をモデル化するために必要なほとんどの機能を持っている．T_1 から T_1^+，あるいは T_2 から T_2^+ を作り出す費用を含めるために，ほんの少しの拡張が必要なだけである．各戦略は，両方をではないが T_1^+ 中の，あるいは T_2^+ 中の外部 tid を得るために使われることに注意する．
4) 戦略の広い多様性の評価……CODASYL プログラマは，通常 T_2 への直接アクセスのためにのみ T_1-T_2 リンクを使用する．最適化機構は，戦略のより広い多様性を評価する．出力の並び順を決定するために通常のフィールドでレコード T_1 をソートする．T_2 を逐次により効果的にアクセスするように T_1 の ftid フィールドで並べる．すべての T_1 レコードへのアクセス回数を削減するために (T_1 の索引からの) T_1 の tid と T_2 の ftid の共通集合をとることも考えられる．Rosenthal と Reiner の文献[10]において，索引の共通集合と“影のリレーション”が問い合せの処理の費用を実質的に削減する例をみた．これらの例は CODASYL リンクにも適合する．
5) 探索限界の発見的手法……いくつかの規準が最適化を速め，陽に考えなければならない結合の実現の数を減らすためにとられる．T_1-T_2 巡航（すなわち (T_1^+Join T_2)）

をモデル化するとき，T_2 が入れ子ループの“内部”オペランドでみなければならないということを指示することにより巡航の方向を仮定させる．同様の機能が，直接アクセス構造とその対象としているリレーション間の結合を制限するために，純粋な関係型システムの中に示されている．内部ループ・オペランドを指定する能力は，また方向づけられた外部結合を実現する時にも有用である[1]．

さらに，索引とリンクの実現は，結合開発方法に影響を与える．あるポインタ・チェイン蓄積構造に対しては，親レコード tid は，親レコードをアクセスすることなしに得ることができない．したがって，親レコードを得る費用を削減するために tid フィールドの並べ換え，あるいは tid リストの共通集合をとることは必要ない．付録Ⅱに，変則のアクセス戦略から利益を得ることができるかどうかにより，典型的なセットの実現法を分類した．

6. おわりに

CODASYL ファイルとして蓄積されたデータに関係型のアクセスを提供するために，関係型システムを拡張する方法を示した．注目すべき機能は次のものである．

1) CODASYL データベースの意味を表現する関係型スキーマは，一貫性制約を持った関係型ビューの集まりとして表現される．
2) 既存のビュー処理機能が，関係型の問い合せを翻訳するために使用される．そこでは，わずかな拡張コードが必要となる．
3) CODASYL 上の関係型更新を可能にするために追加された機能（問い合せ簡素化，単純ビューの更新）は，ソース言語レベルに追加される．これにより CODASYL 使用者と同様に関係型使用者にも利用可能となる．
4) 純粋な関係型システムの最適化機構は，CODASYL レコードを含む任意のアクセス・リレーションへの関係型問い合せを翻訳するように，たやすく拡張できる．
5) CODASYL セットの巡航は，結合演算に変換される．最適化機構は，両方向の巡航を含む結合戦略の費用を非常に広い範囲にわたって削減する．
6) セット・キーの属性は，セット情報を表現するリレーション上の一貫性制約を得るために移される．

なお，この仕事は，部分的に Sperry の大型システム・ソフトウェア部の支援を受けてなされ，Murray Edelberg からは，一貫性情報の保持その他について有益な示唆を得た．

〔付録 I〕 一貫性制約

データベースの利用者は，スキーマのみからデータベースを理解するので，一貫性制約は，ビュー・リレーションにより提起される関係型スキーマの本質的な部分となる．次の規則は，本文中の規則 1)～3)（p.22）により生成されたビュー・リレーションのための一貫性制約を与える．

1) レコード型のキーと NNA フィールドは，そのレコード型を直接にモデル化したアクセス・リレーションのキーと NNA フィールドになる．
2) Rel をビュー・リレーション V-R の From リスト中のリレーションとする．Rel は（規則 2）における）アクセス・リレージョンになるか，あるいは（規則 3）における）ビュー・リレーションとなる．A を V-R の目的リスト中に現われる Rel フィールドの集合とする．
   a) Rel.A が NNA であれば V-R.A は NNA である．
   b) 外部キー R0-R.key が Rel を V-R から持ってきたものとする．
      もしAが Rel のキーならば，A は V-R のキーである．

もしAがV-Rの定義中で使われたセットS中のセット・キーならば，Rel.A∪R0-R.keyはV-Rのキーである．

3) B=－〔V-R中に現れるRelのすべてのフィールド〕とし，Relはビュー・リレーションとする．（Relは規則2）における R0かもしれないし，あるいは規則3）におけるリレーションかもしれない．）

a） V-R中のBの各値は，Relから持ってこられるので，次のように宣言される．

$$V\text{-}R\,〔B〕\subseteq Rel\,〔B〕$$

（Relがビュー・リレーションでなければならない理由は，使用者のスキーマはビュー・リレーションへの参照のみを含むことができるからである．アクセス・リレーションを含む制約は意味がない．）

b） もしSが永久的自動型で，Relが規則3）の R1-Rならば，Relの各組はV-Rの組を生成しなければならない．そこで次のように宣言される．

$$Rel\,〔B〕\subseteq V\text{-}R\,〔B〕$$

Hwang[6]は，彼らが導出したスキーマ（それは，われわれが使用するものに似ている）は，またE-Rシステムへの関係型インタフェースを支援することを指摘している．彼らのモデルにおいては，セット・キーは規則2）で使われるものを除いて，示されない．もしSがm：nであるなら，われわれに必要な唯一の変動は規則2）において，キーが R1-R.key∪R0-R.keyであることである．

〔付録 II〕 典型的なリンク蓄積構造上の処理戦略

提供される結合の機構に従って，セット・リンクを蓄積するための共通の構造を分類する．ここでの分類は，Rosenthal, Reinerの文献[10]での直接アクセス構造のための分類システムの特別な場合である．

1) 独立性：T_1^+ は T_2 のアクセスなしに形成されうる．（同等に，$ftid_1$ はレコード t_2 のアクセスなしに得られる．）

2) 埋め込み性：T_2 は T_1^+ を形成するためにアクセスされなければならない．（すなわち，$ftid_1$ は実際に T_2 をアクセスすることなしに得ることはできない．）

独立構造は，直接アクセスと変った処理の両方のために有用である．（たとえばtidリストや影フィールド，並び換え，共通集合）埋め込み構造は，直接アクセスにのみ有用である．

（例 題）

独立性：

1) 子レコードの実現値は，その親レコードへのポインタを含む場合の子レコードから親レコードへの巡航

2) 子レコードから親レコードへの巡航の場合は，CODASYL子レコードが循環的にその親レコードにつながれているときでさえ，最後の子レコードの実現値を認識することができ，親レコードをアクセスすることなしに鎖フィールドから親レコードのtidを得ることができる．この戦略は，セット実現値の子レコードがその親レコードから離れて一緒に集められている場合に著しい．

3) 親レコードから子レコードへの巡航の場合，ポインタ・アレイがすべての子レコードを識別する．

埋め込み性：

1) 親レコードから子レコードへの巡航で，子レコードは親レコードを先頭とするリンク・リスト中にある．子レコードのリストを得るには，最後の子レコードを除いてすべてをアクセスする必要がある．

2) 子レコードの実現値が，親レコードに循環的につながれている場合の子レコードから親レコードへの巡航では，親レコードのアクセスなしに鎖フィールドから親レコードのtidを得るこ

とはできない.

（ソフトウェア・プロダクト統括部 1100/90導入P 原 潔 訳）

参考文献

[1] A. Chan, S. Fox, K. Lin, D. Ries, "The design of the ADAPLEX DBMS," *Computer Corporation of America*, Cambridge MA (submitted to ACM TODS)

[2] E. F. Codd, "Extending the database relational model to capture more meaning," *ACM Trans. Database Systems* Vol. 4, No. 4, Dec. 1979, pp. 397-434.

[3] D. Connelly, "A single-tuple relational interface for CODASYL databases," *Research report* Sperry Research Center, 1979.

[4] U. Dayal, N. Goodman, "Query optimization for CODASYL database systems," *Proc. ACM-SIGMOD Conf.*, 1982, pp. 138-150.

[5] J. Goldman, "Automated generation of relational schemas for CODASYL databases," *Research paper* 79-13, Sperry Research Center, 1979.

[6] H. Hwang, U. Dayal, "Using the entity-relationship model for implementing multi-model database systems," *Second Entity-Relationship Conf.*, Washington D. C., Oct. 1981.

[7] R. Katz, "Software architectures for heterogeneous database management," *Proc. COMPSAC*, 1981, IEEE, pp. 33-42.

[8] F. Manola, A. Pirotte, "CQLF," *ACM-SIGMOD Conf.* 1982, pp. 94-103.

[9] D. Reiner, A. Rosenthal, "Strategy spaces and abstract target machines for query optimization," *Database Engineering*, Vol. 5 No. 3 Sept. 1982, pp. 56-60.

[10] A. Rosenthal, D. Reiner "An architecture for query optimization," *ACM-SIGMOD Conf.*, 1982, pp. 246-255.

[11] M. Stonebraker, "Implementation of integrity constraints and views by query modification," *ACM-SIGMOD Conf.*, 1975 pp. 65-77.

[12] Y. Vassiliou, F. Lochovsky, "DBMS transaction translation," *Proc. COMPSAC*, 1980 IEEE, pp. 89-96.

[13] C. Zaniolo, "Design of relational views over network schemas," *ACM-SIGMOD Conf.*, 1979, pp. 179-190.

執筆者紹介 Arnon Rosenthal

1969年に Polytechnic Institute of Brooklyn より数学で M. S. を，1974年に California 大学 Berkeley 校より Computer Science で Ph. D を取得．1980年まで，Bell 究所研と Network Analysis Corp に勤務し，同時に，Michigan 大学で教鞭をとる．現在は，Sperry Research Center のテクニカル・スタッフ．従来，離散的アルゴリズムを研究していたが，現在，データベース管理，とくに問い合せ処理とデータベース・システムのアーキテクチャの研究に従事している．

David Reiner

1972年に Cornell 大学より数学で A. B を，1980年に Wisconsin 大学より Ph. D を取得．1980年より Sperry Research Center のテクニカル・スタッフ．IEEE 発刊の Database Engineering の編集にたずさわり，問い合せ処理やデータベース・システムのアーキテクチュア，パフォーマンス評価，オペレーティング・システムに関心を持っている．

報告 デザインとコードにおけるウォークスルーの効果

The Effectiveness of Design and Code Walkthroughs

J. J. Hart

要 約 構造的ウォークスルーをデザイン・レビューやコード・レビューに使用した場合の評価については数多くの議論が行われているが，それらの効果を定量的に議論した例は少ない．本稿は，Sperry 社における Programmers Advanced Debugging System (PADS 1100) の開発時に行ったデザインとコードのウォークスルーにおいて，観察された効果について論じる．とくに，ウォークスルーによって認められたいくつかの非統計的な効果を論じ，一つの非常に有意義な定量的効果を示す．

Abstract There has been much discussion of the value of using structured walkthroughs to do design and code reviews, but there have been few quantitative measurements published to show how effective they are. This paper discusses the benefits seen from doing design and code walkthroughs during the development of the Programmers Advanced Debugging System (PADS 1100) at Sperry. In particular, it discusses some non-statistical benefits achieved by the walkthroughs and presents one very significant quantitative result.

1. はじめに

本稿では，システム・ソフトウェア・プロダクトを開発する際に行った，デザインとコードのウォークスルーの利用で認められた効果の事例を述べる．“ウォークスルー”という用語は，E. Yourdon の文献[1] において論じられている“同僚グループ・レビュー”のやり方を指す．本稿においては，用語“ウォークスルー”と“レビュー”をとくに区別して使用していない限り，同じ意味に用いる．

Programmers Advance Debugging System(原始プログラムの対話型デバッグ・ツール)の初期出荷版の開発における詳細デザインとコード作成の，それぞれの段階に対して構造的ウォークスルーを行った．

ウォークスルーとは，製品の品質，拡張性および保守性を向上させるために用いる技法と道具を一体化したものである．製品の一部は，ウォークスルーに依存せず，他の複数の手順や技法に依存している．このことによってウォークスルーの効果について，いくつかの客観的結論を引き出すことができる．

さらに，技術的知見やデザイン技法やコーディング技法の上達などの間接効果に加えてウォークスルーには，以下のようないくつかの重要な直接効果が認められた．

1) 誤りを早期に検出できた．
2) モジュール間インタフェースと広域データのコミュニケーションがうまくいった．
3) 詳細デザイン文書が段階的に作成された．
4) コードが統合された後でも，発見した誤りの修正は容易であった．
5) レビューした部分（製品の 90 パーセント）について報告された誤りは，調査期間中に報告された重大な誤りの総件数のわずか 25 パーセントにとどまった．

本稿では，まずわれわれのプロジェクトや使用した構造的技法についてその概略を述べ，つづいてデザインとコードのウォークスルーの利用について詳しく述べ，さらに達成された結果について論じる．

2. プロジェクトの全体像

今回の事例研究の主題は Sperry 社の Programmers Advanced Debugging System (PADS 1100) である．このシステムの詳細については，すでに述べてある[2]．PADS 1100 は，(原始のプログラムのデータ名や手続き名などの) 名前づけにより，参照可能な高級対話型原始プログラム・デバッグ・システムである．なお，参照は原始プログラム情報 (PADS Symbolic Debugging Dictionary) を生成するシリーズ 1100 の言語プロセッサによって翻訳される任意のプログラムに対して成される．このように PADS 1100 は，とくにデバッグの計画を立てる必要がなく，最小の経費によってデバッグができるように設計されている．

PADS 1100 は（バッチ・モードと対話型モード；8進数番地づけと原始プログラム番地づけ；割り込みに対する罠の設定；データ項目への参照と内容の変更に対する罠の設定；機械語命令と原始プログラムの命令の両方に対する追跡；機械語命令と原始プログラムの命令の両方に対する指定命令数分の実行；条件指令文の実行；一連の指令文から成る手続き処理機能といった）標準化され，拡張されたデバック能力を有している．PADS が多種言語プログラム処理機能を有するのは，PADS が Universal Compiling System[3] (UCS: 共通翻訳システム) の一部分であるという事実を反映している．なお，PADS 1100 は価格分離ソフトウェア・プロダクトである．

本事例研究は，PADS 1100 の初期出荷までの開発，検査および顧客先への納品までの一年半の期間を調査対象とする．この期間は一年間の開発期間を含んでおり，この期間には詳細デザイン，製作，検査，および以降の版に組み入れるいくつかの高水準設計が含まれる．また，この検査期間には，複数の他のプロジェクトのプログラマたちによる2か月間の実使用テストも含まれている．それからこの出荷版は，4か月間の独立検査と製品評価の期間に入った．さらに，顧客先による2か月間の使用が，このウォークスルーの実施効果を評価するためのデータとして含められている．

この期間中，このプロジェクトには最大6人構成で，平均すれば 3〜4.5 人のプログラマが働いていた．各プログラマは，少なくとも3年以上のプログラミング経験を持っていた．初期出荷版の PADS 1100 は，165 個のコード・モジュールと 180 個のデータ・モジュールから構成されており，約 23,000 のソース・ステートメントを含んでいた．

PADS 1100 は，常に非常に高い信頼性と高い安定性を示す製品となるように配慮されていた．PADS 1100 の正式な誤り報告が始まったのは，ほぼ開発期間の終了時期からである．この時点で他のプロジェクトからプログラマたちが PADS の使用を開始した．その後の8か月間で，開発者自身からや，システム・テストや評価テスト，さらに初期導入の顧客からの重要な誤り報告はわずか 20 件であった．主要な結論を引き出す根拠とするには，この誤り件数は少ないように見えるかもしれないが，製品のレビューを行った部分と行わなかった部分との対比，および製品内の誤りの分布は，レビューの効果の重要な指標となる．このことが本稿の論旨である．

誤りの平均修正時間は1日であり，この修正作業には，プログラム変更制御プロセッサの使用時間と退行テストに要する時間も含んでいる．

2.1 ドキュメンテーションと新しい道具

開発段階の期間中には，以下のものから成る多数の文書が作成された．

1) UCS のデザイン・レビューとコード・レビューの基準書
2) 広域データ記述を含んだ製品デザイン記述書
3) 各モジュールに対する詳細な技術文書
4) 使用者ガイドおよびプログラマ向け解説書
5) 導入ガイド
6) リリース・メモ

開発段階において，今までだれも使用したことのないツールのために，文書作成要求やプロジェクトの構成要員が受けた影響は大きなものであった．

PADS 1100 プロジェクト・チームにとっての新しいツールとは，以下のものであった．

1) ドキュメンテーション・プロセッサ
2) デザイン・データベース・プロセッサ
3) 疑似コードとそのプロセッサ
4) UCS のドキュメンテーション標準
5) UCS のデザイン・レビューとコード・レビューの標準
6) 機械語レベルのデバッグ・ツール
7) プログラム変更制御プロセッサ
8) 検査ベース管理プロセッサ
9) 新メモリ装備能力
10) 各種の出荷工程ツール

2.2 一般的な構造的技法

ウォークスルーに加えて，PADS 1100 の開発には構造化プログラミング，トップダウン・デザイン，非自己本位プログラミング，さらにトップダウンのコード検査とその統合，などの多数の構造化技法が用いられた．

それらの技法とウォークスルーは，グループのだれ一人として正式な訓練を受けていないにもかかわらず採用された．これらの技法の知識は実地の経験，雑誌の記事，会議の会報および他のソフトウェア開発者たちとの討論などにより獲得された．したがって，これらの技法の導入についての気運は，PADS 1100 のグループ内部より現れたと言える．

このプロジェクトに特定した諸アプローチなどは，詳細デザインの開始以前にグループ内において論議されたが，これらのアプローチは製品の定義段階と同時に早期から実行するように拡張された．

比較的短期間に安定した製品が作成された．開発には多数の内部用文書や，外部用文書の作成が含まれる．その上，グループ構成要員は使用経験のない多数のツールの使い方も習得しなければならなかった．それでもすべての使用した構造的技法は，それらを習得しなければならないという否定的な影響度合を最小化し，かつ安定的で拡張性のある製品を生み出すのに有効な文書の作成に役立った．ウォークスルーを除いて，プロジェクト全体は同一の文書化要求，ツールおよび諸技法を用いることになった．

3. デザイン・ウォークスルーとコード・ウォークスルー

PADS 1100 のほとんどの部分のウォークスルーは，デザインとコードのレビューに使用された．そして，初期出荷版の全開発期間を通して実行された．この節では，ウォーク

スルーを行う対象，使用手順，および直面したいくつかの問題点を述べる．

3.1 対象の設定

グループでは，ウォークスルーが適合するいくつかの1次対象と2次対象を設定した．1次対象は誤りの検出とグループ内のコミニュケーションであり，2次対象は効率のよい改良と標準との合致である．誤り検出は，任意のタイプのレビューを実行するのに第1に適した対象である．グループでは，モジュール間とモジュール内部のインタフェース・レベルの両方における誤り検出に関心を持った．

ウォークスルーのもう一つの1次対象は，コミュニケーションの改善であった．すなわち，誤りが製品中に生じる前に，ウォークスルーの段階において仮定を互いに確認し，誤解を解消するということが重要であった．製作中のよりよい共通相互理解で，保守費用を軽減することが予想された．

ウォークスルーの2次対象は，デザインとコードとの両方において，効率を改善すると共に標準化することであった．ソフトウェアは“正当で”効率がよくなければならない．ここで，正当性とはソフトウェアの使用という観点において定められる．もし，アルゴリズムまたはコードが正当でかつ有効でなければ，グループではこれを“検出された誤り”として考えた．（このことは，1次対象の一つを満足させる．）2次対象である効率的改良とは，正当でかつ有効なモジュールを微調整するといった改良である．標準との合致は，UCS のデザインとコードのレビューの標準と，そのグループ内での拡張への合致を意味していた．

3.2 ウォークスルーの手順—全期間にわたって

使用したウォークスルーの手順は，開発段階における多数の段階を通して変化した．詳細デザインの早期の諸段階においては，ウォークスルーはより形式的に行われ，終わりの諸段階に向かうに従がい非形式的になった．しかし，いくつかのウォークスルーの手順は，次のように，全体の開発段階を通して共通であった．

1) ウォークスルーは，個々のモジュールのレベルで行われた．
2) 個々のデザインのウォークスルーは，モジュールの技術文書および全体の製品デザイン文書のすべての関係するデータ記述と節にもとづいて実行した．個々のコードのウォークスルーは，コード自体に加えて，その技術文書と関係するデータ記述にもとづいて実行した．
3) 誤り修正はウォークスルー中ではなく，ウォークスルー後に行った．
4) 代替のアルゴリズムについて，いくつかの論議も成された．
5) 誤り修正の検証は，調整役によって行われた．
6) 前の版がレビューされ認可されたあとで，デザインとコードに大幅な変更があったときには，モジュールの再レビューを行った．個々のプログラマに，再レビューが必要か否かを決定する責任が与えられていた．一般に，プログラマたちはモジュールの再レビューを切望した．それは，再レビューの有効性があまりにも明白であったからであるし，他人の意見に対する健全な関心が育成されていたからである．（しばしば再レビューを行ったひと組の複雑なインタフェース・ルーチンがあった．その結果，グループは“déjà revu”〔とにかく，何度でも見る意〕という言葉を作り出した．）

3.3 ウォークスルーの手順—初期段階

この時期には，純粋な形で構造的ウォークスルーが行われた．前述したすべての段階に共通する手順に加えて，ここではウォークスルーは次の手順が行われた．

1) 与えられたモジュールのデザインとコード・ウォークスルーについて分離した会合を持った．すなわち，デザインをレビューし，プログラマはデザインをコードから分離して開発した．その後，コードのレビューを行った．

2) デザインがウォークスルーにもとづいて認可されるまで，コーディングを開始しなかった．

3) ウォークスルーのためのモジュールの計画は，製品の構造におけるそれぞれのモジュールの位置にもとづいて厳密にトップダウンで計画した．

4) プロジェクトの構成員は，各レビューを受けるように要請された．

5) スーパーバイザを含めたグループ全体は，各ウォークスルーに参加した．調整役と書記役は，それぞれの会合のつど輪番制とした．

6) 修正すべき項目および実施すべき変更点は正式は，“アクション・リスト”に保存された．

7) ウォークスルーの会合は，前もって計画され，あらかじめ取り決めた場所において開かれた．

8) 詳細技術文書，関連するデータ記述および必要に応じて他の文書から成るモジュールの文書は，通常ウォークスルーの開会の2日前にそれぞれの参加者に配付された．

3.4 ウォークスルーの手順—最終段階

これは，ウォークスルー手順の中の終わりのいくつかの段階の意味である．この最終段階では，いろいろな変更が加えられた．ある変更は，レビューをあらためて組み立てるというより，トップダウンでレビューするといった小さな変更であった．ある変更は，より重要なもので，開発段階の終了間際に開発されたモジュールに対して，デザイン・ウォークスルーとコード・ウォークスルーを組み合せて同時に行うといったことであった．ウォークスルー手順の最終段階は，以下のように特徴づけられる．

1) 単一のモジュールに対するデザインと，コードが同時にレビューされた．

2) デザインが認可される前に，コードを作成することが認可された．

3) モジュールは，所与の一つの機能単位内において，一般的にトップダウンでレビューが行われた．ボトムアップのレビューは勧められなかったが，時として行われていた．ボトムアップのレビューが認められるときは，非常によく定義されたインタフェースを持つルーチンとタスク（たとえば，データ変換ルーチンなど）に限ってレビューした．

4) ウォークスルーに対していくつかの例外が認められた．

5) ウォークスルーのグループ構成員はプロジェクトの一部の人たちであり，モジュールとこれとインタフェースを持つモジュールとの両方の作成者から構成された．スーパバイザ自身の作業をレビューする時を除いて，調整後は常にスーパバイザであった．

6) 正式な“アクション・リスト”は保存されなかった．

7) 正式のウォークスルーの会合は成されなかった．おのおのの参加者は独立にレビューを行った．

8) （技術文書やデータ記述など）モジュールの文書は，レビューを行う人間の間を回覧の形で配付された．レビュー・コメントはリスト上か別の紙の上に記入された．

3.5 問 題 領 域

ウォークスルーを実施するときには遭遇した若干の問題がある．これらは，新しいツールや新しい技法の影響，予定日限のプレッシャー，同時ウォークスルーにともなう問題，

回覧方式のレビューにともなう問題，およびウォークスルーに対する例外などである．

3.5.1 新しいツールや新しい技法の影響

前述のようにPADS 1100 グループでは，未知の多数のツールや技法を使用しなければならなかった．ウォークスルーのように，その中のあるものは開発部間にとっても未知のものであった．

このような状況は，プロジェクトに多数の影響を生み出した．とくに，ウォークスルーに対する影響として，グループがその使用法を習得する余裕は限られたものであったから，きちんと習得するための負担は大きかった．

3.5.2 計画の進捗負担

プロジェクトが進むにともなって，プロジェクトの最終期限を守るというあたりまえの負担があった．

ウォークスルーの効果が明白に実証される以前に，ウォークスルーは，無視できないかなりの時間を必要とするから，計画の進捗においてウォークスルー自体が，注目されるのは自然なことである．

ウォークスルーを行おうという方針は，プロジェクト管理側からではなくて，グループ自身から出たものであったから，ウォークスルーの実施に対する負担は増加した．

最終期限が近づくにつれてウォークスルーを急がせ，ついにはウォークスルーを“一時中断する”という負担となった．

トップダウン処理を続けるに従い，モジュールの複雑さは減少し，またデザイン文書は詳細になって，コーディングは事実上単純な翻訳になることがわかったから，グループではウォークスルーをやや非形式的に行うことを決定した．したがって，グループではデザインとコードのウォークスルーを同時に行うことにした．

のちに，負担がさらに高まったとき“回覧方式”のレビューを開始した．これら変更したウォークスルーの手順は，ある程度の役割を果たしたが，つぎに説明するようなある不利益をもたらした．ある形態でのレビューは開発段階全体を通して行われた．

3.5.3 同時ウォークスルーにともなう問題

デザインのウォークスルーとコードのウォークスルーを分離して行うと，デザイン上の誤りをコードのウォークスルーを行っている最中に発見することができた．しかし，デザインとコードの同時ウォークスルーを行うと，批判者はデザインを調べる2回目の機会を失った．

さらに，デザインとコードの同時ウォークスルーを行った場合，以下の三つの傾向が生まれた．

1) すでにコード化されているから“正しいに違いない”という先入観により，デザインを綿密に検査しないという傾向．
2) 一度コードを作ってしまえばモジュールが変更されることはないであろうという理由で，代替案を提示しなくなるという傾向．
3) 批判者が，こうすべきだと考えたことをコードが実現しているか否かをチェックすることに専念して，そのコードが本来のデザインと正しく対応しているかを調べることを忘れるといった，コードのレビューのみに集中する傾向．

3.5.4 回覧方式のレビューの問題

デザインとコードの同時ウォークスルーを開始した後にも，計画の予定期限のプレッシャーは増大した．

そこで，残ったモジュールの複雑さは減少したので，“回覧方式”の手順に変更することに決定した．レビュー資料として一部のコピーを作り，このモジュールをレビューするために選ばれたすべてのプログラマ達へ回覧した．批判者達は，自分のコメントと質問をリスト上に直接記入したり，別の紙に記入して他のレビュー者にその資料を渡した．この回覧資料は“調整役”であるプロジェクトのスーパバイザに戻し，それから作成者に戻された．

ときには，レビュー者達は直接作成者にコメントを与えることがあった．

この方式による問題点は，以下のとおりであった．

1) 重要な意味があったにもかかわらず，作成者を探し出して曖昧な点について質問しようとしないという傾向があった．

時には質問が作業者宛にリスト上，または別の紙の上に書かれたが，作成者へフィードバックすることが徹底していなかったから，あとの伝達をやめてしまった．

2) 各レビュー者は，かなり独立して仕事をしており，探し出してまで他人を助けようとはしないから，やっかいな問題点を取り上げてまで“問題を起こす”ことをいやがる傾向があった．

3) モジュールの中にひそんでいる，いくつかの重要でない誤りは検出されなかった．

この“回覧方式”のレビューはもはやウォークスルーではないが，それでもある程度の目的を達した．

3.5.5 ウォークスルーに対する例外

ウォークスルーに対する例外は，この技法の効果を調査する上で役に立つが，好ましくないいくつかの副次効果をもたらした．

1) レビューを行わなかった部分の保守や拡張は，レビューを行った部分に比べて，かなり困難である．

2) 仕事のレビューを受けたグループの人員は，“摘出された”と感じることがある．つまり，何の見返りもなく自分の仕事だけが，よりきびしい基準に合致するようにとり出されたと感じるのである．

3) はじめに“わざと分散された”チームの構造は壊れた．

プロジェクトのグループは，レビューを行う人と行わない人の対立したグループに分割された．

例外を認める感情的反抗は少し意外なことであった．しかし，グループ構成要員全体は，ウォークスルーの効果を見い出したことによって満足した．

4. ウォークスルーの結果

ウォークスルーの結果は，非常に肯定的であった．

次のいくつもの重要な直接効果があった．

1) 誤りは，早期に検出された．

2) インターフェースと広域データの使用法は，よく伝達された．

3) 詳細デザイン文書は，段階的に作成された．

4) コードが統合された後でも，発見した誤りの修正は容易であった．

5) 調査期間中に報告された重大な誤り件数の内の75パーセントは，レビューを行わなかった製品全体の10パーセントの部分の誤りであった．

以下の節において，それらの直接効果を詳細に述べ，レビューから除外した作業との対

比を述べる.

4.1 早期の誤りの検出

製品の中に誤りが埋め込まれる前に，製品のレビューを行った部分について多くの誤りが検出できた．もちろん早期の誤りの検出は，ウォークスルーが有効性として幅広く論ぜられている[1],[6]．誤りは，単にウォークスルーの準備中，あるいはウォークスルーの会合で検出できたのではない．ウォークスルーの手順における次の四つの異なる場面で，実際に発生した.

1) 資料がウォークスルーに出される前に，ある誤りはプログラマ達によって取り除かれた.
2) レビュー者と作成者双方が，ウォークスルーのために準備する間にある誤りは検出された.
3) ある誤りは，ウォークスルーの作業中に検出された.
4) デザインの誤りが，コードのウォークスルー中においても検出された.

これとは対照的に，ウォークスルーをしない資料は机上の検査，個別に行うインタフェースの検査，そのモジュールによく知らない人々の自発的な質問，および徹底的な検査などの旧来の方法による誤り検出を当てにしている.

4.1.1 ウォークスルー前の誤りの除去

ウォークスルー中に検出される誤りの数は，数回のウォークスルーを行っただけでも相当な数になる．これは，部分的にはウォークスルーに資料を提出するということで，プログラマいつもより，さらに注意深くが曖昧な点を明瞭にし，いろいろな仮定をチェックするからである．さらに，レビューの参加者は，彼等がレビューをしている資料で発見する同じ型の誤りを，自身の作業の中でも探し出すことで二重の検査を行うことになる.

4.1.2 ウォークスルーの準備中における誤りの検出

ウォークスルーの準備段階における批判者の作業については，文献[1],[6] に論じられている.

PADS 1100 グループで実施したウォークスルーでは，一般に批判者達はアルゴリズムの正当性をチェックし，データ記述に照してデータの使用法をチェックし，インタフェース記述と対照にインタフェースをチェックし，そして標準に照して資料をチェックした．コードのウォークスルーでは，批判者達は，さらに文書化したデザインと対応させてにコードをチェックした．もちろん，いくつかの誤りはこの過程で発見された.

上の文献には，作成者もウォークスルーを開くに先立ってレビュー資料を再チェックし，誤りを検出することは述べられていない．たとえば，PADS 1100 グループでは，資料を提出して，質問に答えられるように資料の内容を正確に記憶するために，作成者は自分の資料をしばしばチェックした.

多くの場合，資料に対する新しい物の見方によって，作成者はウォークスルーの開始前に文書の誤りと論理的誤りの両方を検出できた.

4.1.3 ウォークスルー中の誤りの検出

ウォークスルーの準備中に検出する誤りに加え，ウォークスルーの最中にも誤りのあるものは検出された．場合によっては，作成者の詳細な説明によって，批判者達が気づかなかった仮定が発見されるからである．その他の場合には，誤りはグループとしての会合の共同の効果とされた．すなわち，一人の参加者による質問や，問題提起が他の参加者からの質問を引き出させるきっかけとなった．そうでなければ，他の参加者は何も質問しな

かったろう．そして多くの場合に，継続した議論によって検出されなかったかもしれない誤りが発見された．

4.1.4 コードのウォークスルーにおけるデザイン上の誤りの検出

デザイン・ウォークスルーとコード・ウォークスルーを分離して行うと，批判者達も作成者も，デザイン上の誤りを発見するための2度の機会を持つことになる．ときどき，誤りがデザイン・ウォークスルーをすり抜けることがあるから，2度目の機会は有効であるということがわかった．さらに，デザインを新鮮に見渡すと多くの情報が使えるから，批判者達は時として多くの誤りを検出することができる．作成者も，コードを作成した時よりもデザインを新鮮に見渡すことができる．

4.2 コミュニケーションの改良であるインタフェースと広域データの伝達

ウォークスルーの1次対象の一つは，コミュニケーションの改良である．この改良は確実に達成できた．明白な形で，すべてのプロジェクトのグループ間においても，インタフェースと広域データの使用法は，ある程度相互に伝達された．しかしながら，ウォークスルーによって確実に，コミュニケーションがグループの過程となり，インタフェースと広域データの使用法がコードに正確に反映できるまで実施され，かつ使用法がプロジェクト全体を通して一致した．さらに，批判者の出した質問によって，しばしば誤った仮定が発見された．

ウォークスルーの各会合は，コミュニケーションの始めではなく，またそれだけでもない．ウォークスルーの会合の予告がなければ，コミュニケーションは広がりがなかったであろうし早くならなかったであろう．実際，製品の中のレビューを行っていない部分の作業をしている人々は，ウォークスルーの参加者達が任意のレビューずみコードに関するインタフェースの質問に答えることができることをしばしば発見した．

4.3 段階的デザイン文書化

“われわれのプロジェクトの成すべきことは終えた．　今，われわれは資料を作成している．”と，過去ソフトウェアの開発者が何度述べたであろうか．

ソフトウェアのデザイン初期には，各プログラマが，文書内容をかなりよく更新していることがしばしばある．しかし，開発が進み，いろいろな負担が生じると，デザインの新しい部分はきちんと文書化されないか，まったく文書化されず，またデザインの古い部分も更新されない．製品の作成が“終わる”と，デザイン情報は通常貧弱でかつ古くなってしまった文書を組み合せた最終版の文書から導き出さなければならない．しかも，時間も金もエネルギーもすべてが低いレベルになった終了時点で導き出すことになってしまう．

適切なドキュメント標準と共に組み合わされているウォークスルー（と文書化を支援するウォークスルーの要求）は，モジュール・レベルの詳細デザイン文書資料の段階的な作成を強力に支援する．重要な変更が生じたときのモジュールの再レビューは，現存の文書の保守を支援した．したがって，開発段階の終了時には，プロジェクトはコードだけを持っているのではなく，レビューの過程の副次効果としての詳細なデザイン文書をも保有していた．他方，製品の中のレビューを行わなかったほとんどの部分についての詳細文書は，製品の中のコードに組み込まれた後で作成された．しかも，その文書化では従来どおりの問題すべてを証拠立てることになった．

4.4 誤り修正の容易さ

各報告が到着すると，グループは大いにエキサイトした．報告会が開かれると，ウォークスルーに参加した人は，自分が関係した機能部分をただちに識別できた．また，そのモ

ジュール，またはモジュール群をしばしばすばやく識別できた．ある作成者がその場を離れたときでも，彼が戻るまでに残った者がしばしば彼の部分を識別できた．これは，ウォークスルーの他の結果である誤り修復の容易さについて，いくつかの点を例証している．相対的な容易さは，次のように分類される．

1) 誤りのあったモジュール群はすばやく識別できる．
2) 1人以上の人が解決策を示唆することが可能であった．
3) 修復が，どのようにシステムの他の部分に影響を与えるかを各プログラマが理解しているから，修復に信頼がおける．

前述のように，製品のレビューした部分にある1件の報告された誤りを修正するのに必要な平均時間は1日であった．この修復時間には，プログラム変更制御プロセッサと退行検査の利用に要する時間も含まれている．他システムへの依存性や広域データの使用法について身につけた理解により，修復は一般に退行の原因とはならなかった．対照的にウォークスルーに参加しなかった人は，個別に問題を見つけて解決しなければならない．ウォークスルーに参加しなかった彼等がグループ内の他の人員の助けを得ようとすると，彼等はウォークスルーから早期に脱落しているから，二重の問題に落ち入ることになった．一つは，グループ内での孤立がコミュニケーションを妨げるという問題である．もう一つは他の人員がその特定の部分に詳しくないことから，意味のある入力を作ろうとすると多くの教育がいるということである．平均すると，レビューを行っていない製品の部分における問題は，非常に長い解決時間を要するし，また安定しない．

4.5 製品の90パーセントの部分が占める誤りの割合は25パーセント

前述のように，プロジェクト全体は同一の構造的技法を使用し，同一の新しい道具を使用し，同一の文書化要求に従った．ただし，製品の10パーセントの部分については，ウォークスルーも“回覧方式”のレビューもしなかった．しかし，レビューを行った残りの90パーセントの部分に対して重大な誤りの報告がなされたのは，誤り報告全体の25パーセントにすぎない．

レビューを行わなかったプロダクトの部分は，かならずしもレビューされた部分よりも複雑であるわけでないし，また作成者のスキルはほぼ同じであったから，結論としてウォークスルー（と“回覧方式”のレビュー）は誤りの分布を上のようにしたことになる．

5. おわりに

PADS 1100 プロジェクトは，デザイン・ウォークスルーとコード・ウォークスルーの実践の価値を実証した．プロジェクト全体は同一の文書化要求，新しいツールの使用，そして構造的技法の使用に従った．ただし，ウォークスルーに従わなかった製品の部分は除く．使用したウォークスルーの手順は，対象とした期間を通して変更された．そしてウォークスルーの手順におけるいくつかの問題に直面した．

ウォークスルーを実践した結果としての有意義な直接効果は，次のとおりである．

1) 誤りは早期に検出できた．
2) インタフェースと広域データの使用法について，コミュニケーションがうまくいった．
3) 詳細デザイン文書は，段階的に作成された．
4) コードが統合された後でも，発見した誤りの修復は容易であった．
5) 調査期間中に報告された重要な誤り件数のうち，レビューを行った部分（製品全体

の 90 パーセント）に誤りがあったのは，誤り件数全体の 25 パーセントであった．

（ソフトウェア・プロダクト統括部　ソフトウェア一部　白川　利夫 訳）

参考文献
[1] E. Yourdon, *Structured Walkthroughs*, YOURDON Press, 1978.
[2] J. Hart, "The Advanced Interactive Debugging System (AIDS)", *SIGPLAN Notices*, Vol. 14, No. 12, Dec. 1979. (ユニバック技報　第1号，8月，1981年)
[3] H.C. Gyllstrom, R.C. Knippel, L.C. Ragland, K.E. Spackman, "The Universal Compiling System," *SIGPLAN Notices*, Vol. 14, No. 12, Dec. 1979. (ユニバック技報　第0号，2月，1981年)
[4] Private Communication from the Current Supervisor of the PADS 1100 Project.
[5] M. Mantei, "The Effect of Programming Team Structures on Programming Tasks", *Communications of the ACM*, Vol. 24, No. 3, Mar. 1981.
[6] M.E. Fagan, "Design and Code Inspections to Reduce Errors in Program Development", *IBM Systems Journal*, Vol. 15, No. 3, 1976.

執筆者紹介　Jolene J. Hart
1972年に Minnesota 大学より数学で B.A. を，1978年に Minnesota 大学よりコンピュータ・サイエンスで M.S. を取得．1977年より Sperry 社でシステム・プログラミングに従事．PADS 1100 プロジェクトでは，言語システムの senior programmer を務めた．現在，CAD 部門のソフトウェア工学の Specialist.

訳 者 注　PADS 関連の日本語のマニュアルとしては，「ASCII COBOL 解説書　応用編（レベル5）」(資料コード 481206304-1) がある．

報告 高水準言語の下方拡張機能

A Facility for the Downward Extension of a High-Level Language

T. N. Turba

要 約 本稿では，電子計算機とオペレーティング・システムをアクセスするための，高水準言語の機能拡張方法について述べる．個々の特殊機能を利用するに当たり，コンパイラを修正する必要がない拡張方法である点を本質とする．アセンブラ言語に似た言語で拡張を指定し，コンパイル時に実行可能なコードに変換し，コンパイラが生成するコードに統合する．この機能は，あるシステム記述言語で実現されたが，この言語で直接扱えない機能をアクセス可能にする目的で設計された．この実現方法は，言語の目に見える変更を最小にとどめると共に，任意の言語でコード列を生成する目的に適合している．拡張機能の作成者は，高水準言語のコンパイル時に可能な情報をアクセスし，拡張機能を使った文脈に応じてコードを選択・生成でき，そのコードをコンパイラが生成したコードに統合できる．また，誤まって用いた場合には理解しやすいエラー・メッセージを出力させる能力も備えている．

Abstract This paper presents a method whereby a high-level language can be extended to provide access to all the capabilities of the underlying hardware and operating system of a machine. In essence, it is a facility that allows a user to make special purpose extensions to a language without requiring the compiler to be modified for each extension. Extensions are specified in an assembler-like language that is used at compile time to produce executable code to be combined with compiler-generated code. This facility has been implemented in a systems-programming language and was designed to provide access to facilities not directly available in the language. The way in which the facility was implemented calls for a minimum of user-visible language changes and is well suited for generating code sequences for any language. The facility provides the extension writer access to information available in the high-level language during compilation, permits the selective generation of user-defined code sequences depending on the context in which they are being used, provides for the integration of this code with compiler-generated code, and provides for the generation of user-understandable error messages when an extension is used incorrectly.

1. はじめに

多くの高水準言語は，上方向への拡張機能を備えている．通常，この拡張性は新しいデータ型とそれに対する演算を定義する特別な言語構文として用意されるか，あるいはこれをシミュレートするマクロ機能によって用意される．いずれの場合にも，拡張は新しいデータ型とその演算を，言語の基本機能を用いて定義することにほかならない．まさに，これが原因で拡張可能言語は，本来備えている基本機能以上の能力を追加できない．拡張可能言語が与えてくれるのは，応用分野に適したデータ型を用いて，より自然な方法で演算を施せるということにすぎない．そのこと自体は，プログラム言語における価値のある目標であり，大きな一歩であっても決して過小評価すべきではない．しかし，拡張可能言語は言語に何ら新たな能力を付加するものではなく，既存の機能の書き方の変形を許すのみである．演算によっては，拡張可能言語では記述不可能であったり，極端に効率が悪くなったりする．とくに，新しいデータ型に対する新しい演算を多く用いるプログラムで

は，本来の演算に比べて効率が悪くなるのが普通である．

古くは ALGOL から，新しくは Ada に至るまで，多くの言語はプログラム中にアセンブラ言語，あるいは機械に依存したコードを含めることが必要であると考えられ，言語の定義文書中にそのような機能の必要性を記載している．この機能を用意する理由のいくつかは，次のようなものである．

1) オペレーティング・システムへの要求を，言語本体に組み込むことなく使用可能にする．
2) コンパイラが通常生成することのない，特別なハードウェア命令を使う．
3) 高水準言語で書かれたプログラムが，他の言語（通常アセンブラ）で書かれた既存のコードと，インタフェース・ルーチンの介在なしでインタフェースをとる．

このような理由と要望にもかかわらず，そのための実際の機能も，コンパイラの生成コードとの統合の方法もほとんど定義されていない．定義がないということは，疑いもなく一つにはこの種の機能を持つコンパイラがほとんど存在しないこと，またあったとしても限られた機能しか備えていないことに原因があろう（ただし，この種の機能を持つコンパイラの多くはメーカの専用言語であり，一般には文書が公開されていないので，断定は不可能である）．

本稿では，使用者が定義したコード列をコンパイラの生成コードに統合する機能について，一般的な考察をする．そして，Sperry 社の シリーズ 1100 の高水準システム記述言語PLUS (Programming Language for UNIVAC Systems) (訳注) において実現された機能にもとづいている．この機能は最初の実現以来，単にシステム機能をアクセスする命令語を直接挿入するという段階から，文脈に応じて洗練されたコード列を選択し，コンパイラの生成コードに効果的に統合する段階へと，幾多の改良が加えられた．

2. インタフェースの場所

アセンブラ言語に似たコードを入れる機能を設計するに当たり，そのコードをどこで許すかをまず考えなければならない．理想的にはプログラムの中の任意の場所に挿入できるようにするのがよいが，これは実現上問題がある．また，同じくらい重要なことであるが，使用者インタフェースもよく定義できないという欠点もある．

つぎに，プログラム内でのコード列の使用頻度を考えなければならない．ここで述べるシステムでは，多くの場所で繰り返し使われるコード列を考えた．したがって，コード列の定義を一度だけとし，あとで何度でも使えるようにする方法を採用した．

アセンブラ言語に似たコードを使用するインタフェースの場所については，言語の論理的な構文単位と対応させねばならない．構文単位には，手続き，関数，演算子，文などがある．PLUS システムにおいて選んだ構文単位は，手続きの呼び出しと関数の参照である．この言語には演算子や文に対する拡張性がないので，これらの構文単位は除外した．ただし，演算子や文をインタフェースの場所に含めても，論理上の差異は生じないであろう．

アセンブラ言語に似たコードを入れるための宣言は，この言語の手続き宣言や関数宣言と同様である．形の上での違いは，使用者定義コード（インライン・コードと呼ぶ）での呼び出し時に引数の個数や型が異なってもよいような汎用手続きや汎用関数も許されることである．このような汎用性を普通インライン機能に含める必要はないが，含めるとすれば特別な考慮が必要である．たとえば，引数の個数と型を高水準言語の中で指定したり検査したりすることはできない．この種の検査は，インライン・コードを書く人が，低水準

のコードの特別な機能を用いて行う．高水準言語での宣言の形は次のとおりである．

INLINE 名前；

および，

INLINE 名前 RESULT 型；

2番目の形式のように，名前と共にインライン関数の型を指定する必要がある．見方によれば，引数として汎用の型が受け取れるのに，汎用の型を返せないのは矛盾していると考えられよう．しかし，この高水準言語では関数について強い型付けを行っているので，この制約はこの言語自体と整合性がとれている．一方，インライン用の言語には強い型付けがないので，引数を高水準言語から低水準言語へ渡すときに，型の検査を行わないのが自然である．

3. 情報の受け渡し

アセンブラ言語に似たコードの挿入を一般的に有用なものとするためには，高水準言語から低水準言語へ情報を渡す手段が必要である．情報の受け渡しは一般性を持つべきで，インライン・コードを使用する場合，コンパイラが持つ情報のすべての問い合せを可能とするのが理想であろう．しかし，PLUS システムの設計では，一般性は重要とは感じられなかった．したがって，インライン使用時の引数に関する情報だけを得ることができる方法を採用した．これは，インライン言語中で一群の問い合せ用組込み関数を用いて可能となる．これらの関数では次の情報を検索できる．

1) インライン使用時の引数の個数
2) 個々の引数の型（整数型，実数型，文字型など）
3) 引数の類別（定数，自動変数，静的変数など）
4) 引数参照用情報（番地，大きさ，ビット，オフセットなど）
5) 引数の可用性状況（レジスタ中にロードされているかどうかなど）

これらの関数は情報をアセンブラ水準のコードに引き渡す．インラインを書く人は，呼び出し時の引数に関する情報を利用して，引数に応じてコード生成を変化させることができる．

4. コード生成

インライン言語中での機械語命令は，アセンブラ言語によく似た形で指定する．すべての機械語命令が指定できるが，マクロのようなアセンブラ言語特有の複雑な機能は含んでいないので，アセンブラよりは簡単な言語となる．また，通常アセンブラ言語が持つ入口点や静的変数の定義機能も存在しない．インライン・コードは一般的にいって，プログラム全体ではなく，コンパイラ生成コードに統合させるための直線的で短いコード列から成るので，この単純性は容認できる．インライン言語で，アセンブラ言語と同じ働きをする部分は，アセンブラ言語と同じ文法規則に従うので，プログラマがそのために別の言語を学ぶ必要はない．問い合せ関数のような新規の機能については，アセンブラ言語の似た機能と整合した形式を採用している．これらの関数は，引数の性質に依存する命令語に情報を与えたり，実際の生成コードを決める文を制御するために用いる．

コード生成を選択する基本的な制御構造は，条件式の結果にもとづいてコードを含めたり，除いたりする機構である．条件式が真のときコードを生成し，偽のときは生成しない．この制御構造は，呼び出し時に渡される引数の型，個数，類別などに応じて，単一のインラインから異なるコード列を生成するために必要となる．PLUS システムで採用した

制御構造は，アセンブラ言語での似た機能に対応させた単純なオン・オフ機構である．これは高水準言語の IF-THEN と同様である．形式は次のとおりである．

ラベル ON　条件式

⋮ } 条件式が真のとき含めるコード

OFF

異なるコード列の選択は，コンパイラが異なるコード・スケルトンを選ぶ方法と同様である．インライン呼び出し時に渡される値に応じて，異なるコードを生成する．インライン機能の使用者は，コンパイラの可能なコード・スケルトン群を増やしているのだということができる．

条件付きコード生成のオン・オフ制御構造に加え，コード列の途中から抜け出て，残りのコード生成を抑制する ESCAPE 文も存在する．オン・オフ制御構造では入れ子構造が可能であるが，外側の ON 文に付けたラベルを参照すれば，何段かにわたって外に出る ESCAPE 文も許される．

PLUS システムの別の制御構造として，ENTER 文がある．これは局所手続きの呼び出しと同値であり，原始テキスト内で別のインラインを参照し，その場所において呼ばれたインラインのコードを生成する．その後，元のインラインに戻ってコードを生成を続ける．各インラインは，コンパイル時の変数，ラベル，その他の名前などに対し，局所的な有効範囲を備えている．

他の制御構造，たとえば CASE 文についての提案があり，今後付加することもできよう．

5. インタフェース指示

インタフェース指示は問い合せ関数と対を成すものである．問い合せ関数が情報をインラインにもたらすのに似て，インタフェース指示はインラインからの情報をコンパイラに戻す．両者は共に情報をやりとりし，生成したコードを周りのコンパイラ生成コードに効果的に統合する補助となる．

インタフェース指示の一例をあげると，インライン関数が返す値の場所についての情報を知らせるものがある．通常，関数は，あらかじめ決められたレジスタに値を返す．しかし，他のレジスタに値を返す方が有利な場合には，この指示で結果を含むレジスタを示せばよい．

他の例は，インライン・コード列の最後で，ある変数の値がレジスタ中にあることをコンパイラに知らせる指示である．コンパイラは後のコード生成で値をレジスタにロードする必要があるときに，この情報を利用することができる．

生成コードに関する指示に加えて，作業領域の共有や統合のための指示もある．この指示により，インライン・コードから手続きの自動スタック上に作業領域を確保する要求を出すことが可能になる．この領域はインライン・コードの実行中に中間結果や一時的な情報を蓄えるのに利用される．インラインからは，外部名を持つ静的な大域変数や参照呼び出しの引数の記憶域をアクセスすることも可能である．

6. レジスタ操作

すでに見たように，ここで述べるインライン機能は，操作する値をレジスタにロードする必要があるレジスタ構成の電子計算機上で実現されたものである．このような電子計算

機のコンパイラは，通常最適化を施してレジスタ中の値を再利用可能にしているので，インラインを呼び出しても最適化の防げにならず，またインラインで使用するレジスタを確保する何らかの手段を見出さなければならない．インラインでレジスタの退避と復元を行ったり，最適化を中断させたりする方法もあるが，効率を考えこれらの代案は見送った．

選んだ方法はインライン呼び出し時にいくつかの「自由」なレジスタを決め，コンパイラがこれらのレジスタをインラインの呼び出しをまたいで値を保持する目的で使わないようにすることである．自由レジスタは，28個の汎用および指標レジスタ中の7個と，16個の反復レジスタ中の4個から成る（すなわち1/4のレジスタが自由レジスタに含まれる）．こうしても最適化用には，十分なレジスタが依然残っている．インライン中で，自由レジスタより多くのレジスタや，ある特定のレジスタを必要とするときには，インライン・コード中で退避と復元操作を行わなければならない．

自由レジスタを決める方法は，レジスタの個数が多いことと，インライン・コード列が短いものであるため，PLUS システムにおいては十分受け入れられるものである．もし，レジスタの個数が少なくて，インライン・コード列が長いものであれば，別のより洗練された方法が必要となる．一つの方法は，インラインで使用するレジスタを宣言し，コンパイラがそのレジスタを使わないように考慮するものである．別の方法は，インラインでは一般的なレジスタを使っておき，コンパイラがそれを取りまくコードに応じて実際のレジスタに結びつけるものである．どちらの方法も受け入れられるものであり，試行された．インライン・コードをコンパイラ生成コードに統合するためのコンパイラの修正は，ほんのわずかなものであった．しかし，注意しなければならないのは，インライン・コードをあるレジスタ使用規約に従った既存のコードとインタフェースをとるために用いるのであれば，どのレジスタを使うのかインライン・コード側で指定できるものでなければならないことである．これは，システムの機能や既存ライブラリをインライン・コードでアクセスする場合に，よく起きる状況である．

インライン・コードが特定レジスタを予約しておく別の例は，実行時の環境を確立するときである．たとえば，スタックの保守や情報の通達用にレジスタを用いる場合である．このようなレジスタ中にインラインの呼び出しにまたがって値を保持し続けるときは，それをコンパイラに示して，使用を禁止する必要が生じる．これは，コンパイラに対して特定のレジスタを使用させないようにするインタフェース指示によって可能である．

7. エラー処理

これまでインラインのコード生成と統合に関してのみ考察したが，これは問題の一部にしかすぎない．インラインを書き，テストした後は実用の段階に入るが，しばしば誤まった使われ方をすることがある．たとえば，まちがった型や個数の引数を渡すことがある．そのようなエラーも考慮に入れ，対処しなければならない．組み込み関数の使用上のエラーと同質のエラー・メッセージを出すことがインラインに対しても望まれる．それが可能でないにしても，何らかのエラー・メッセージを出す機能を用意しなければならない．

PLUS システムでは，インライン言語の DISPLAY 文によってエラー・メッセージを処理する．この文は通常，いくつかの可能性を検査する条件文の最後の節として置かれる．インライン・コードを生成中に DISPLAY 文があると，あたかもコンパイラ自身がエラーを発見したかのように，出力リスト中にエラー・メッセージが出す．このエラー・

メッセージの前には，error, warning, message の区別を示す語，エラーを検出したインラインの名前，高水準言語中でインラインが使われた行の番号，DISPLAY 文が渡した説明的なテキストなどが置かれる．説明上のテキストはそれぞれの DISPLAY 文で変わり，インラインの誤まった使い方に関する情報，たとえば，どの引数が誤りを含むかを示す番号なども含む．

8. 実現方法の概略

PLUS システムでインラインを処理する方法は，著者の知るかぎり，他のシステムとは異なっている．多くのシステムでは，高水準言語の中に直接アセンブラ・コードが現れる．これは，通常，つぎに続くのは高水準言語ではなく，アセンブラ・コードであることをコンパイラに示す指示語によって成される．PLUS では，全インライン・コードを，高水準言語の外に置く方法を採用した．高水準言語で現れる文は，宣言と参照のみである．宣言は名前と，関数の場合には結果の型も定義する．参照は，通常の手続きや関数の呼び出しと同様に現れる．

インライン・コードは，高水準言語とは別の原始テキストとして書かれる．また，コンパイラが処理する前に，別のライブラリに入れておかなければならない．ライブラリは，原始テキストではなく，コンパイラが処理しやすい内部形式で維持される．内部形式への変換は，特別のプロセッサによって別途行われる．高水準言語中でのインライン名の宣言は，インライン・ライブラリと結合され，コンパイラが必要に応じて，ライブラリからコードを取り出す．高水準言語中のインライン名の参照ごとに，インラインで定義したコード列が生成される．前に述べたように，一つのインラインを呼び出しても，引数に応じて実際に生成されるコード列は変わり得る．

インライン・コードの呼び出しと，生成の状況は図1に示す．

インラインの定義と呼び出しを分けた理由にはいくつかあるが，次のように要約できる．

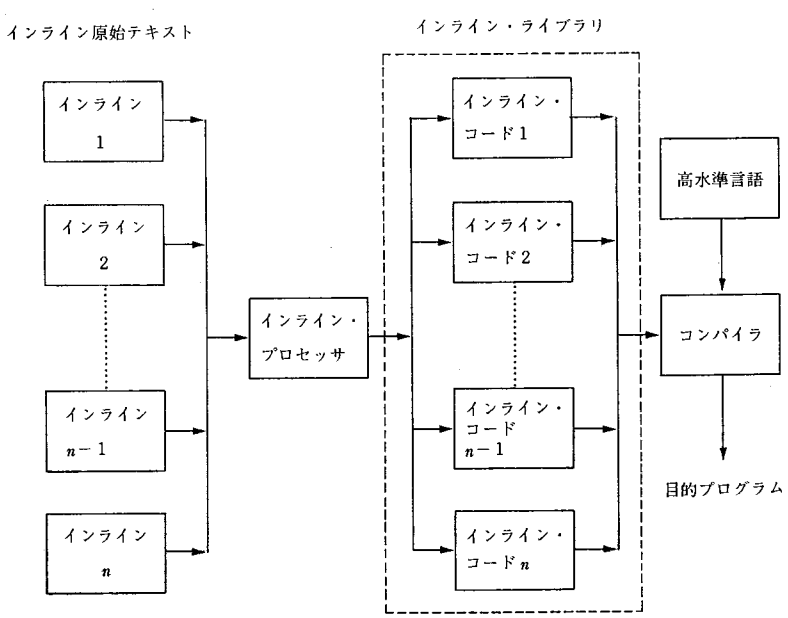


図 1 インライン機能の使用

Fig. 1 Use of the inline facility

1) インライン・コードと高水準コードを分離して書きたいという要望……これは，高水準言語の使用という利点を損わないために必要であった．
2) インラインの設計と使用を個別にではなく，プロジェクト単位で行うよう推進する要望……これをさらに推進するため，コンパイル時には，一つのインライン・ライブラリだけを使用するように制限した．
3) 使用者に不要な情報を隠す高位の隠蔽機能に対する要望……使用者にとって，知る必要があり，かつ見なければならないのは，インタフェースとなるインラインの宣言だけである．
4) 効率のよい実現方式に対する要望……内部形式で保持されるので，使用時に毎回インラインの定義を処理し直す必要がなくなった．
5) 別々にコンパイルされたモジュール間で，インラインを共有する要望……共通のインライン・ライブラリを持つことで，プロジェクト内，あるいはプロジェクト間で同じインラインを共有することができる．

さて，定義と呼び出しの分離に関して一つの欠点が明らかになった．インライン定義者は，一つのインラインを二段階でテストしなければならない．最初にインラインを内部形式に変換して，ライブラリに収める．つぎに，テストする高水準言語のコンパイルが必要である．しかし，定義は呼び出しに比べて頻度が少ないので，これが致命的な欠点というわけではない．別段階のテストを行うので，インラインのテストがより完全になるともいえよう．定義と呼び出しの分離に関する上記の理由に加え，他にも理由があるので，とくに述べておく．

1) 言語独立性……定義と呼び出しの分離によって，任意の高水準言語から使える単一の機能が生まれた．言語に，インライン機能を追加するといった少しの修正を加えるだけでよい．この修正自体は PLUS と共有することができるので，インラインを処理する別の解析ルーチンは不要である．もし，個々の言語のコンパイラに独立に組み込むとしたら，それぞれ別の解析ルーチンが必要となろう．コンパイラに施す重要な変更は，インライン・コードとコンパイラ生成コードを統合する後半の部分である．これも共通の方法で行うことができ，適切な設計が採用されていれば，同じルーチンを使うこともできよう．
2) 機種独立性……定義と呼び出しの分離により，機種独立性ももたらした．インラインを使用しているプログラム中には，機種に依存するコードはまったく含まれていない．インラインを使用するプログラムは，他の機種のもとでもインライン・プロセッサが存在していれば，移植可能である．これは実際，複数機種でコンパイラが存在する PLUS システムにおいて経験ずみである．しかし，インライン・プロセッサが存在していないとしても，定義と呼び出しの分離によりモジュール性が高まっているので，コードの移植はやさしいものとなる．PLUS システムの例でいえば，インラインの使用は手続きや，関数の呼び出しに置き換えるだけでよい．
3) 環境の設定……インライン・コードはコンパイラや実行時ライブラリの通常のコード生成規約を無視できないが，インライン・コードの使用により，特定のコンパイラの実現方法に束縛されない不変の実行時環境を作ることもできる．したがって，個々のプロジェクトで各々の実行時環境を設定してもよく，コンパイラで将来予想される変更からも自衛できる．

9. 内 部 表 現

コンパイラが処理するインラインの内部形式は，インライン言語の各命令に対応するインタプリティブ（解釈実行可能）な命令の集まりとして表現するのが最も都合がよい．インライン・プロセッサは，実質上，インタプリティブな命令を作り出すインライン言語用のコンパイラであるといえる．この命令はコンパイラが内在しているインタプリタによって実行される．実行により，コンパイル中のプログラムとインラインの双方の情報をもとにして，実際に生成されるコードが決められる．

インタプリティブな命令には，2種類の型がある．それは，コード生成用と，そうでないものである．コード生成用の命令は，インライン原始テキスト上で実際のコードを示す命令と一対一で対応している．他の命令は，引数の評価，値の計算，メッセージの作成，コード列の選択，コンパイラへの情報の提供などの仕事を行う．

10. 将 来 の 展 望

ここで述べたインライン機能は使用経験をもとに進化しつつあり，現在も新機能が必要となれば変更を加えることができる．将来，あるいは別のシステムとして実現すべき，多くの機能向上案がある．つぎに，そのいくつかをあげる．

1) 引数の安全保護……現在のシステムは引数の検査の責任をインライン記述者に任せている．責任のいくつかは，インライン・インタプリタやコンパイラに委ねることができよう．高水準言語で，引数の型の可能な集合を宣言することにし，コンパイラが渡された引数の型の正当性検査を行えばよい．さらに，すべての引数の型がインライン記述者によって考慮されているかどうかを，インライン・プロセッサが検査できる．

2) レジスタのロード機能……現在，インライン記述者は，大きさや語の境界との割り付け方法にかかわらず，引数をレジスタにロードする命令語まで書かなければならない．必要な情報がすべて判明しているので，これが可能となっているが，もし引数をレジスタにロードする任意の命令語の列を生成する，一般的なロード命令がシステムに用意されれば，エラーや繁雑さも減少するであろう．レジスタから任意の割り付け方のレコード・フィールドへ値を格納するときにも，同様の議論が成り立つ．

3) 汎用出力型……現機能では，インライン関数の型を宣言する必要があり，したがって固定されている．インラインの引数が汎用化されているので，関数の型も汎用化する方が望ましい．これにより，たとえば引数の型によって結果の型を決める MAX や MIN のような汎用の関数を定義することが可能となる．関数の型は，インライン宣言で関連づける一群の型の集合の一つとしてもよいし，また前もって型の集合を指定せずに，渡された引数の型に応じて，動的に選択するようにしてもよい．

4) 汎用レジスタ……インライン記述者は，退避や復元を必要としない一群の自由レジスタを使用できる．これは，このシステムを使う機種が多くのレジスタを備えていることにより，可能となったものである．この方法は少しのレジスタしか持たない機種では適用できない．解決法としては，汎用レジスタの概念を導入して，どのレジスタを使うかをコンパイラが決めるようにすることである．これは，必要に応じて特定のレジスタを用いる方法と併用できよう．このような複合的な方法を用いた場合には，コンパイラがインライン記述者の意図を考慮しながらレジスタの割り当てを行う必要が出てくる．

11. おわりに

ここに述べたインライン・システムの経験を通して，高水準言語を下位方向に拡張する機能が可能で，かつ実用的であり，さらに効果的かつ効率よくコンパイラ生成コードに統合できることが判明した．この機能の本質をまとめると，次のようになる．

1) インライン・コードを許す場所の選択
2) 渡された引数の情報を得る問い合せ関数
3) 選択的なコード生成を行う制御構造
4) レジスタの割り当て方法
5) コンパイラに情報を返す指示
6) エラー診断能力

高水準言語のすべて，あるいは大多数の使用者にとって，このような機能が必要であるとは考えられない．しかし，たとえ少なくとも，必要な人に対してコンパイラに多大な負担を与えず，高水準言語の利点を損わず，モジュール化した方法で機械語やオペレーティング・システムをアクセスする方法を提供できることが示された．

PLUS のインライン・システムが持つ一般性や隠蔽機能は，より頻繁にアクセスする，Ada のような他の言語に対しても，言語が直接用意しない基本的な演算をよく適応すると考えられる．

（ソフトウェア・プロダクト統括部　ソフトウェア企画部　真田　正二 訳）

執筆者紹介 Thomas N. Turba

1972年に Wisconsin 大学より Computer Science で Master を取得．Sperry 社に入り，General Syntax Analyser (GSA), Spelling Checker, MACRO プロセッサの開発を担当．GSA, SPELL, MACRO, Pascal, Ada のプロジェクトに参画．現在，同社の言語システム・グループのテクニカル・コンサルタント．

訳者注 PLUS 言語は，Sperry 社のコンピュータ・システム共通のシステム記述用言語として位置づけられる高水準言語であり，そのコンパイラは OS 1100, OS/3, VS/9 のもとで稼動している．OS 1100 のもとでは，UCS コンパイラ群，UDS データベース・システム，OS 1100, IPF 1100, ADVISE 1100, その他各種ユーティリティの主開発言語として用いられている．本稿は，シリーズ 1100 の OS 1100 上で動く PLUS コンパイラで実現されたインライン機能をもとに記述されている．

報告 ソフトウェア開発環境 "CASE"

The Environment for Software Development "CASE"

斎 藤 哲 郎

要 約 CASE (Computer Aided Software Engineering)は，ソフトウェアのライフ・サイクルを全般にわたって使用できるソフトウェア開発保守のための抱括的な環境である．

本稿では，このCASE に含まれるツールに焦点を当て，その概要と効果および日本語処理の一例を述べる．

これらのツールは Sperry 社で開発され，同社ならびに日本ユニバック(株)内で EXEC を含めた基本ソフトウェアの開発・保守に使用され，生産性および品質の向上に寄与している．

Abstract CASE (Computer Aided Software Engineering) is the comprehensive environment for software development and maintenance, applicable to the entire software life cycle.

This paper descrives the tools included in CASE, results of the usage, and an example of the Japanese language processing of CASE tool.

These tools were developed in Sperry Computer Systems, and are being used in Sperry and Nippon UNIVAC for deveopping OS 1100 software including EXEC. They contribute to the improvement of productivity and quality.

1. はじめに

今日のハードウェア製造における生産性および品質の急速な向上には，目を見はらざるを得ない．これには，その設計・生産・検証という工程で使用されているさまざまなツールが大きく寄与していると考えられる．

翻って，われわれの携っているソフトウェア開発・保守に目を向けると，このようなツールが少くなく，あっても一貫性がなく，汎用的ではない不完全なものが多いことに気づく．ソフトウェアを作っているわれわれが，自分たちの仕事を助けるためにソフトウェアを有効に活用していなかったとも言えよう．

一方，コンピュータ・テクノロジの進歩と相俟って，問題解決のために莫大な量のソフトウェアが必要になってきた．われわれは小さなモニタや単純なデバイス・ハンドラの域から脱して，巨大なデータベース・マネジメント・システムの域に達してきたのである．そして，このような大規模ソフトウェアの開発は，少人数のプログラマでできるわけではなく，多人数のプログラマや複数のプログラミング・チームによって行われ，それらの間のコミュニケーションといった問題が起こってくる．このためソフトウェアの開発工程を標準化し，その工程を次のように定義した．

1) 要求定義……各種の要求（新製品開発，既存製品に対する改良など）を分析，とりまとめて定義する．
2) 開発検討……製品の開発または改良を実施するか否かを検討する．
3) 定義……開発されるべきソフトウェアの機能，性能などを定義する．
4) 設計……上記の定義をもとに，それを実現する方法（アルゴリズム）を作成し，設計仕様書にまとめる．
5) コーディング……設計仕様書にもとづいて，モジュールをプログラミング言語で

記述する.

6) 単体テスト……各々のモジュールが，その設計仕様書を満たすか否かテストする.

7) インテグレーション……各々のモジュールをまとまった機能を有するソフトウェア・コンポーネントに統合する.

8) ファンクション・テスト……ソフトウェア・コンポーネントが，定義された機能・性能を有するか否かを検査する.

9) システム・テスト……ユーザに提供されるソフトウェア・システムとドキュメントの整合性を検討する.

10) リリース……リリースの可否を検討し，注文に応じてソフトウェアをユーザに提供する.

11) プロダクト・サポート……ユーザの円滑なソフトウェアの導入と使用を支援する.

12) 設置……ソフトウェアをユーザが使用可能な状態にする.

13) 保守……ソフトウェアに検出された誤りを修正する.

これは開発工程（ソフトウェアのライフ・サイクルといえる）を13の工程に分けている．各々の工程ごとに期待される入力，必要な出力，処理過程，計画立案，検証機構が定義されている.

このようなソフトウェアのライフ・サイクルを支援するツールには一貫性があり，相互に関連性を持つものが必要とされる．また，デザイン・ドキュメントや他のドキュメントを生成するだけのものではなく，検証機構を持ち適切なテスト環境を作り出し，問題や修正の追跡，レポートを行う必要がある．これらの要求を満たすため考えられたツールを抱括した環境を CASE (Computer Aided Software Engineering) と呼ぶ.

なお，CASE は OS 1100 ソフトウェアの開発・保守を対象としている.

2. CASE ツール

CASE は，ソフトウェアのライフ・サイクルの各ステップを支援するツールの集合体である（これらには，すでに開発されているもの，機能を拡張中のものが含まれている）.

ソフトウェアのライフ・サイクルと，それを支援する CASE ツールとの対応を示したのが図1である.

つぎに，主な CASE ツールについて説明する.

2.1 RDP 1100 (Requirement and Development Processor 1100)

RDP 1100 は，主に要求定義・開発検討・定義設計の工程を支援するツールである.

RDP 1100 はソフトウェア開発にかかわる情報を定型的な言語で記述し，それをデータベースに格納する．このデータベースから情報の検索，検証，報告書の作成を行う.

従来，この種のツールとしては Michigan 大学の ISDOS プロジェクトの PSL/PSA が知られている．RDP 1100 は，この PSL/PSA を基礎にしたもので，ソフトウェア開発情報を記述する非手続き言語 RDL と，その処理システム RDP から成っている.

2.1.1 RDL (Requirement and Development Language)

RDL では，開発対象ソフトウェアの論理構造を表現するために，対象 (Object)，関係 (Relation)，属性 (Property) という三つの基本概念を用いる.

対象というのは，開発対象システムの構成要素のことであり，名前と対象型 (Object type) を持つ．システムの表現はシステムを構成するすべての対象について，その対象と他の対象との関係，および対象の属性を記述することによって行われる.

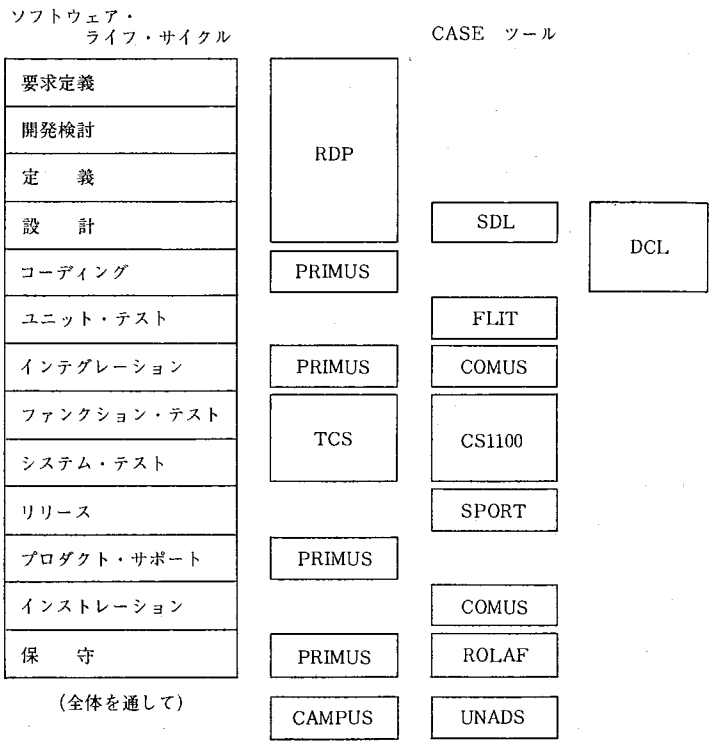


図 1 ソフトウェア・ライフ・サイクルと CASE ツール

Fig. 1 Software life cycle and CASE tools

RDL の記述例を下に示す．ここで RDL のキーワードは大文字で，ユーザが定義する部分は小文字で表現している．また，感嘆符（!）は終止符として用いる．

```
DEFINE DESIGN_FUNCTION fetch-specified-object !
DESIGN_LEVEL IS UNIT !
TITLE !
fetch specified object !
PURPOSE !
given the name of a user-defined object, fetch-
specified-object returns the type code of the object. !
ALGORITHM !
module fetch-specified-object
begin
  call fined-name to verify object's existence and get its type ;
  if error in fine-name ?
    begin
    set 0 type field in packet ;
    set error status ;
    end ;
  else
    begin
    set type field inpacket ;
    staus=0 ;
    end ; !
PROVIDES specified-object-retrieval !
FORMAL_PARAMETERS ARE object-packet (UPDATE), status (OUTPUT) !
INVOKES find-name PASSING para-list !
```

DEFINE 文は，もし fetch-specified-object という名の DESIGN-FUNCTION 型（システムの設計仕様の要素を表す対象型）の対象が，まだデータベース中に存在しなければ，これを生成させる．存在していれば，それに対して適用されることになる．

この例の場合，fetch-specified-object 対象の属性は DESIGN-LEVEL, TITLE, PURPOSE, ALGORITHM の各文によって行われる．属性には2種類あり，DESIGN-LEVEL のような値として定義で許されたものしか指定できないものと，TITLE, PURPOSE, ALGORITHM のように自由形式のもの（テキストと呼ぶ）がある．

ALGORITHM 属性は，この DESIGN-FUNCTION のアルゴリズムを記述するものである．この例ではブロック構造の擬似コード（SDL…後述する）を用いている．

残りの文は，他の対象との関係を記述するものである．FORMAL-PARAMETER 関係は DESIGN-FUNCTION の呼出し系列を記録するために用いられる．INVOKES 関係は，モジュール間の呼出しの関係および実引き数を指定する．PROVIDE 関係は工程間にまたがる関係を定義する．すなわち，定義の工程と設計の工程の対象とを関係づけるものである．

ここでは例としてあげていないが，階層関係を表す PARTS 関係もある．この関係を使用することで，より下位へ，より細部へと設計の詳細化を行ったことを反映できる．

関係の記述では，その補完関係が自動的に定義される．INVOKES **B** PASSING **C** という関係を **A** という対象に記述したならば，**B** 対象には INVOKED BY **A** PASSING **C** という関係が，**C** 対象には PASSED BY **A** INVOKING **B** という関係が自動的に付加される．これにより，さまざまな見方に対する関係の完全性が保たれる．

また，この RDL による対象の記述では全体を完全な形で一度に記述する必要はなく，情報が発生した時点で段階的に構築していくことが可能である．

2.1.2 RDP (Requirement and Development Processor)

RDP は，RDL で記述された開発情報を蓄積するデータベースの作成・保守・検索ならびに，これらの情報にもとづく各種の解析および文書化を行うシステムである．このシステムを用い，設計仕様の修正が他に及ぼす影響の分析，各種の規約に関する適合性のチェ

```
                    STRUCTURE  REPORT  FOR
                  RELATION SUBFUNCTIONS_ARE
                   MAXIMUM DEPTH = 15 LEVELS
                        1983 May 13

01  TCS_KEYWORD_PROCESSING
    02  REG_KEY_TEST
        03  CHECK_KEY
        03  REGISTER_KEY_CTS
    02  KEY_SEARCH_XQT
        03  CREATE_KEY_TEST_DB
        03  STATEMENT_ANALYZE
        03  KEY_SEARCH_SELECT
        03  GEN_TCS_RUN
    02  CROSS_REFER
        03  CROSS_REF_CTS
        03  LIST_CROSS_REF
    02  MODIFY_KEY
        03  MODIFY_KEY_CTS
        03  CHECK_MOD_KEY

END REPORT
```

図 2 RDP ストラクチャ・リポート

Fig. 2 The structure report by RDP

```
Number     |  UPDATE DEPARTMENT EXPENDITURES          |  Update LVL |  1-1
           |  1982 Nov 09                             |             |  Page
```

```
                                        1.  UPDATE DEPARTMENT EXPENDITURES


1.1.  GENERAL

RDL name is EXPENDITURES-UPDATE.

DESIGN_LEVEL IS "UNIT".

IMPLEMENTATION_LANGUAGE IS COBOL.

1.1.1.  PURPOSE

To update on a monthly basis the department expenditures for salaries,
vacation pay,  and sick  leave pay. These  expenditure totals are then
used with the budget to calculate balances remaining to be spent

1.1.2.  SCOPE

1.1.3.  ALGORITHM

        FLOW        'EXPUPD'
        call        'call GETEXP to get this month expenditures'
        call        'call EXPTOT to get total expenditures'
        process        'add   this   month   expenditures   to   total
expenditures'
        process     'output expenditure totals'
        endflow

1.1.4.  DESIGN STRUCTURE

PARTS ARE PRODUCT-UPDATE, PRODUCT-MODIFY.

PART OF DEPARTMENT-RECORDS-SYSTEM.

1.2.  INTERFACES

1.2.1.  CALL INTERFACES

INVOKES GETEXP;
INVOKES EXPTOT;
INVOKES EXPTOT1.
```

図 3　RDP デザイン・ドキュメント

Fig. 3　The design document generated by RDP

ック，文書の自動作成，データの変換（コンパイラの入力作成など）を行うことができる．

ここでは，紙面の都合上 RDP の機能のうち，報告書作成機能を用いて生成したものを例としてあげておく．

- ストラクチャ・レポート（図 2）
- 設計仕様書（図 3）

たとえば，仕様書を作成するには，まず RDP によって文書作成プロセッサである UNADS (UNivac Automated Document System)[注1] の入力になるようなデータを，データベース中の情報から出力する．つぎにそのデータを UNADS に入力し，実行する．

2.2 SDL (Structured Design Language)

システム・プログラムの設計のために開発された言語 SDL は，PLUS (Programming Language for Univac System)[注2] をモデルに設計されている．しかし PLUS より自由度の高い表現ができる．この SDL を入力とし，字下げがなされたリスト（図 4），NS チャート（図 5）を出力するプロセッサが提供されている．

2.3 TCS (Test Controller System)

TCS は大量のテスト・プログラムを管理し，テスト作業（テスト・プログラムの実行，結果判定，報告書作成）を支援するツールである．

```
MODULE;
PROCEDURE sample_flow_chart
          BEGIN
            WHILE chain is not empty DO
              BEGIN
                IF scanned item < > '@EOF'
                  THEN
                    BEGIN
                      IF scanned item = 'PROCESS'
                        THEN
                          BEGIN
                            compute dimensions;
                            draw rectangular box;
                          END;
                        ELSE
                          BEGIN
                            CALL compound statement;
                            record computed dimensions;
                          END;
                    END;
                CASEENTRY line type
                  CASE dotted:
                    output dotted vector command;

                  CASE fragmented:
                    output fragmented vector command;
                  CASE unbroken (solid):
                    output unbroken vector command;
                ENDCASE;
              END;
            RETURN;
          END;
TERM;
```

図 4 SDL リスティング

Fig. 4 SDL output listing

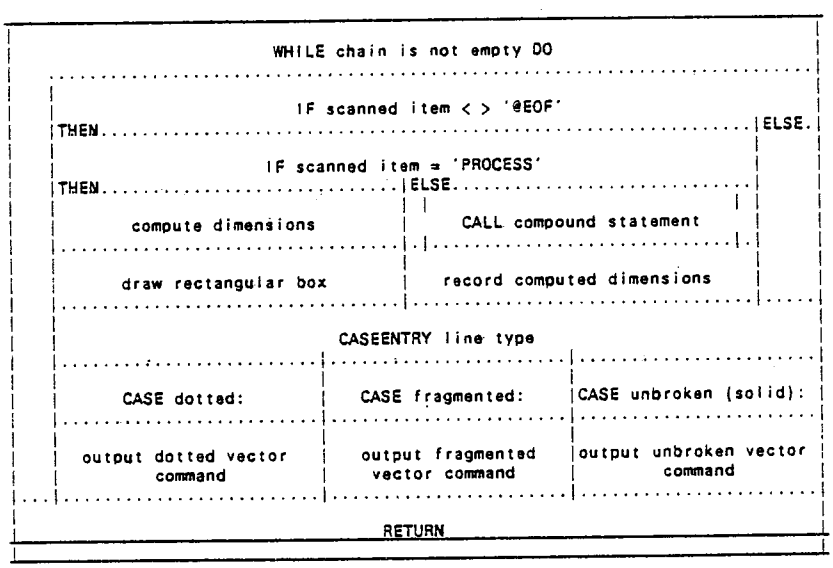


図 5 NS チャート

Fig. 5 Nassi-Shneiderman diagram

TCS は，コントローラと機能ごとに分割されたユーティリティから構成されている．コントローラはすべてのユーザ・インタフェースをつかさどり，使用者からの指示に従ってテスト・プログラムを選択・実行し，適時ユーティリティ・プログラムを実行する機能を持つ．また，対話型の使用を前提に考えており，教育機能を持つことでドキュメントなしでも操作できるようになっている．

TCS の機能は以下のとおりである.

1) テストに必要なファイルの選択，ファイルとテスト・プログラムとの連結の準備.

2) テスト・プログラムの選択，テスト・ランストリームの生成および実行. テスト・プログラムを，デマンド・モードで直接実行したり，バッチ・モードでバックグラウンド・ジョブとして実行することも可能.

3) あらかじめ実行し保存してあるテストの出力リストと，今回実行されたテストのリスト.

4) テストに通らなかったもののみ，その実行リストを出力する.

5) テスト結果サマリの出力（図6）.

```
HARDCOPY STATUS OUTPUT FOR USER-NUMBER 01 - RUNNING TEST CONTROLLER SYSTEM (TCS) ROUTINES        DATE 051783

TEST CONTROLLER SYSTEM STATUS PRINTED AT 10:03:40 ON  5-16-83
*** ON THE FLY STATUS -- BATCH MODE ***
*** USER NUMBER 01 - RUNNING TEST CONTROLLER SYSTEM (TCS: 1R1) ROUTINES ***
 44 TEST(S) PASSED.
  1 TEST(S) BYPASSED DUE TO ENVIRONMENT.
-----
 45 TOTAL TESTS PASSED.
  3 TEST(S) FAILED.
  1 TEST(S) TERMINATED ABNORMALLY.
-----
  4 TOTAL TESTS FAILED.
  1 TEST(S) STILL ACTIVE.
*****
 50 TOTAL TESTS RUN.

*** TESTS THAT HAVE FAILED ***
AA010         AA020         AA022         AA026
AA031        HAS BEEN ACTIVE FOR     1 MIN.
*** COMPREHENSIVE BATCH STATUS OF ALL TESTS RUN ***
*** USER NUMBER 01 - RUNNING TEST CONTROLLER SYSTEM (TCS: 1R1) ROUTINES ***
### FOLLOWING TEST(S) WERE RUN ON AN 1100/80A UNDER EXEC LEVEL 38R2 ###
  AA001        PASSED.                                9:45;52 ON  5-16-83
  AA002        PASSED.                                9:46:03 ON  5-16-83
  AA003        PASSED.                                9:46:17 ON  5-16-83
  AA004        HAS NEWLY GENERATED BASE.              9:57:44 ON  5-16-83
  AA005        PASSED.                                9:46:52 ON  5-16-83
  AA006        PASSED.                                9:47:09 ON  5-16-83
  AA007        PASSED.                                9:47:27 ON  5-16-83
  AA008        PASSED.                                9:47:48 ON  5-16-83
  AA009        PASSED.                                9:48:00 ON  5-16-83
 >AA010        VARIED IN COMPILATION            <     9:48:12 ON  5-16-83 ***  BASE MASM 3R1    ; TEST MASM 3R1
  AA011        PASSED.                                9:48:32 ON  5-16-83
  AA012        PASSED.                                9:48:49 ON  5-16-83
  AA013        PASSED.                                9:49:06 ON  5-16-83
  AA014        PASSED.                                9:49:16 ON  5-16-83
  AA015        PASSED.                                9:49:24 ON  5-16-83
  AA016        PASSED.                                9:49:37 ON  5-16-83
  AA017        PASSED.                                9:49:57 ON  5-16-83
  AA018        PASSED.                                9:50:12 ON  5-16-83
  AA019        PASSED.                                9:50:25 ON  5-16-83
 >AA020        VARIED IN COMPILATION-EXECUTION<       9:50:40 ON  5-16-83 ***  BASE MASM 3R1    ; TEST MASM 3R1
  AA021        PASSED.                                9:50:56 ON  5-16-83
 >AA022        VARIED IN             -EXECUTION<      9:51:11 ON  5-16-83
  AA023        PASSED.                                9:51:23 ON  5-16-83
  AA024        PASSED.                                9:51:31 ON  5-16-83
  AA025        PASSED.                                9:51:38 ON  5-16-83
 >AA026        TERMINATED ABNORMALLY * * * * *<                           DUE TO USER RUNSTREAM ERROR.......
  AA027        PASSED.                                9:52:00 ON  5-16-83
  AA028        PASSED.                                9:52:10 ON  5-16-83
  AA029        PASSED.                                9:52:20 ON  5-16-83
  AA030        PASSED.                                9:52:31 ON  5-16-83
  AA031        IS CURRENTLY ACTIVE.                  10:02:19 ON  5-16-83
  AA032        PASSED.                                9:52:57 ON  5-16-83
  AA033        PASSED.                                9:53:08 ON  5-16-83
  AA034        PASSED.                                9:53:18 ON  5-16-83
  AA035        TEST EXECUTION BYPASSED * * * *                            DUE TO ENVIRONMENT CHECK FAILED.
           ENVIRO COMMAND: SYSTYP=1100/60
```

図 6 TCS テスト結果サマリ

Fig. 6 TCS test result summary listing

6) 3）の比較に必要なリストの作成・保存.

7) コミュニケーションのテスト……CS 1100[注3] と連結をとり，端末装置（TSS, リアルタイム）のシミュレーションを行い，コミュニケーション・ソフトウェアの負荷テストや機能テストを行う. 機能テストにおいては，端末装置での利用者の操作を簡単な言語で記述し，それに従ってシミュレーションを行う機能を持つ.

2.4 PRIMUS (Problem Reporting Integration and Maintenance for UNIVAC System)

PRIMUS は，プログラムのコーディング保守に対する支援を行い，ソフトウェアのインテグレーション，およびプロダクト・サポートの工程を管理し，その情報を与えるものである．

この目的を達成するために，これらにかかわる情報を格納する一つのデータベースを提供する．

このデータベースには，下記の情報が格納される．

1) SUR (Software User Report, ユーザからのトラブル報告), TQ (Technical Question, ユーザからの技術上の問い合せ)……その内容，優先度，回答，責任者などの情報．
2) PLE (Problem List Entry)……すでに解決されているソフトウェアのトラブルの情報（外部・内部現象，修正方法など）が格納される．トラブルの解析に用いる．
3) CHG (Changes)……ソフトウェアの修正，あるいは改造を行ったコードおよび誤りの内容，修正の内容，適用条件などの情報．
4) CIF (Customer Information File)……ソフトウェアをリリースしたユーザの情報．
5) 上の相互関係

データベースは，端末装置とオンラインで結ばれ情報の発生時点で入力され，また必要なときにその場で検索することができる．

これらの情報から管理者に対して，さまざまな報告書を生成することも可能である．

2.5 COMUS (Computerized On-site Maintenance for User's System)

ソフトウェアの設置や保守（修正コードの取り込みなど）を自動化し，簡素化するために使用される．

主な特徴は以下のとおりである．

1) 統一されたソフトウェアの生成，設置のしくみを提供する．
2) 対話型のユーザ・インタフェースを持ち，作業の省力化，簡素化をはかる．
3) 緊急保守，完全化保守に簡便な方法を提供する．

2.6 CAMPUS (Computer Aided Management and Planning for Univac System)

ソフトウェア開発に関するさまざまな管理情報をデータベースに格納，およびそのデータベースから各種の管理資料を出力するツールである．データベースは RDP 1100 のものを使用し，使用者とは全画面のインタフェースを提供している．管理情報として格納できる主なものは以下のとおりである．

- プロダクトの目標と概要
- プロダクトのプラン，マイルストーン
- 資　源
- 長期管理計画
- リリース計画

2.7 その他のツール

1) DCL (Declaration Control Language)……データ構造を記述する言語であり，出力としてデータ・チャート，アセンブラコード，PLUS コードが得られる．
2) FLIT……実行用プログラムを主記憶装置に入れた状態で，プログラムのテストやデバッグを対話型で行う．実行用プログラムに対してスナップ・ショットなどの診断命令を挿入せずに，プログラムの追跡やデバッグができるほか，直接記憶場所の内容を変更してプログラムの実行が可能である．

3) SPORT……ソフトウェアの出荷作業を支援するツール.

4) ROLAF (Remote On-Line Analyze Facility)……トラブルの早期解決のために遠隔地からその解析を行うツール.

3. 日本語処理

近年，日本語処理が注目をあび，とくにソフトウェア開発支援ツールに関しては，その要求が非常に強い.

前章で述べた CASE ツールの中で，RDP 1100 に関して日本語処理の試みがなされている．現状では RDL の文法すべてを日本語化するには至っていないが，テキスト部分を日本語で記述し，それをデータベースに格納し，そこから完全な日本語の文書を出力するという処理を行えるようにし，現在試行中である.

この概要を図7に，また出力例を図8に示す.

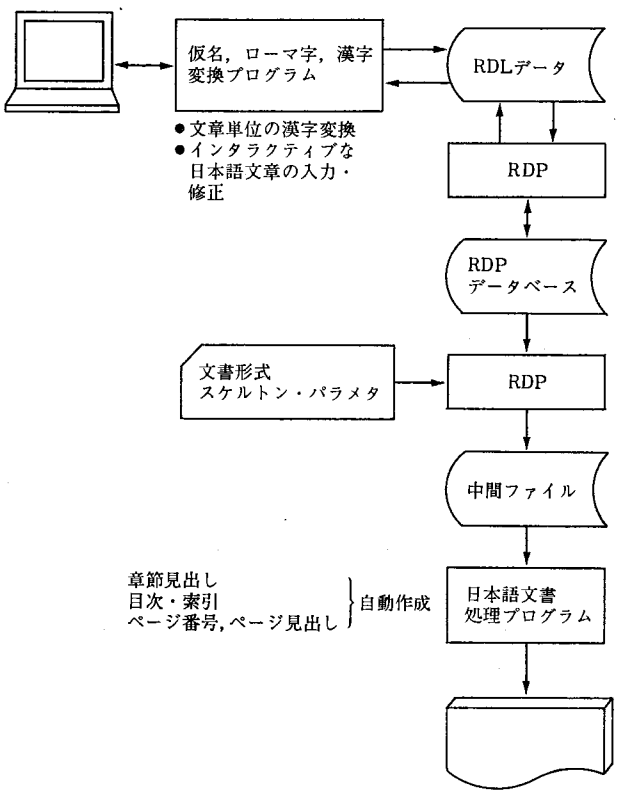


図 7 RDP の日本語処理概要

Fig. 7 Japanese language processing of RDP

4. 効　果

CASE の環境下でのソフトウェア開発で得られる効果をあげてみる.

1) 正確かつタイムリなドキュメンテーション……現在のドキュメンテーションの方法ではその規模があまりに大きく，たとえばシステムの初期開発が終了した時点でもドキュメントは期限に遅れており，その内容も正しくないということがたびたびある.

たとえドキュメントができたにせよ，各人が特定の機能についての情報やテーブルを見たいとしても，それを捜し出すのは容易なことではない.

NIPPON UNIVAC KAISHA.LTD. COMPONENT PRODUCT SOFTWARE DISCRIPTION

NUMBER: CPSD-001	REV A	TCSのキーワード処理機能ユーティリティ	4-1

4.. インタフェース

4. 1. ユーザ・インタフェース
キーワードとテスト名との関係はCTSサブルーティンREGISTERで登録する。
此の時インプットされたキーワードがAIDSキーワード・データベースに登録されているかチェックし登録されていなければエラーとする。
キーワードとテスト名の関係を変更するにはCTSサブルーティンMODIFYを使用する。此の時も同様にキーワードのチェックがなされる。
TCSの実行に際してはキーワードと論理演算子から成るブーリアン式でテストを選択し実行させることが出来る。

4. 1. 1. テスト登録時のキーワードチェック機能

RDL name is KEY_CHECK_J

概要

此の機能は現在使用されているテスト登録ユーティリティの中に組み込まれる。
又、此の機能はPCEエレメント（JCLファイル、テストパッケージ名）の中に次のような指定があるときのみ働くようになっている。

```
<PCEエレメント>
*テストパッケージ名
:LEVEL
:PARAMETER
  .
  .
:DOCUMENTATION
 CHECK-KEYWORDS AIDSのプロダクト名

:END
```

オペレーション

```
@CTS,I  CASE*TCS.
  .
テスト登録時のオペレーション
  .
  .
---  Special purpose keyword ---
```

図 8 RDP 外部仕様書（日本語）

Fig. 8 The functional specification document (Japanese)

RDP 1100 によるソフトウェアの開発では，より自然な方法でドキュメント作成が可能となる．すなわち，システムの開発と同時に，ドキュメントの開発も行うことができる．さらに，データベースから欲しい時に最新の正確なドキュメントを入手できるようになる．

2) 各工程間のトラッキング……RDP 1100 によりソフトウェアの要求・定義・設計の記述を行えば，必然的にこれらの工程間の関係が記述されていくことになる．このことにより，たとえばあるモジュールに変更があった場合の影響を迅速に発見することが可能である．

3) テスト制御……テストは非常に骨のおれる作業で，充分な管理ができずにテストもれや，リグレッションの問題が生じることが多い．TCS を使用することにより，テストの収納・選択・実行・結果のチェックを自動的に行い，テスト作業の生産性の向上や正確性の向上が図れる．また，管理者に対しては，テスト状況を伝えるためにテスト結果の報告書が出力できる．

4) プロダクト・サービス……ユーザ数の拡大や大規模ソフトウェアの開発にともない，

問題点をタイムリに解決し，その結果をユーザに回答すべく状況を監視し続けていく仕事の量が増大してきた．

SUR や TQ，PLE がデータベースに記録されていることにより，他のユーザから報告された SUR が既知のものであるかどうか，すでに解決されているものであるか否かを迅速に知ることができる．さらに SUR を受け取ったとき，これらを既存の特定の問題と関連づけて問題の発生条件や発生回数を記録することができる．また，これらの情報によりソフトウェアのレベルアップ時のリグレッションの問題も改善されていく．

4) 開発の進捗状況管理……データベース上には，開発各ステップの予定や，マイルストーン，完了日などが記録できる．CAMPUS により開発の進捗状況に応じ，これらを迅速に更新していくことができ，必要なときに簡単に検索ができる．また，管理者には，これらの情報をまとめた報告書が出力できる．

5. おわりに

われわれは，この CASE ツールの使用経験やその研究により，これが

1) 管理面からの開発過程の可視性と制御の向上
2) 生産性の向上
3) 品質の向上

に有効であることを知った．

注記 1) 一般書籍から技術論文，手紙など各種定型文書の作成に広く利用できる汎用性に富んだ英文文書作成プロセッサである．
2) Sperry 社で設計された社内のシステム・プログラム開発用の高水準プログラミング言語．
3) 実際の端末装置を使わず，シミュレーションによって回線系のテストを行うソフトウェア．

参考文献 [1] D. Teickroew, E. A. Hershey, "PSL/PSA : A Computer Aided Technique for Structured Documentation and Analysis of Information Processing Systems," *IEEE Trans. Softw. Eng.*, Vol. 3, No. 1, 1977, pp. 41-48.
[2] H. C. Heacox, "ソフトウェア開発言語 RDL", 技報, 日本ユニバック(株), No. 0, 1981, pp. 20-29.
[3] W. F. Rohde, "Engineering our software solutions-The RDP role", *USE Inc. Fall Conference*, Nov. 1982, pp. 785-807.

執筆者紹介 斎藤哲郎 (Tetsuo Saito)
昭和 29 年生，52 年京都大学工学部情報工学科卒業，同年日本ユニバック(株)入社．ソフトウェア検査，ソフトウェア開発支援ツール関係に従事．現在，ソフトウェア・プロダクト統括部ソフトウェア企画部に所属．

論文 テスタビリティを目的とした VLSI 設計論

Design Methodology for Testability VLSI

C. H. Chen

要 約 本稿は，「テスタビリティを目的とした論理設計」という主題を論理設計の方法論の立場から研究したものである．論理回路のテストのライフ・サイクルについても触れつつ，VLSI のテストの特徴を分析する．

高いテスタビリティを持つ論理 (logic) を得るためには，DC テストのためのスキャン・セット技法(scan/set technique) を越えるような設計上の制約条件 (design constraints, 設計制約) を考えなければならない．

この設計制約には，クロックのずれ (skew) の影響の最小化，回路動作の同期化，ハザード (hazard) やレース (race) の除去が含まれる．

一般化されたトポロジカル・ネットワーク・モデルを用いて，動的なテストが可能であるための必要十分な設計の制約条件を，ブール式の形で表す．また，ある種のレベル・センシティブ・スキャン設計 (LSSD: Level Sensitive Scan Design) 規則は，この式の特殊解となっていることを示す．

Abstract This paper studies the subject of "Design for Testability" from the aspect of logic design methodology. An analysis of VLSI testing characteristics as well as the life cycle of logic testing are presented.

To achieve high logic testability, design constraints which are beyond the scan/set technique for DC test must be considered. These constraints include clock skew effect minimization, synchronized logic operation, and hazard/race avoidance.

By using a generalized topological network model, design constraints which are necessary and sufficient for dynamically testable logic are described in terms of Boolean equations. It is shown that some Level Sensitive Scan Design (LSSD) Rules are particular solutions to the equations.

1. はじめに

本稿はテスタビリティを目的とした VLSI 論理回路の設計原理について述べたものである．テスト可能な論理を実現するための設計上の制約条件はブール式で表現される．

"テスタビリティを目的とした設計" についての正しい認識を得るためには，VLSI のテストの特徴を分析すると共に，VLSI 論理回路のテストのライフ・サイクルが明らかにしておかなければならない．また，最近ではカスタム VLSI 開発のコストおよび設計サイクルを削減するためには，テスタビリティのための論理設計の制約条件を課すのが，最も有効な方法であることが明らかになってきている．

従来，"テスタビリティを目的とした設計" は主にコスト，工学的処理 (engineering mangement), あるいはスキャン・セット技法の変形といった観点から議論されてきた．また，簡単にテストできる論理機能を実現すべく考え出された理論には，さまざまな実際上の限界が存在してきた．それは，ユネイト関数 (unate function) の必要性やファンアウト

制約の除去，あるいは遅延の制約など(たとえば，Reed Müller 表現[1])である．さらに，状態遷移グラフ (state graph) の観点から，テスタビリティを持つ順序回路を設計する研究[2]は理論的興味にとどまっている．

本稿では，“テスタビリティを目的とした設計”という主題を設計の方法論の視点から議論する．得られた結果は最少限の遅延の犠牲だけで，論理機能に対する制限は一切加えずに任意の論理回路に適用できる．本研究には，レベル・センシティブ・スキャン設計への構造的アプローチ[3]，スキャン・セット技法のさまざまな適用，および，さまざまな出版物たとえば Gibson や Schneider の文献[5],[16] で提起されているアド・ホック (ad-hoc) な設計規則が含まれている．

LSSD[3] は，テスタビリティを目的とする設計に対するシステマティックなアプローチであり，従来から非常に首尾よく適用されてきている．LSSD のエッセンスは，設計に対する制約規則の集まりであり，それによりテスト・ベクトルが生成でき，大規模テスト・システムに容易に適用可能となる．この場合，レースの起きる可能性もまた除去されている．テスタビリティを目的とした構造的設計手法として，LSSD はシステムに特定の制約条件を課する形の一つの解である．一方，本稿の力点は動的なテストを可能にする，より一般的な基本的設計規則と設計原理を明らかにすることに置かれている．

2. VSLI テストの概観

テスト仕様，テスタビリティ設計という視点から，論理は以下の三つに分けられる．

1) 実装コンポーネント (off-the-shelf component)
2) アレイ・ロジック (array logic)
3) カスタム LSI

である．

商用の実装コンポーネントは，価格上の理由からテスト回路を組み込んではいない．しかし，論理機能のテストは可能である．それは，①チップの機能が明確 (well defined)，②現場における保守には関与しない，③大量に使用されるためテスト・ベクトル生成に発見的手法が適用可能，などによる．

メモリ[6]，マイクロ・プロセッサ[7]，あるいは，CRT-コントローラやコミュニケーション・インタフェースなどのペリフェラル・ロジックの機能テストの手法も多くの人々により研究されてきている．

アレイ・ロジックについては，PLA (Programmable Logic Array) のテストが広く研究されており，多くの有用な結果が得られており[8]，SLA (Storage Logic Array) のテストについても研究がさらに進められようとしている[9]．本稿では，主として，ライフ・サイクル・コストや設計期間が，クリティカルなコンピュータ・システムの設計に使われるカスタム VLSI 論理のテストについて述べる．

2.1 テストのライフ・サイクル

VLSI のテストのライフ・サイクルは，以下の四つのフェーズから成っている．

1) 論理設計の検証
2) 電気的設計の検証
3) 製品テスト
4) 現場における保守テスト

論理設計の検証は設計の正当性のテストであり，実際にハードウェアを作り上げる前に

行われる．テストは通常，論理シミュレータを用いてソフトウェア的に行われる．SSI/MSI を組み合せて VLSI に対応するブレッド・ボードを作って行うテストは，コストの点からもテスト期間の点からも無駄が多い．第2フェーズは，クロス・トーク，浮遊容量，内部結線などのための実際のハードウェア・デバッグ，または電気的レベルでのデバイス・シミュレーションである．これらのフレーズは，設計プロセスの一貫としてあるまとまった時期に行われるものであるが，製品テストや現場における保守テストは，以後繰り返し行われていくことになる．

一般に製品テストはチップ・レベルから始まり，ボード，モジュール，システムのテストと進んでいく．一方，現場における診断テストは故障個所を見つけるためのものであり，逆の順序で行われる．故障を見い出すためのコストは，設計段階，チップの作成，ボードの作成段階と進むに従い，指数関数的に増大する．したがって，テストのライフ・サイクルの早い段階で故障を発見することが至上命令となる．テスタビリティを目的とした設計は，サイクルの最も早い場面で行われるため，テスタビリティ・メジャー (testability measure)[10] やコンパクト・テスト法[11] などに比べて，コストの面で，最も効率のよい方法だと言える．

2.2 VLSI テストの特徴

VLSI のテストは，LSI 以前のテストと何ら異なるところはないし，むずかしいというわけでもないという主張が数多く存在する．LSI 以前のボードのテストで起きた問題が，LSI のチップ・レベルのテストでも起きるようになり，モジュールのテストが VLSI のボードのテストになったというわけである．

この違いを明らかにするために，VLSI のテストに必要な項目を分析し，以下に VLSI 以前のテストとの差異を示しておく．

1) **ゲートは前もってテストされていない**

LSI 以前のボード・レベルのテストにおいては，各 MSI/SSI コンポーネントの特性は十分明らかにされており，その機能もアセンブル前に十分テストされている．したがって，ボード・レベルの故障検査はそう高度ものでなくてもよかった．それに反し VLSI のマクロセルはテストされておらず，VLSI のチップ・レベルのテストでは，チップ内の全ゲートを対象としたほとんど 100 パーセントの故障検出率が必要となってくる．ゲートは複雑に組み合わされており，機能テストを余すところなく行うのは，もはや実際的でない．

2) **アクセス・ポイントが限定されている**

VLSI にあっては，内部ゲートはプローブしたり，イン・サーキット・テストをすることはもはや不可能であり，こうしたゲートの外部ピン当たりの数も非常に多く，高い故障検出率を得ることをより困難にしている．

3) **膨大なゲート数**

VLSI のゲート単価は低く，今や冗長設計 (redundancy design) がコスト効果を発揮し始めている．多くの既存の CAD は，ゲートの組み合せが高度になるにともない時代遅れになるであろう．とくに，冗長な回路のテスト・リストの生成，大規模デジタル・システムの論理シミュレーションが今後の課題である．

3. 設計方法論

このセクションでは，テスタビリティの要件を分析しテスト可能な論理のための設計原

理について述べる．

多くの文献が，VLSI テストの主たる困難として以下のものをあげている．

1) 複雑な順序回路の故障検査

2) グリッチ (glitch) やレース (race) の動的なテスト

順序回路のテストの問題を状態遷移グラフ (state graph) の観点から研究したものも数多く存在するが[2]，それらは学問としての研究である．動的な異常機能 (malfunction) を持つ順序回路は，さらにより困難を増す．一方，組み合せ論理回路の静的なテスト (static test) は比較的容易である．

以上のことから，VLSI テストの問題を解決するステップは次のようになる．

ステップ(1) 順序回路のテストという複雑な問題を，より簡単な組み合せ回路のテストとメモリのテストに分解する．

ステップ(2) 動的なテストの問題を静的遅延テスト (static delay test) の問題に帰着させる．

第1のステップは，アクセスできてテストできるメモリを持つ順序回路の Huffman モデルを用いて行われる．最近では，大規模コンピュータの設計には複雑な順序回路のテストを省くために，スキャン・セット技法を用いて成功している．この技法はすべてのメモリ素子（フリップ・フロップ）をシフト・レジスタに結合させ，すべてのフリップ・フロップが独立してテストでき，組み込まれている組み合せ回路のテストのアクセス・ポイントとしても使うことができるようにしたものである．

第2のステップは同期回路の特徴である．イベントは基本クロックで制御され，素子の遅延は基本クロックにもとづいて定められる．レースが現れないように設計された同期回路においては，組み合せ回路の部分のテストは容易であり，単に最大遅延のみが重要となる．

スキャン・セット技法をくわしく述べる必要はないだろう．以下では，動的なテストを可能にする設計手法について述べる．

3.1 論理回路モデル

テスタビリティ設計原理を図1のネットワーク・モデルにもとづいて述べる．まず，記号の説明は次のとおりである．

S: クロック同期記憶素子

X, Y: データ入力 D_i，クロック入力 C_i 用の組み合せ回路．JK，あるいは RS 同期フリップ・フロップに対する入力 J, K, R, S は入力 D_i に対応する．

Q_i: クロック ϕ_i で同期される出力

$\{P_X\}$: 回路Xへの1次入力信号の集まり

$\{P_Y\}$: 回路Yへの1次入力信号の集まり

$\{\phi_X\}$: Xへの1次入力クロック信号

$\{\phi_Y\}$: Yへの1次入力クロック信号

$\{Q_X\}$: レジスタから出てXにはいる信号の集まり

$\{Q_Y\}$: レジスタから出てYにはいる信号の集まり

論理回路モデルへの入力としては，1次入力信号 $\{P_{X,Y}\}$，クロック入力 $\{\phi_{X,Y}\}$，仮想入力信号 (virtual input) $\{Q_{X,Y}\}$ の三つのタイプから成っている．

任意の X, Y に対して，その出力は特定のクロック ϕ_i について展開すると，

$$X = F + H\bar{\phi}_i + G\phi_i$$

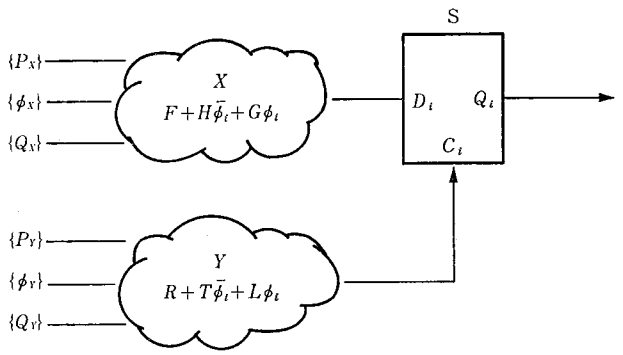


図 1 論理回路モデル

Fig. 1 A logic network model

$$Y=R+T\bar{\phi}_i+L\phi_i$$

となる．ここで，F,H,G,R,T,L は $\phi_i,\bar{\phi}_i$ を含まない論理式である．

任意の論理回路は，図1の形の部分回路に分割することができる．したがって，以下に述べる設計原理が各 ϕ_i に対して適用されることとなる．

3.2 設 計 原 理

高いテスタビリティを実現するためには，スキャン・セット技法を越える設計の制約条件が要求される．この設計の制約条件には，クロックのずれの影響の最小化，回路の同期動作化，ハザード/レースの除去，テスト・リスト作成を考慮することなどが含まれる．明らかに，スキャン・セット論理はシリコン資源を余分に使うことになる．もっとも，このオーバヘッドは，製造のライフ・サイクルのコストの面から見れば問題にならない．ゲート/ピン比率が 25 以上の VLSI については，とくにそうである．一方，レース/ハザードの除去は適当な設計原理を課することにより，ハードウェアをほとんど追加することなく可能である．さらに，ある種のテスト回路は論理機能として使うことも可能である．

1) 回路の同期動作

デジタル・システムのレジスタ・トランスファ・レベル (register transfer level) での記述という観点から見ると，回路を同期形で設計することが実際的でもあり，自然でもある．なぜならば，設計者はデジタル・システムの機能を，あるアーキテクチャのもとでのレジスタ間の並行な情報の流れとして実現しようとするからである．情報の流れの伝達関数は組み合せ回路の機能である．レジスタへのローディングは，クロックの所定の位相で制御される．

RS-フリップ・フロップのような非同期メモリ素子，ディレイ・ライン，シングル・ショット，オシレータなどのテストは問題であり，これらは1次入出力ピンで制御/テストされなければならない．一方，同期回路のテストは各フリップ・フロップのクロック C_i がシステム・クロックで制御されているとすると容易である．つまり，すべての内部クロックが一つ，または複数のクロック ($\{\phi_i\}$) でパルスのタイミングを制御できるように生成されている場合である．このための必要十分条件は，もし ϕ_i=off ならば，C_i=off となることである．

クロック信号を奇数回反転させたものと偶数回反転させたものをゲートした出力 (odd parity reconvergent fan-out) は，ハザードの原因となる．したがって，クロックを入力として持つすべての組み合せ回路は，注意深く設計されていなければならない．こうしたものは，重り合わないクロックの相互に排他的な性質を利用することにより，

除去可能である．

レースのない同期回路の論理設計の制約条件を以下の(a)，(b)に示す．

(a) レジスタへのデータ入力は，そのレジスタのクロックの関数であってはならないものとする．クロックを正-アクティブとすると，各クロック ϕ_i に対して，

$$\bar{F}\cdot L\cdot G\equiv\bar{F}\cdot L\cdot H\equiv 0$$

となる（図1）．

すなわち，D_i と C_i の間のレースは除去されている．

この方程式の一つの明らかな解は，

$$G\equiv H\equiv 0$$

である．

これは必要以上に限定的であるが十分な方程式の解である．この場合，クロック入力は直接，間接を問わずレジスタのデータ入力に影響しない[3]．

(b) レジスタへの1次クロック入力 ϕ_i は，ϕ_i の関数である別の信号と同一のゲートに入力されてはならない．

一つの1次クロック入力 ϕ_i が二つのフリップ・フロップ Q_i, Q_i' を組み合せ回路 $(X, Y), (X', Y')$ を通して制御しているとする．Q_i' の出力は，$\{Q_X\}$ あるいは $\{Q_Y\}$ として X, Y に入っているものとする．

このときは，

$$L\cdot L'\equiv 0$$

となる．

すなわち，Q_i' が Q_i の入力信号を供給していれば，Q_i と Q_i' が同一のクロック位相でスイッチされることはないのである．

一つの明らかな解は

$$L\equiv 0 \text{ または } L'\equiv 0$$

である．

これは必要以上に限定的であるが，十分な方程式の解である[3]．

2) レジスタの設計

スキャン・セットの概念は，1959年に P. D. Eldred[12] によって導入された．最近になって，LSI/VLSI のゲート単価の低下にともない，この手法が広く採用されてきている．

スキャン・セット設計は，最小のハードウェア・オーバヘッドと入出力ピンでレジスタのテストを可能にする現在のところ知られている唯一の方法である．この技法では，組み込まれた組み合せ回路のテストのためのアクセス点としてレジスタを用いている．

LSSD ラッチは，こうした要求に合致した設計法の一つである．Mulder[13] は，動的回路技法 (dynamic circuit technique) を用いた MOS メモリ素子の効果的な設計法について述べている．Yamada[14] は LSSD で用いられている レベル・ラッチとは異なったメモリ素子の設計について述べている．

VLSI の動的テストにおける一つの問題は，チップ上のクロックのトレランス (on-chip clock tolerance) である．VLSI 設計において，チップ上の信号遅延は出力負荷と配線に強く依存しており，クロック・トレランスは，テスト不能な同期動作を起こすためテスト上重要な問題である．エッジ・トリガ・フリップ・フロップは，設計の単純さゆえに，多くの設計者にとって魅力あるものだが，レベル・ラッチと異なりクロックなまり (skew) の効果が累積してしまう．

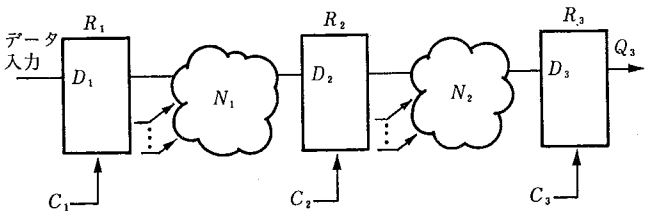


図 2 タイミング解析のための信号の流れ
Fig. 2 A signal path for timing analysis

以下，この点について述べる．

図2に三つのレジスタ R_1, R_2, R_3 と，二つの組み合せ回路 N_1, N_2 とから成る信号の流れの例を示す．C_i は R_i へのクロック入力，t_{Ri}, t_{Ni} は R_i と N_i の伝播遅延を示す．一般性を失うことなく各レジスタのセットアップ時間は0であると仮定し，各クロック C_i のトレーリング・エッジ (trailing edge) を t_{ci} とし，$t_{c_1}=0$ とする．詳細なタイミング解析を行う[15]とレベル・ラッチに対しては，

$$t_{C_3} > \bar{t}_{R_1} + \bar{t}_{N_1} + \bar{t}_{R_2} + \bar{t}_{N_2}$$

となり，トレーリング・エッジ・トリガ・フリップ・フロップに対しては，

$$t_{C_3} > \bar{t}_{R_1} + \bar{t}_{N_1} + \bar{t}_{R_2} + \bar{t}_{N_2} + \Delta t$$

となる．

2段のレジスタを持つデータ・パスの場合には，最大クロック周波数 $1/t_{c_2}$ は双方のレジスタについて同一の値をとることを証明できる．しかし，三つ以上のレジスタから成るデータ・パスの場合は，レベル・ラッチがクロックのずれ (clock skew) の影響を減小させる効果を持つ．レベル・ラッチにおいては，t_{c_1} は t_{c_2} に独立であり，C_2 のパルス幅が遅延の許容度 $t_{R_1}+t_{N_2}$ 以上であることのみが要求される．こうしたことから，レベル・ラッチの動的テストは容易なのである．

エッジ・トリガ・フリップ・フロップの優利な点は，①設計者にとって概念的に使いやすく，またクロックのオーバーラップや，最小遅延などに関与しなくてすむ，②各レジスタに対してただ一つのクロックで済む，などである．

また不利な点は，①レベル・ラッチに比べてレースしやすい，②レジスタの機能上のテストしかできず，ゲート遅延および静的 DC テストができない，などである．

一方，マスタ/スレーブ・レベル・ラッチでは，もう一つの余分なクロックが必要であるが，より完全なレジスタのテストが静的/動的の双方で可能である．以上の理由から，VLSI の設計においては，エッジ素子よりもレベル・ラッチが使われるべきなのである．

4. アド・ホックなテスタビリティ設計規則

さまざまなアド・ホックなテスタビリティ設計規則が検討されてきている[5],[16]．
これらは，以下のようにまとめることができる．

1) 電源，グランド，共通バスなどへの標準入出力ピンの割り当て．
2) 機能，素子のファミリー，V_{cc} などによる回路の分割．
3) 以下のアクセスができるようにすること（初期化，クロック，シングル・ショット，フィード・バック・パス，ファンイン・ファンアウトの大きい節点）．

4) プル・アップする際には V_{cc} に直接接続せず，抵抗を介して行わねばならない．

5) ボード・レベル以上でのワイヤード OR を避ける．これは回路の診断機能を落とすことになる．

アド・ホックな経験則は，主にボード・レベルの設計に適用されてきたため，VLSI のために追加すべきテスト仕様は考えられてこなかった．その中には，広い検査対象，テスト・リスト生成の容易さ，信頼性のある動的な回路動作のテストなどが含まれる．また，アド・ホックな規則の適用の中から示唆されてきたテストの最重要課題，すなわち回路の初期化とフィード・バック・パスの問題はスキャン・セット設計により解決可能となる．

5. おわりに

VLSI テストの特性を分析し，論理テストのライフ・サイクルについて述べた．また，テスタビリティを目的とした VLSI 設計方法論についても述べた．

論理回路のテストは，設計の確認，ハードウェア・デバッグ，製品テスト，現場における保守テストから成っている．テストにかかるコストは，VLSI の製造コストの 30 パーセントを超える．テストのライフ・サイクルの分析結果は，“テスタビリティを目的とした設計” が VLSI のテストの問題を解決するに当たっての最もコスト的に有効な方法であることを示している．この方法は，テスト・コストを減らし，設計サイクルを短くし，テスト・リストの生成，故障診断を容易にし，ほとんど 100 パーセントの故障検出を可能にする．

一般化されたトポロジカル・ネットワーク・モデルを用いて，スキャン・セット技術とは異なるテスタビリティ設計原理を設定することにより，コスト効果を持つ設計方法論が得られた．

とくに，動的にテスト可能な VLSI 論理を実現するために必要な設計の制約条件が論理式の形で表される．この式は設計のテスタビリティを確かめるために，ただちに適用できるものである．ある種の LSSD 規則がこの式の特殊解であることを示した．

最後に，チップの内部クロックのトレランスの影響が少ないことから，VLSI のテストのためにはレベル・ラッチがすぐれていることを示した．

（エンジニアリング・センター　開発一部　上谷　彊輔 訳）

参考文献 [1] S. M. Reddy, “Easily Testable Realization for Logic Functions,” *IEEE TC C*-21 Nov. 1972.

[2] Rene David, et al., “Detecting Transition Sequences,” *FTCS*-9, 1979.

[3] E. B. Eicheberger, T. Williams, “A Logic Design Structure for LSI Testability,” *DA Conference Proc.*, June 1977.

[4] J. H. Stewart, “Future Testing of Large LSI Circuit Cards,” Cherry Hill Test Symposium, 1977.

[5] J. Gibson, “Testability Guidelines,” ATE Seminar and Exhibit, Jan. 1979.

[6] D. S. Suk, S. M. Reddy, “An Algorithm to Detecta Class of Pattern Sensitive Faults in Semiconductor Random Access Memories,” *FTCS*-9 1979.

[7] S. M. Thatte, et al., “User Testing of Microprocessors,” *COMPCON Spring Digest of Papers*, 1979.

[8] D. L. Ostapko, et al., “Fault Analysis and Test Generation for PLA's,” *FTC*-8, 1978.

[9] S. Patil, T. Welch, “An Approach to Using VLSI in Digital Systems,” 5 th Computer Architecture Symposium, 1978.

[10] J. E. Stephenson, et al., “A Testability Measure for Register Transfer Level Digital Circuits,” *FTCS*-6 *Proc.*, June 1976.

[11] K. P. Parker, “Compact Testing : Testing with Compressed Data,” *FTC*-6 *Proc.*, June 1976.

[12] R. D. Eldred, *Datamatic* 1000 *Vacuum Tube Proc.*, 1959.
[13] C. Mulder, et al., "Layout and Test Design of Synchronous LSI Circuits," *ISSCC* 1979.
[14] A. Yamada et al., "Automatic System Level Test Generation and Fault Location for Large Digital Systems," *DA Conference Proc.*, 1978.
[15] C. H. Chen, "VLSI Design for Testability," 1979 *Semiconductor Test Conference.*
[16] D. Schneider, "Design Logic Boards for Automatic Testing," *Electronic International*, July 1974.

執筆者紹介 Chung Ho Chen

台湾大学と Kansas 大学よりそれぞれ B. S. E. E. と M. S. E. E を取得，1977 年に Pennsylvania 大学よりコンピュータ・サイエンスで Ph. D を取得．1965 年に Sperry 社に入り，磁気メモリ，マイクロプロセッサ・システム，論理設計やテスト，および耐故障計算などの研究開発に従事．現在はの CAD/CAM 研究部門の Senior Computer Scientist.

論文 配置改良アルゴリズムにおける配線可能性の測度

Wirability Measure in Placement Improvement Algorithm

A. M. Patel

要 約 本稿では，配置設計の質を計測する配線可能性の測度について示す．全配線長を最小化する従来の設計アプローチでは，心ずしも配線可能な配置が得られない．本稿で提案する反復的な配置設計アルゴリズムは，配線可能性指標と配線長の二つを配置改良の評価基準として用いる．配置改良アルゴリズムのための初期配置は，従来どおりの構成的要素交換アルゴリズムによって得られる．この初期配置を配線可能性改良アルゴリズム (WIA ; Wirability Improvement Algorithm) によって，反復的に改良する．このアルゴリズムは，ゲートアレイ LSI 自動設計システムに組み込まれており，実際のゲートアレイ・チップの配置結果を詳細に述べる．この結果は明らかに本稿のアルゴリズの有効性を示している．

Abstract: This paper develops a wirability measure to gauge the placement quality. It is shown that the conventional approach of minimizing the total wire length does not assure a wirable placement. An iterative placement algorithm which operates on the wirability index and wire length improvement criteria is presented. An initial placement was obtained using the conventional constructive and interchange algorithm. This placement was iteratively improved by the wirability improvement algorithm (WIA). This algorithm has been incorporated in the Automated Gate Array Chip Design System. The results of real gate array chip placement problems are presented in detail. The results clearly demonstrate the effectiveness of the proposed algorithm.

1. はじめに

多くの現実の計画や設計の問題において，複数の適当な位置に複数の物体を割り付けるというような，基本的に組合せ論的作業を必要とする．通常この種の問題は，グラフ理論によって，自然にモデル化できる．この場合，配置する物体を，グラフの頂点と見なし，物体相互の関係をグラフ網にて表現する．たとえば，任意の 2 頂点間の直接リンクは，辺に限るという制約を加えると，問題は線形グラフの頂点と辺によって表現できる．したがって，問題はグラフの辺の長さの合計などの目的関数を最適化するような，最適な（あるいは少なくとも準最適な）頂点の配置を決定する問題に帰着できる．このように抽象化したグラフ問題，すなわち，2 次元割付け問題は，チップ上のゲートの配置，プリント基板の設計，設備計画や都市計画などの位置問題を解くのに用いることができる[1]。LSI チップ上のゲート，モジュール上の LSI チップ，基板上のモジュールなどの位置決め問題は，すべて基本的に同型の問題である．

以下の節で，配置アルゴリズムの概要と，このアルゴリズムのテスト結果を述べる．このアルゴリズムでは初期配置を構成的に生成し，全配線長と配線可能性指標の両方を目的関数として配置状況を反復改良し，最適化する．

2. 初期配置

初期配置は，まだ位置決めされていない頂点の集合の中から一つの頂点を選んで，チッ

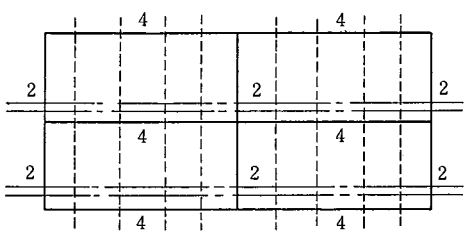
プのすでに形成済みの部分に追加する作業を反復することによって生成する．

構成的配置が得られたならば，二つの頂点の互いの位置を交換する手続きを繰り返して，この配置を改良する，つまり，配線長コスト関数が増加しないように，次々と任意の二つの頂点の位置を互いに交換することによって，与えられた配置を反復改良する．配線長コスト関数は，Manhattan-Steiner 木の構成規則にもとづいて計算する．この方法は，ゲートアレイ・チップのレイアウト問題に関しては，最良と思われる．この構成と交換の手続きは Khokhani らの文献[4] に詳しい．

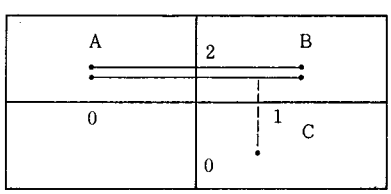
3. 配線可能性指標 (wirability index)

通常，配置を評価するには，全配線長，あるいは全配線必要領域面積を使用する．すなわち，配線長の増加にともない，複雑度は増すという見地から，ほとんどすべての配置アルゴリズムは複雑さを全体として取り扱う尺度として全配線長を用い，これを最小にすることを目指す[2]．配線可能性という概念があり，Heller らの文献[3] にも紹介されているが，これを，明確に配置アルゴリズムに組込む研究は，ほとんどなされていない．わずかに Khokhani らの文献[4] において，頂点の交換手続きの終局過程において，配置を選択するときの測度として配線可能性指標が採用されているにすぎない．ここでは，配置状況の配線可能性を評価してこの測度にもとづいて，配置を反復的に改良する手続きをつくり上げることを試みた．次段では，配線長および配線可能性指標の両方にもとづいた配置の測度を導入する．

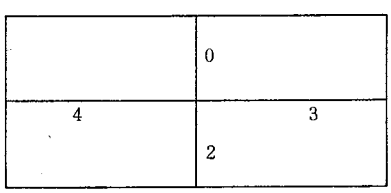
ここで言う配線とは，チップ上の物理的な導線の配置を言う．電気的制約および半導体製造プロセスからの制限によって，一つのチップ上に設けられる配線用トラックの本数に



(a) 4本の垂直トラックと2本の水平トラックを持つ四つのセルからなるチップ



(b) 配置された三つのゲートについての配線設計とそのトラック要求数



(c) 余ったトラック供給数

図 1 4セルチップによる例題： トラック供給数とトラック要求数

Fig. 1 An example of 4 cell chip: supply and demand of tracks

は上限がある．チップ上の配線用トラックのレイアウトは，配置配線設計を行う時点で，わかっている．ゲートアレイ形の IC チップの設計が，まさにその例である．一般に，トラックは2層（あるいは二つのレベル）に分かれている（図1）．

水平層には，水平なトラックしか存在しない．同様に垂直トラックは垂直層にある．垂直トラックと，水平トラックは，ビア（貫通穴）を通して結合できる．

配線によってネット（等電位な配線の集合；訳者）を幾何的に実現するには，導体は一つの層からビアを通って，他の層へ通じていなければならない．設計された配置の配線可能性を評価するために，それぞれのネットの配線は Manhattan-Steiner 木結線法[4]にもとづくものと仮定する．まず各ネットに対して Steiner 木を構成し，次のステップで各セルの辺に対して過剰に要求される配線トラックの本数を計算する．一般に，IC チップの上には，セル格子を重ねて考えることができる．ゲートアレイ IC の設計の場合には，図1(a)のように基本ゲートが一つのセルを構成する．一つのセルの辺を通過する木 Steiner の枝を決定し，その数を，利用可能なトラック供給数から引く．この手続きによって各辺に対する過剰トラック要求数が求められる．このようにして得られた不足トラック数の集合は一つの配置の配線可能性をよく表わしている．そこで，これによって配線可能性指標を定める．これは，初期配置されたセル間の初期のセル間の配線にもとづく悲観的な評価であって，過剰トラック要求数を最小にするように(配線の引き直しをすることによって)さらに改良できることに注意されたい．この配線可能性指標がゼロであること（すなわちトラックの不足がゼロであるということ）は，セルが相互に配線可能な位置に配置されていることを意味している．しかしながら，正確にトラックに配線を割り当てたとき，あるセルの中に矛盾を生ずる可能性はある．そこで，過剰関数 (congestion function) とよばれる配線可能性指標を過剰トラック要求数にもとづいて，次のように設定した．

$$W=\sum_{i=1}^{B}\{S_i+\sum_{j\in k}bS_j\}$$

ここで，

S_i—i 番目のセルの辺についてのトラック不足数（あるいは，過剰トラック要求数）．

$$S_i>0,\quad i=1,2,\cdots,B$$

ここで，Bはトラックの不足しているセルの辺の総数である．

K—各 S_i について，図2に示されるような，隣接する辺の集合 K_i が存在する．この集合は，少なくとも一つの端点を辺 i と共有している辺および，辺 i に並行でかつ最も近い二つの辺を含んでいる．

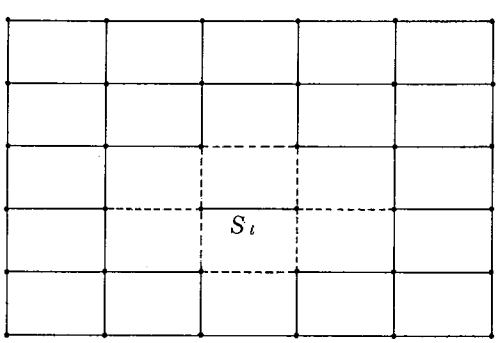


図 2　不足数 S_i を持つ辺に関する隣接辺 S_j を点線で示す．

Fig. 2　The adjacent edges S_j shown as dotted lines for an edge with shortages S_i

J—集合Kに含まれる辺に付した指標.

b—i番目の境界における不足と，隣接辺の不足の間にペナルティをつけるための重み係数.

上の過剰関数は，与えられた配置のセルを配線する際の困難さを示している．この関数では，すべてのトラック不足数を考慮し，配置のトラック不足数の集まり（クラスター）にペナルティづけをする．また，適当に重み係数bを選べば，隣接した辺の不足数の集合の影響を変えることができる．セルiに隣接する辺の影響は，その辺自身の不足数 S_i より効くことは，ありえないから，$b<1$ である． もし $b=0$ なら，隣接した配線面のトラック不足数の荷重効果を，本質的に無視したことになり，Wは全トラック不足数に等しくなる．Wが0に近いところでは，多くの場合，これでも配線可能性の指標として十分であろう．トラック不足数を用いるかわりに，余剰チャネル数を用いても，等価な公式を得ることができる.

表1にテスト用例題の諸元を示す． これらは，どれも160ゲートの TTL ゲートアレイ・チップを用いた実例である.

図3にそれぞれ，例題a，b，cに対する，配線長の関数としての配線可能性指標の振舞いを示す.

このときの配線可能性指標は，$b=0.1$ を用いて計算されている．$b=0.1$ は，任意に選んだ値であるが，トラック不足数の荷重効果をおだやかに効かせている．配線可能性指標は，ランダムな初期配置と構成的な初期配置の双方について，交換改良アルゴリズムをほどこしながら一定の区間ごとに計算してある．この図に表れた傾向は，一般に配線長を短くすれば配線可能性をよくするという，広く信じられている事実を肯定している．しかしながら，同時にほぼ同じ配線長を持つ二つの配置で配線可能性が著しく異なることがある

表 1 160ゲート TTL ゲートアレイによる試験用例題の諸元

Table 1 Description of the test problems for 160 TTL gate array chips

例 題	ゲート数	ゲート当たりの平均ピン数	ネット数	結 線 数
a	155	3.4	136	391
b	144	3.1	130	316
c	106	3.3	87	262
d	158	3.6	140	428

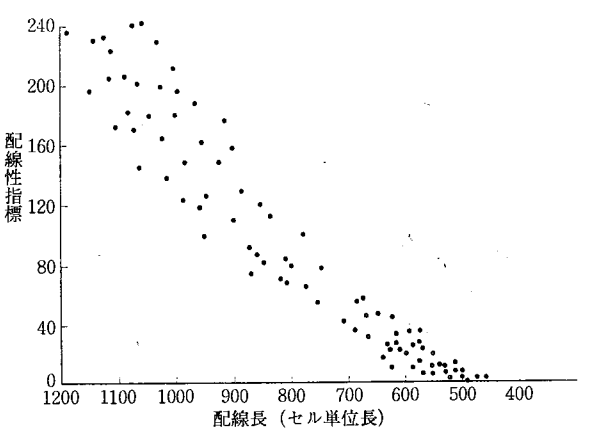


図 3 配線可能性指標 対 配線長

Fig. 3 Wirability index vs. wire length

ということも示唆している．このことは，交換アルゴリズムの終了点付近において，配線可能性指標と配線長という二つの目的関数を最小化するように，アルゴリズムの切り換えを行う必要があることを意味している．次節では，この二つの目的関数を改良するアルゴリズムについて述べる．

4. 配線可能性改良アルゴリズム（WIA）

WIA は，配線可能性指標を最小にするように選び出した頂点の対を交換することを基本としている．WIA は，次のような手順で進められる．

ステップ 1：前述した CON+INT 手法（構成的交換法）によって初期配置を生成する．なるべくよい初期配置が望ましい．

ステップ 2：a．次の不等式を満たす頂点の対のリスト E_1 を作成する．

$$\Delta L(V_i \cdot \mathrm{INT} \cdot V_j) \leq D$$

これは，頂点 V_i と V_j の位置を交換しても，配線長の増加 ΔL が D 以下であることを意味する（E_1 が空のときは，$D=D+1$ として，もう一度このステップをやり直す）．E_1 を，$\Delta L_1 \leq \Delta L_2 \leq \Delta L_3 \leq \cdots$，となるように並べ直す．

b．次の条件を満たす頂点の対から成るリスト E_2 を作成する．

1. $(V_i, V_j) \in E_1$
2. Steiner 木の枝のうちの 1 本が i 番目の境界を通過し，かつ $S_i > 0$ となるネットの集合が存在し，V_i あるいは V_j が少なくとも一つのそのようなネット N_k に属している．

ステップ 3：a．リスト E_2 が空であるか，ストップ 3 の前回の実行で，配線可能性指標が改善されなかったならば，ステップ 4 へいく．

b．リスト E_2 の最初の対をとり出して，もし $\Delta W(V_i \cdot \mathrm{INT} \cdot V_j) \leq 0$，すなわち，交換することによって配線可能性指標が増加しないならば，この対の二つの頂点 V_i と V_j の位置を入れ換える．リスト E_2 が空リストになるまでこれを繰り返す．ステップ 5 へ進める．

ステップ 4：リスト E_1 の最初の対を取り出して，もし

$$\Delta L(V_i \cdot \mathrm{INT} \cdot V_j) \leq 0, \text{ かつ,}$$
$$\Delta W(V_i \cdot \mathrm{INT} \cdot V_j) \leq 0$$

ならば V_i と V_j の位置を交換する．これをリスト E_1 が空リストになるまで繰り返す．

ステップ 5：ステップ 4 の前回の実行で全配線長が改良されるかぎり，ステップ 2 へいく．

簡単に言うと，WIA は，まず配線可能性指標の最小化を試みる．そして，もはや改良されなくなったならば，配線可能性指標が増加しないという条件の下で，全配線長の最小化を試みる．重要なことは初期配置の解は質のよいものでなければならないということであり，さもないと，多大な計算時間を要してしまう．ランダム配置を初期解として用いる場合には，計算時間の面から見てリスト E_1 のエントリ数に上限を与えることが望ましい．図 4 に配線可能性改良アルゴリズム（WIA）を用いた自動配置手続きを示している．

WIA で初期解として使ったのは，CON+INT（構成的交換法）で求めた最終解である．二つのテスト例題 a，b においても，WIA は配線可能性指標ゼロを達成することができた．

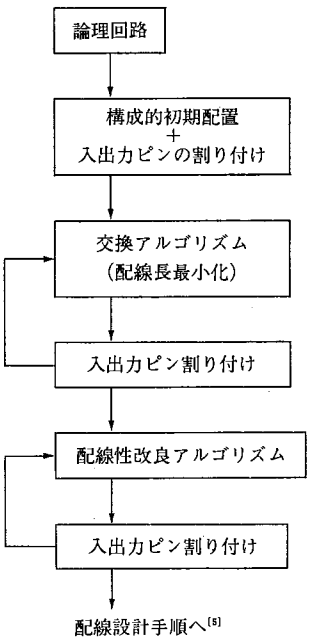


図 4 配線性改良アルゴリズムによる自動配置設計手順

Fig. 4 Automatic placement procedure with wirability improvement algorithm

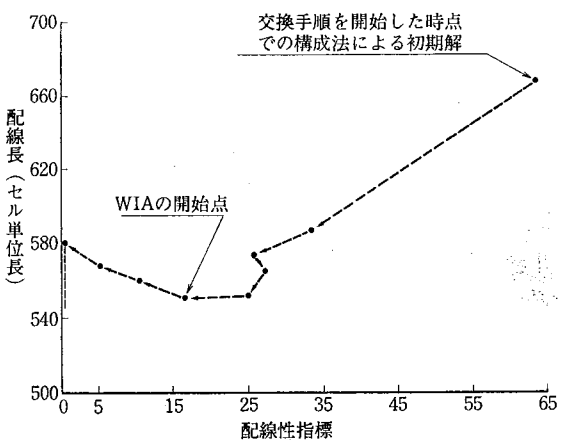


図 5 例題 a についての WIA の結果

Fig. 5 Results of WIA of problem "a"

図5と図6に，例題aとbそれぞれに対する配線可能性指標と，配線長で計った改良の実際の過程を図示する．

図中の点は，それぞれ実行の過程で，あらかじめ定めた間隔で採取したものである．

両方の場合において，WIA が配線可能な配置を得るのに非常に有用であることが証明された．交換手続きの終盤における配線可能性指標の振舞いによって，明らかに配線長だけでは，配置の質を計測するのに十分ではないことが証明される．例題bの場合(図6)には，配線長が579単位長から572単位長に改良されたとき，配線性は18から33へと大きく増加している．

われわれのゲートアレイ設計システムで，150ゲート級のたくさんのチップを処理した．

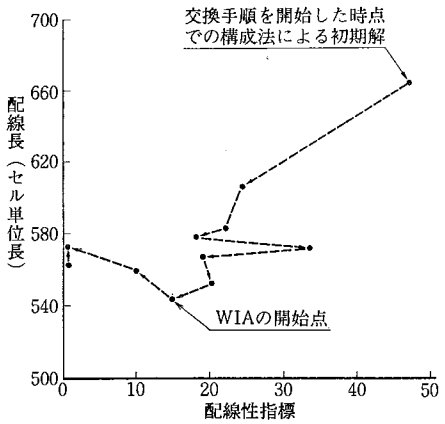


図 6 例題 b についての WIA の結果
Fig. 6 Results of WIA for problem "b"

ゲートアレイは，13×13 の行列構造である．行当たり 4 本の水平トラックと，列当たり 8 本の垂直トラックがある．100 部品の例題に対する配置の平均配線可能性指標は，構成的交換法の終了時において約 28 である．WIA を追加することにより，平均配線可能性指標は 11 に減少し，その減少率は約 62 パーセントであった．このときの配線長の平均増加率は 2 パーセントであった．したがって，WIA をほどこしても，ほどこさなくても配線後の平均配線長は，ほぼ同じであることがわかる．

例題 d に，最もむずかしい例をあげてある．この問題では，配線後に 68 の未結線が残っており，配線は困難であった．配線可能性改良アルゴリズムを適用したところ，配線後には未結線は 39 本に減った．われわれの経験では，手作業で一つの結線を追加するのに，0.5 時間かかることがわかっている．したがって，WIA はこのチップを完成するのに約 15 時間の節約を可能にしたことになる．

WIA の計算時間は，160 ゲートアレイ・チップで UNIVAC シリーズ 1100/80 システムを用いて約 4 分であった．この中には，約 2 分のステップ 1 の CON+INT 法（構成的交換法）の実行が含まれている．各チップについてわずか 2 回の反復で過剰配線トラック要求を減らしている．（すなわち，ステップ 2 は 2 回実行されている．）問題の規模の増大にともなう WIA の計算時間の増加率を定めるのはむずかしい．構成的交換法の計算時間の増加率は，n をグラフの頂点の個数とするときに n^2 に比例する．したがって，WIA の計算時間増加率は，少なくとも n^2 である．WIA の中で最も計算時間がかかるのは，ネットに対して最小距離の Steiner 木を見つけるところである．ネットの中の点が 10 以下なら，n^2 が妥当な計算時間の増加率の目安となろう．

配置アルゴリズムにおいては，計算時間は重大な関心事で，WIA でも，またしかりである．VLSI の問題では，階層的な配置手続きを用いると計算時間の節約に多大の効果がある[6]．2 段階の階層を考えるならば，まず第 1 段階として，ゲートレベルの論理グラフを，複数ゲートのクラスターに分割して，各ゲート群相互の結線が最小になるようにする．この各ゲートのクラスターを上位ノードとして，新しい論理グラフが上位のノードネットで構成される．この上位ノードは本質的には，すべて同じサイズであり，構成的交換法によって配置できる．大局的な配線可能性の観点からいえば，WIA を上位ノードの配置改良にも使用することがすすめられる．VLSI の問題に対しては，ゲートレベルの設計

で WIA を用いることは，必ずしも得策ではない．

5. おわりに

あらかじめ予想したように，配線可能な配置を見い出すには，配線長を最小化するだけでは十分でないことがわかった．本稿で述べた配線可能性改良アルゴリズムは，ゲートアレイ IC のレイアウト・システムに効果的であることが判明した．

最後に，本研究の過程で，L. C. Cote 博士から受けた多大で技術的な貢献に深く謝意を表す．（応用ソフトウェア部 長島 毅 訳）

参考文献

[1] M. Hanan, J. M. Kurtzberg, "Review of the Placement and Quadratic Assignmen Problems," *Siam Review*, Vol. 14, No. 2, April, 1972, pp. 324-342.

[2] M. Hanan, P. Wolff, B. Agule, "A Study of Placement Techniques," *Design Automation & Fault-Tolerant Computing*, Vol. 1, 1977.

[3] W. R. Heller, W. F. Mikhail, W. E. Donath, "Prediction of Wiring Space Requiremen for VLSI," *Journal of Design Automation and Fault-Tolerant Computing*, June, 1978, pp 117-144.

[4] K. Khokhani, A. M. Patel, "The Chip Layout Problem-A Placement Procedure," *ACM IEEE 14th Design Automation Conference Proceedings*, June, 1977.

[5] L. F. Todd, et al, "CGAL-A Multi Technology Gate Array Layout System," *ACM IEEE 19th Design Automation Conference Proceedings*, June, 1982.

[6] A. M. Patel, L. C. Cote, "Partitioning for VLSI Placement," *ACM IEEE 18th Design Automation Conference Proceedings*, June, 1981.

執筆者紹介 Ash M. Patel

1971年に New York 州立大学 Buffalo 分校よりオペレーション・リサーチで Ph. D を取得．1968 年から1978年まで，IBM 社の East Fishkill 研究所に勤務し，LSI/VLSI 自動設計システムの開発に当たる．1979年に Sperry 社に入社，CAD/CAM Research Manager となる．同時に，Pennsylvania 大学の Computer and Information 学科の准教授である．

報告 座標変換差分法による連続体解析

Analyses of Continuum by Means of Finite Difference Method Refering to Coordinate Transformation

藤野 勉

要 約 任意形状の境界を持つ連続体の解析のためには，すでに述べた[1],[2]ように差分法 (FDM) と有限要素法 (FEM) がよく利用されてきた．与えられた問題に対して極座標系，楕円座標系のような直交曲線座標系が利用できる場合は，座標変換による FDM を用いることが便利である．このとき，物理平面上の直交曲線格子系は解析平面上の直交直線格子系に置き換えられ，したがって微分方程式は容易に差分式に変換される．直交曲線格子系が確立していない他の系では，任意節点配置差分法か有限要素法などにより，まずこの系を設定することが必要である．

Abstract For the analyses of continuum with arbitraly shaped boundary, the finite difierence methods (FDM) and the finite element methods were effectively used as we mentioned before. When we can apply orthogonal, curvilinear coordinate system such as poler coordinate, eliptic coordinate for the given problem, it is convenient to use FDM refering to coordinate transformation. In this case the orthogonal curvilinear lattice on physical x-y plane are replaced to the orthogonal straight line lattice on analytical ξ-η plane, therefore differential equations are easily transformed to difference equations by this system. For the analyses of other system where no orthogonal curvilinear lattice system we established, it is necessary to make this system by means of numerical methods such as FDM with arbitraly distributed nodes or FEM.

1. はじめに

任意形状を持つ連続体の解析には，有限要素法と任意節点配置差分法が有効であることはすでに述べた[1],[2]が，その境界の形が直交曲線座標系に適合している場合は，下記に述べる座標変換差分法が都合よく用いられることがある．代表的な直交曲線座標系の例としては，2次元では円，楕円，3次元では筒，球などがあげられる．

座標変換差分法を用いるためには，まず x-y または x-y-z 座標系で与えられた微分方程式を曲線座標系に変換し，ついで曲線座標系内に設定された直交直線格子系を用いて差分化を行えばよい．微分方程式の座標変換には従来，数学的な手続きが用いられていたが，ここではなかば物理的，力学的な考察による方法を採用することとする．たとえば，2階の微分方程式で，前者では2次微係数までの変換を必要とし，かなり複雑な式の変形を必要とする場合があったが，後者では1次微係数の変換のみでよく，微分方程式の変換は Green 積分を通して機械的に行うことができる．

このように既知の直交曲線座標系が存在する場合は，上述の方法を用いることができるが，そうでないときは新たにこれを定めることが必要で，その後に座標変換差分を行うこととなる．与えられる境界の形に応じて直交曲線座標系を設定するためには，一般に有限要素法，または任意節点配置差分法などの数値計算を必要とするので計算手続きは複雑となる．このような困難をともなうけれども座標変換差分法を既存の直交曲線格子系に限らないで，さらに利用の範囲の拡大を図るためには，任意形状境界を持つ領域について直交曲線格子系を定める有効な方法論の確立が必要である．単発の計算の場合は，この方法は

計算手続きが複雑となり不利と思われるが，同じ形の連続体について荷重条件，境界条件など，種々に変化する組み合せ計算については，直交曲線格子系設定の計算を1回行えばよく，その後は簡単な直交直線格子系差分法が用いられるので，効果的と思われる．

2. 直交曲線座標系の微分幾何

ここでは，便宜上連続体の存在する直線座標系 x-y を物理平面，曲線座標系 ξ-η を解析平面と呼ぶこととする．

いま座標 x, y は曲線座標 ξ, η により，

$$x=x(\xi,\eta),\quad y=y(\xi,\eta)$$

と関係づけられ，物理平面における点 (x, y) と解析平面における点 (ξ, η) は1対1に対応しているものとする．

つぎに，物理平面における原点 O(0, 0) と点 P(x, y) を結ぶ位置ベクトルを，

$$\boldsymbol{r}=i_0x(\xi,\eta)+j_0y(\xi,\eta) \tag{2-1}$$

によって定義する．ここに i_0, j_0 は，x, y 方向の単位ベクトルである． 点の微小移動による位置ベクトルの微分は，

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

によって与えられる．ただし，x_ξ, x_η はそれぞれ $\partial x/\partial\xi$, $\partial x/\partial\eta$ を表す． y についても同様である．

式(2-2)より，次の諸量が導入，定義される．

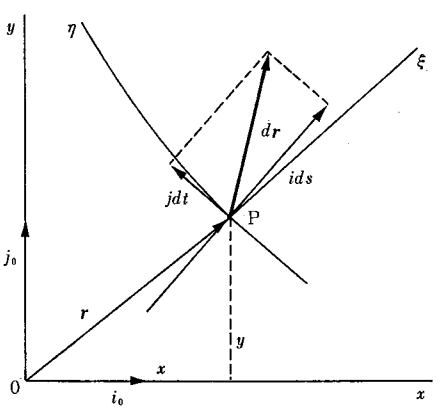


図 1 位置ベクトル

Fig. 1 Position vector

測度係数	$g_1=\sqrt{x_\xi^2+y_\xi^2},\quad g_2=\sqrt{x_\eta^2+y_\eta^2}$	(2-3)
単位ベクトル	$i=(i_0x_\xi+j_0y_\xi)/g_1,\quad j=(i_0x_\eta+j_0y_\eta)/g_2$	(2-4)
微小線素	$ds=g_1d\xi,\quad dt=g_2d\eta$	(2-5)

ここに i, j は，ξ, η 方向単位ベクトル，ds, dt は i, j 方向の微小線素である．とくに，

$$g_1=g_2=g \tag{2-6}$$

が満たされるならば，等測度係数曲線坐標系である．

物理平面では i_0, j_0 は長さ，方向共に一定の定数であるが，i, j は長さは1で一定であるが，その方向は一般に場所によって変化する量で曲線座標の方向を表し，その微係数は

曲線座標の曲がりを代表する尺度である．i, j は直交単位ベクトルであるから，

$$i\cdot i=\left(\frac{x_\xi}{g_1}\right)^2+\left(\frac{y_\xi}{g_1}\right)^2=1,\quad i\cdot j=\frac{x_\xi}{g_1}\frac{x_\eta}{g_2}+\frac{y_\xi}{g_1}\frac{y_\eta}{g_2}=0,\quad j\cdot j=\left(\frac{x_\eta}{g_2}\right)^2+\left(\frac{y_\eta}{g_2}\right)^2=1 \tag{2-7}$$

が満たされ，したがって i_0, j_0 との間に

$$\begin{Bmatrix} i \\ j \end{Bmatrix}=\begin{bmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{bmatrix}\begin{Bmatrix} i_0 \\ j_0 \end{Bmatrix} \tag{2-8}$$

の変換式が成立する．ここに α は i_0 と i，または j_0 と j の挟む角である．

$$\cos\alpha=\frac{x_\xi}{g_1}=\frac{y_\eta}{g_2} \tag{2-9}$$

$$\sin\alpha=\frac{y_\xi}{g_1}=-\frac{x_\eta}{g_2} \tag{2-10}$$

$$\tan\alpha=\frac{y_\xi}{x_\xi}=-\frac{x_\eta}{y_\eta} \tag{2-11}$$

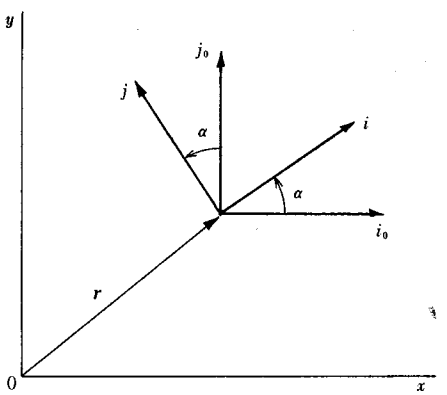


図 2　単位ベクトル
Fig. 2　Unit vector

さらに，式(2-9), (2-11)を微分して次式を得る．

$$\alpha_\xi=(x_\xi y_{\xi\xi}-x_{\xi\xi}y_\xi)/g_1{}^2=(x_\eta y_{\xi\eta}-x_{\xi\eta}y_\eta)/g_2{}^2 \tag{2-12}$$

$$\alpha_\eta=(x_\xi y_{\xi\eta}-x_{\xi\eta}y_\xi)/g_1{}^2=(x_\eta y_{\eta\eta}-x_{\eta\eta}y_\eta)/g_2{}^2 \tag{2-13}$$

つぎに，式(2-9), (2-10)により，式(2-12), (2-13)を次のように変形する．

$$\alpha_\xi=(\cos\alpha\, y_{\xi\xi}-\sin\alpha\, x_{\xi\xi})/g_1=-(\cos\alpha\, x_{\xi\eta}+\sin\alpha\, y_{\xi\eta})/g_2 \tag{2-14}$$

$$\alpha_\eta=(\cos\alpha\, y_{\xi\eta}-\sin\alpha\, x_{\xi\eta})/g_1=-(\cos\alpha\, x_{\eta\eta}+\sin\alpha\, y_{\eta\eta})/g_2 \tag{2-15}$$

同じように，単位ベクトルの微分を求めると，

$$di=i_\xi d\xi+i_\eta d\eta,\quad dj=j_\xi d\xi+j_\eta d\eta \tag{2-16}$$

$$i_\xi=j\alpha_\xi,\quad i_\eta=j\alpha_\eta,\quad j_\xi=-i\alpha_\xi,\quad j_\eta=-i\alpha_\eta \tag{2-17}$$

となり，測度係数の微分を求めると

$$g_{1\xi}=(x_\xi x_{\xi\xi}+y_\xi y_{\xi\xi})/g_1=\cos\alpha\, x_{\xi\xi}+\sin\alpha\, y_{\xi\xi} \tag{2-18}$$

$$g_{1\eta}=(x_\xi x_{\xi\eta}+y_\xi y_{\xi\eta})/g_1=\cos\alpha\, x_{\xi\eta}+\sin\alpha\, y_{\xi\eta}=-g_2\alpha_\xi \tag{2-19}$$

$$g_{2\xi}=(x_\eta x_{\xi\eta}+y_\eta y_{\xi\eta})/g_2=\cos\alpha\, y_{\xi\eta}-\sin\alpha\, x_{\xi\eta}=g_1\alpha_\eta \tag{2-20}$$

$$g_{2\eta}=(x_\eta x_{\eta\eta}+y_\eta y_{\eta\eta})/g_2=\cos\alpha\, y_{\eta\eta}-\sin\alpha\, x_{\eta\eta} \tag{2-21}$$

が得られる．式(2-19)と(2-20)の関係は，単位ベクトルの直交性より導入されるもので，後述のように式の簡素化に役立つ重要な式である．つぎに，曲線座標の曲率ならびに曲率

半径について考える．

曲線座標 ξ,η の曲率 κ_1,κ_2，曲率半径 ρ_1,ρ_2 は図3に示す幾何学的関係である．図からもわかるように，

$$\rho_1 d\alpha = ds = g_1 d\xi,\quad \rho_2 d\alpha = dt = g_2 d\eta \tag{2-22}$$

の関係がある．この式(2-2)により，次のように与えられる．

$$\kappa_1 = \frac{1}{\rho_1} = \frac{\alpha_\xi}{g_1} = -\frac{1}{g_1 g_2}\frac{\partial g_1}{\partial \eta},\quad \kappa_2 = \frac{1}{\rho_2} = \frac{\alpha_\eta}{g_2} = \frac{1}{g_2 g_1}\frac{\partial g_2}{\partial \xi} \tag{2-23}$$

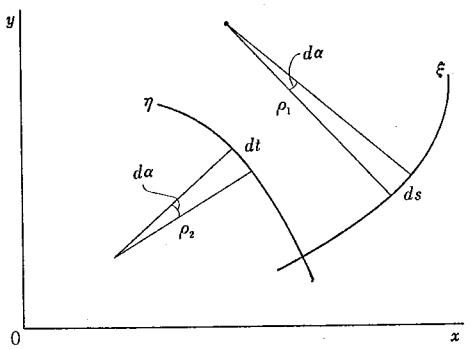


図 3　曲線座標の曲率半径

Fig. 3　Radius of carvature of curvilinear coordinate

3.　1変数場の微分方程式

ξ,η の変数を $u(\xi,\eta)$ とするとき，その微分

$$du(\xi,\eta) = \frac{\partial u}{\partial \xi}d\xi + \frac{\partial u}{\partial \eta}d\eta = \frac{1}{g_1}\frac{\partial u}{\partial \xi}ds + \frac{1}{g_2}\frac{\partial u}{\partial \eta}dt \tag{3-1}$$

により，歪ベクトル $\varepsilon_1,\varepsilon_2$ を導入する．

$$\varepsilon_1 = \frac{1}{g_1}\frac{\partial u}{\partial \xi},\quad \varepsilon_2 = \frac{1}{g_2}\frac{\partial u}{\partial \eta} \tag{3-2}$$

つぎに，連続体解析でよく利用される微分式の曲線座標系による表示を示す．

$$\mathrm{grad}\, u = i\frac{1}{g_1}\frac{\partial u}{\partial \xi} + i\frac{1}{g_2}\frac{\partial u}{\partial \eta} = i\varepsilon_1 + i\varepsilon_2 \tag{3-3}$$

$$\begin{aligned}\Delta u &= \left(i\frac{1}{g_1}\frac{\partial}{\partial \xi} + j\frac{1}{g_2}\frac{\partial}{\partial \eta}\right)\cdot\left(i\frac{1}{g_1}\frac{\partial u}{\partial \xi} + j\frac{1}{g_2}\frac{\partial u}{\partial \eta}\right)\\ &= \frac{1}{g_1}\frac{\partial}{\partial \xi}\left(\frac{1}{g_1}\frac{\partial u}{\partial \xi}\right) + \frac{1}{g_2}\frac{\partial}{\partial \eta}\left(\frac{1}{g_2}\frac{\partial u}{\partial \eta}\right) + \frac{\alpha_\eta}{g_1 g_2}\frac{\partial u}{\partial \xi} - \frac{\alpha_\xi}{g_1 g_2}\frac{\partial u}{\partial \eta}\\ &= \frac{1}{g_1^2}\frac{\partial^2 u}{\partial \xi^2} + \frac{1}{g_2^2}\frac{\partial^2 u}{\partial \eta^2} + \left(-\frac{1}{g_1^3}\frac{\partial g_1}{\partial \xi} + \frac{\alpha_\eta}{g_1 g_2}\right)\frac{\partial u}{\partial \xi} - \left(\frac{1}{g_2^3}\frac{\partial g_2}{\partial \eta} + \frac{\alpha_\xi}{g_1 g_2}\right)\frac{\partial u}{\partial \eta}\end{aligned} \tag{3-4}$$

式(3-4)は，(2-19)，(2-20)により，次のように変形される．

$$\Delta u = \frac{1}{g_1^2}\frac{\partial^2 u}{\partial \xi^2} + \frac{1}{g_2^2}\frac{\partial^2 u}{\partial \eta^2} + \frac{1}{g_1^2}\left(-\frac{1}{g_1}\frac{\partial g_1}{\partial \xi} + \frac{1}{g_2}\frac{\partial g_2}{\partial \xi}\right)\frac{\partial u}{\partial \xi} + \frac{1}{g_2^2}\left(\frac{1}{g_1}\frac{\partial g_1}{\partial \eta} - \frac{1}{g_2}\frac{\partial g_2}{\partial \eta}\right)\frac{\partial u}{\partial \eta} \tag{3-5}$$

とくに，$g_1 = g_2 = g$ ならば，(3-6)のように簡単な式に変形される．

$$\Delta u = \frac{1}{g^2}\left(\frac{\partial^2 u}{\partial \xi^2} + \frac{\partial^2 u}{\partial \eta^2}\right) \tag{3-6}$$

このように，$g_1 = g_2$ の直交曲線座標系は数値解析上重要な位置を占めるものである．

つぎに，さらに一般的な取り扱いについて述べる．いま，σ_1,σ_2 を $\varepsilon_1,\varepsilon_2$ に対応する応

力ベクトルとして，Green 積分

$$\iint(\sigma_1\delta\varepsilon_1+\sigma_2\delta\varepsilon_2)\,ds\,dt=\iint\left(\sigma_1\frac{1}{g_1}\frac{\partial\delta u}{\partial\xi}+\sigma_2\frac{1}{g_2}\frac{\partial\delta u}{\partial\eta}\right)g_1g_2\,d\xi\,d\eta$$

$$=\int\sigma\delta u\,d\lambda-\iint L\delta u\,ds\,dt \tag{3-7}$$

ただし，

$$\sigma=e_\xi g_2\sigma_1+e_\eta g_1\sigma_2\quad e_\xi=d\eta/d\lambda\quad e_\eta=-d\xi/d\lambda \tag{3-8}$$

$$L=\frac{1}{g_1g_2}\left\{\frac{\partial}{\partial\xi}(g_2\sigma_1)+\frac{\partial}{\partial\eta}(g_1\sigma_2)\right\}=\frac{\partial\sigma_1}{g_1\partial\xi}+\frac{\partial\sigma_2}{g_2\partial\eta}+\frac{\sigma_1}{\rho_2}-\frac{\sigma_2}{\rho_1} \tag{3-9}$$

を行う．

たとえば，構成方程式を

$$\sigma_1=k\varepsilon_1\quad\sigma_2=k\varepsilon_2 \tag{3-10}$$

とすると，

$$L=\frac{1}{g_1g_2}\left\{\frac{\partial}{\partial\xi}\left(k\frac{g_2}{g_1}\frac{\partial u}{\partial\xi}\right)+\frac{\partial}{\partial\eta}\left(k\frac{g_1}{g_2}\frac{\partial u}{\partial\eta}\right)\right\} \tag{3-11}$$

となる．

ここで，$k=1$ と置けば，式(3-11)は(3-5)と一致する．

さらに，一般には構成方程式

$$\sigma_1=k_{11}\varepsilon_1+k_{12}\varepsilon_2\quad\sigma_2=k_{21}\varepsilon_1+k_{22}\varepsilon_2 \tag{3-12}$$

を想定すればよい．とくに，自己随伴微分式を導入するときは $k_{21}=k_{12}$ とする．

4. 2変数場（ベクトル）の微分方程式

二つの変数を $u(\xi,\eta)$，$v(\xi,\eta)$ とするとき，これらが共にスカラ変数の場合と下記に示すベクトル変数の場合がありうる．ここでは，後者のベクトル変数のみを扱うこととし，

$$\boldsymbol{U}(\xi,\eta)=iu(\xi,\eta)+jv(\xi,\eta) \tag{4-1}$$

により，変数ベクトルを定義する．つぎにその微分により，歪テンソルを導入する．

$$\begin{aligned}d\boldsymbol{U}&=i\,du+j\,dv+u\,di+v\,dj\\&=i\left(\frac{\partial u}{\partial\xi}d\xi+\frac{\partial u}{\partial\eta}d\eta\right)+j\left(\frac{\partial v}{\partial\xi}d\xi+\frac{\partial v}{\partial\eta}d\eta\right)+(ju-iv)(\alpha_\xi d\xi+\alpha_\eta d\eta)\\&=i(e_{11}ds+e_{12}dt)+j(e_{21}ds+e_{22}dt)\end{aligned} \tag{4-2}$$

$$\begin{aligned}e_{11}&=\frac{1}{g_1}\left(\frac{\partial u}{\partial\xi}-\alpha_\xi v\right)=\frac{\partial u}{g_1\partial\xi}+\frac{\partial g_1}{g_2\partial\eta}\frac{v}{g_1}=\frac{\partial u}{g_1\partial\xi}-\frac{v}{\rho_1}\\e_{12}&=\frac{1}{g_2}\left(\frac{\partial u}{\partial\eta}-\alpha_\eta v\right)=\frac{\partial u}{g_2\partial\eta}-\frac{\partial g_2}{g_1\partial\xi}\frac{v}{g_2}=\frac{\partial u}{g_2\partial\eta}-\frac{v}{\rho_2}\\e_{21}&=\frac{1}{g_1}\left(\frac{\partial v}{\partial\xi}+\alpha_\xi u\right)=\frac{\partial v}{g_1\partial\xi}-\frac{\partial g_1}{g_2\partial\eta}\frac{u}{g_1}=\frac{\partial v}{g_1\partial\xi}+\frac{u}{\rho_1}\\e_{22}&=\frac{1}{g_2}\left(\frac{\partial v}{\partial\eta}+\alpha_\eta u\right)=\frac{\partial v}{g_2\partial\eta}+\frac{\partial g_2}{g_1\partial\xi}\frac{u}{g_2}=\frac{\partial v}{g_2\partial\eta}+\frac{u}{\rho_2}\end{aligned} \tag{4-3}$$

つぎに連続体解析において，よく利用されている微分式について述べる．

$$\begin{aligned}\text{div }\boldsymbol{U}&=\left(i\frac{\partial}{g_1\partial\xi}+j\frac{\partial}{g_2\partial\eta}\right)\cdot(iu+jv)\\&=\frac{\partial u}{g_1\partial\xi}+\frac{\partial v}{g_2\partial\eta}+\frac{\alpha_\eta}{g_2}u-\frac{\alpha_\xi}{g_1}v=\frac{\partial u}{g_1\partial\xi}+\frac{\partial v}{g_2\partial\eta}+\frac{u}{\rho_2}-\frac{v}{\rho_1}\end{aligned} \tag{4-4}$$

$$\text{rot }\boldsymbol{U}=\left(i\frac{\partial}{g_1\partial\xi}+j\frac{\partial}{g_2\partial\eta}\right)\times(iu+jv)$$

$$=k\left(\frac{\partial v}{g_1\partial\xi}-\frac{\partial u}{g_2\partial\eta}+\frac{\alpha_\xi}{g_1}u+\frac{\alpha_\eta}{g_2}v\right)$$

$$=k\left(\frac{\partial v}{g_1\partial\xi}-\frac{\partial u}{g_2\partial\eta}+\frac{u}{\rho_1}+\frac{v}{\rho_2}\right)=k(-e_{12}+e_{21}) \tag{4-5}$$

したがって，$e_{12}=e_{21}$ ならば $\mathrm{rot}\,\boldsymbol{U}=0$ である．

つぎに $\boldsymbol{U}$ を $\mathrm{grad}\,U$ で置き換えれば(2-19), (2-20)により，

$$\mathrm{rot}\,\mathrm{grad}\,U=k\left\{\frac{\partial}{g_1\partial\xi}\left(\frac{\partial u}{g_2\partial\eta}\right)-\frac{\partial}{g_2\partial\eta}\left(\frac{\partial u}{g_1\partial\xi}\right)+\frac{\alpha_\xi}{g_1{}^2}\frac{\partial u}{\partial\xi}+\frac{\alpha_\eta}{g_2{}^2}\frac{\partial u}{\partial\eta}\right\}$$

$$=k\left\{\frac{1}{g_1{}^2}\left(\alpha_\xi+\frac{\partial g_1}{g_2\partial\eta}\right)\frac{\partial u}{\partial\xi}+\frac{1}{g_2{}^2}\left(\alpha_\eta-\frac{\partial g_2}{g_1\partial\xi}\right)\frac{\partial u}{\partial\eta}\right\}=0 \tag{4-6}$$

となる．

つぎに，Green 積分による一般的な取り扱いを行う．歪テンソル $e_{11}, e_{12}, e_{21}, e_{22}$ に対応する応力テンソルを $\sigma_{11}, \sigma_{12}, \sigma_{21}, \sigma_{22}$ として下記の積分を行う．

$$\delta U=\iint(\sigma_{11}\delta e_{11}+\sigma_{12}\delta e_{12}+\sigma_{21}\delta e_{21}+\sigma_{22}\delta e_{22})dsdt$$

$$=\iiint\left\{\sigma_{11}\left(\frac{\partial\delta u}{g_1\partial\xi}-\frac{\delta v}{\rho_1}\right)+\sigma_{12}\left(\frac{\partial\delta u}{g_2\partial\eta}-\frac{\delta v}{\rho_2}\right)\right.$$

$$\left.+\sigma_{21}\left(\frac{\partial\delta v}{g_1\partial\xi}+\frac{\delta u}{\rho_1}\right)+\sigma_{22}\left(\frac{\partial\delta v}{g_2\partial\eta}+\frac{\delta u}{\rho_2}\right)\right\}g_1g_2d\xi d\eta$$

$$=\int(\sigma_1\delta u+\sigma_2\delta v)d\lambda-\iint(L_1\delta u+L_2\delta v)dsdt \tag{4-7}$$

$$\sigma_1=e_\xi g_2\sigma_{11}+e_\eta g_1\sigma_{12},\quad \sigma_2=e_\xi g_2\sigma_{21}+e_\eta g_1\sigma_{22} \tag{4-8}$$

$$L_1=\frac{\partial\sigma_{11}}{g_1\partial\xi}+\frac{\partial\sigma_{12}}{g_2\partial\eta}-\frac{\sigma_{12}+\sigma_{21}}{\rho_1}+\frac{\sigma_{11}-\sigma_{22}}{\rho_2} \tag{4-9}$$

$$L_2=\frac{\partial\sigma_{21}}{g_1\partial\xi}+\frac{\partial\sigma_{22}}{g_2\partial\eta}+\frac{\sigma_{11}-\sigma_{22}}{\rho_1}+\frac{\sigma_{12}+\sigma_{21}}{\rho_2} \tag{4-10}$$

応力テンソルが対称 $\sigma_{12}=\sigma_{21}$ のときは，歪テンソルを

$$\varepsilon_{11}=e_{11},\quad \varepsilon_{12}=e_{12}+e_{21},\quad \varepsilon_{22}=e_{22} \tag{4-11}$$

と置いて，

$$\delta U=\iint(\sigma_{11}\delta\varepsilon_{11}+\sigma_{12}\delta\varepsilon_{12}+\sigma_{22}\delta\varepsilon_{22})dsdt$$

$$=\int(\sigma_1\delta u+\sigma_2\delta v)d\lambda-\iint(L_1\delta u+L_2\delta v)dsdt \tag{4-12}$$

をつくる．ただし，

$$L_1=\frac{\partial\sigma_{11}}{g_1\partial\xi}+\frac{\partial\sigma_{12}}{g_2\partial\eta}-2\frac{\sigma_{12}}{\rho_1}+\frac{\sigma_{11}-\sigma_{22}}{\rho_2} \tag{4-13}$$

$$L_2=\frac{\partial\sigma_{12}}{g_1\partial\xi}+\frac{\partial\sigma_{22}}{g_2\partial\eta}+2\frac{\sigma_{12}}{\rho_2}+\frac{\sigma_{11}-\sigma_{22}}{\rho_1} \tag{4-14}$$

である．

このように構成方程式が x-y 座標系で定義されているとき，直交曲線座標系でも微視的には x-y 座標系とまったく同様であるから，そのままの形の構成方程式を採用することができる．たとえば，x-y 座標系における構成方程式が，

$$\sigma_{im}=k_{im,jn}\varepsilon_{jn} \tag{4-15}$$

であるならば，曲線座標系でも同様の式を用いることができる．たとえば，直接応力 σ_{xx}

を省略した平板の面内変形問題で，構成方程式は

$$\begin{Bmatrix}\sigma_{ss}\\ \sigma_{st}\\ \sigma_{tt}\end{Bmatrix}=E'\begin{bmatrix}1 & & \nu\\ & \alpha & \\ \nu & & 1\end{bmatrix}\begin{Bmatrix}\varepsilon_{ss}=e_{ss}\\ \varepsilon_{st}=e_{st}+e_{ts}\\ \varepsilon_{tt}=e_{tt}\end{Bmatrix} \tag{4-16}$$

によって与えられる．ただし，$E'=E/(1-\nu^2)$，$\alpha=(1-\nu)/2$ である．平板に作用する面力を S,T とし，平衡方程式は式(4-13)，(4-14)により

$$L_1+S=0,\quad L_2+T=0 \tag{4-17}$$

により与えられる．ここでは 板厚 h=一定 として，

$$L_1=E'\left\{\frac{\partial}{g_1\partial\xi}(\varepsilon_{ss}+\nu\varepsilon_{tt})+\alpha\frac{\partial\varepsilon_{st}}{g_2\partial\eta}+(1-\nu)\left(-\frac{\varepsilon_{st}}{\rho_1}+\frac{\varepsilon_{ss}-\varepsilon_{tt}}{\rho_2}\right)\right\}$$

$$L_2=E'\left\{\alpha\frac{\partial\varepsilon_{st}}{g_1\partial\xi}+\frac{\partial}{g_2\partial\eta}(\nu\varepsilon_{ss}+\varepsilon_{tt})+(1-\nu)\left(\frac{\varepsilon_{ss}-\varepsilon_{tt}}{\rho_1}+\frac{\varepsilon_{st}}{\rho_2}\right)\right\}$$

となる．図4に示されるように x-y 座標系では，dx,dy，また曲線座標系では，ds,dt を定数とする微視的格子系は回転 α を除いて同等である．したがって，連続体が等方性ならば同じ構成方程式を用いることができる．

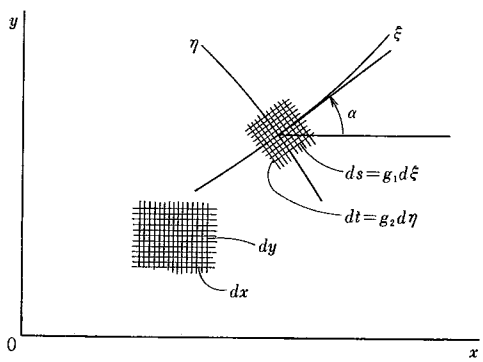


図 4　x-y, ξ-η 座標系の微視的格子系

Fig. 4　Microscopic network of x-y and ξ-η coordinate system

5.　4階微分方程式

板曲げや，2次元 N-S 方程式（流れ関数法）などの挙動は4階の微分方程式によって記述されている．いまスカラ変数を $u(\xi,\eta)$ として，まずその1次の歪テンソルを求める．

$$\boldsymbol{\varepsilon}=\mathrm{grad}\,u=i\frac{\partial u}{g_1\partial\xi}+j\frac{\partial u}{g_2\partial\eta}=i\varepsilon_1+j\varepsilon_2 \tag{5-1}$$

つぎに，その微分により2次の歪テンソルを導く．

$$\begin{aligned}d\boldsymbol{\varepsilon}=d(i\varepsilon_1+j\varepsilon_2)&=\frac{\partial}{\partial\xi}\left(i\frac{\partial u}{g_1\partial\xi}+j\frac{\partial u}{g_2\partial\eta}\right)d\xi+\frac{\partial}{\partial\eta}\left(i\frac{\partial u}{g_1\partial\xi}+j\frac{\partial u}{g_2\partial\eta}\right)d\eta\\&=i(e_{11}ds+e_{12}dt)+j(e_{21}ds+e_{22}dt)\end{aligned} \tag{5-2}$$

$$\left.\begin{aligned}e_{11}&=\frac{\partial}{g_1\partial\xi}\left(\frac{\partial u}{g_1\partial\xi}\right)-\frac{\alpha_\xi}{g_1}\frac{\partial u}{g_2\partial\eta}=\frac{\partial}{g_1\partial\xi}\left(\frac{\partial u}{g_1\partial\xi}\right)-\frac{1}{\rho_1}\frac{\partial u}{g_2\partial\eta}\\e_{12}&=\frac{\partial}{g_2\partial\eta}\left(\frac{\partial u}{g_1\partial\xi}\right)-\frac{\alpha_\eta}{g_2}\frac{\partial u}{g_2\partial\eta}=\frac{\partial}{g_2\partial\eta}\left(\frac{\partial u}{g_1\partial\xi}\right)-\frac{1}{\rho_2}\frac{\partial u}{g_2\partial\eta}\\e_{21}&=\frac{\partial}{g_1\partial\xi}\left(\frac{\partial u}{g_2\partial\eta}\right)+\frac{\alpha_\xi}{g_1}\frac{\partial u}{g_1\partial\xi}=\frac{\partial}{g_1\partial\xi}\left(\frac{\partial u}{g_2\partial\eta}\right)+\frac{1}{\rho_1}\frac{\partial u}{g_1\partial\xi}\end{aligned}\right\} \tag{5-3}$$

$$e_{22}=\frac{\partial}{g_2\partial\eta}\left(\frac{\partial u}{g_2\partial\eta}\right)+\frac{\alpha_\eta}{g_2}\frac{\partial u}{g_1\partial\xi}=\frac{\partial}{g_2\partial\eta}\left(\frac{\partial u}{g_2\partial\eta}\right)+\frac{1}{\rho_2}\frac{\partial u}{g_1\partial\xi}$$

歪テンソル e_{mn} に対応する応力テンソルを σ_{mn} として，つぎに Green 積分を実行する．

$$\delta U=\iint\sigma_{mn}\cdot\delta e_{mn}\ dsdt=\int(\sigma_1\delta\varepsilon_1+\sigma_2\delta\varepsilon_2+\sigma\delta u)d\lambda+\iint L\delta u dsdt \tag{5-4}$$

$$\sigma_1=e_\xi g_2\sigma_{11}+e_\eta g_1\sigma_{12},\quad \sigma_2=e_\xi g_2\sigma_{21}+e_\eta g_1\sigma_{22} \tag{5-5}$$

$$\sigma_1{}^*=\frac{1}{g_1}\left\{\frac{\partial}{\partial\xi}(g_2\sigma_{11})+\frac{\partial}{\partial\eta}(g_1\sigma_{12})+\frac{\partial g_1}{\partial\eta}\sigma_{21}-\frac{\partial g_2}{\partial\xi}\sigma_{22}\right\} \tag{5-6}$$

$$\sigma_2{}^*=\frac{1}{g_2}\left\{\frac{\partial}{\partial\xi}(g_2\sigma_{21})+\frac{\partial}{\partial\eta}(g_1\sigma_{22})-\frac{\partial g_1}{\partial\eta}\sigma_{11}+\frac{\partial g_2}{\partial\xi}\sigma_{12}\right\} \tag{5-7}$$

$$\sigma=-(e_\xi\sigma_1{}^*+e_\eta\sigma_2{}^*) \tag{5-8}$$

$$\begin{aligned}L&=\frac{1}{g_1g_2}\left(\frac{\partial\sigma_1{}^*}{\partial\xi}+\frac{\partial\sigma_2{}^*}{\partial\eta}\right)\\&=\frac{\partial}{g_1\partial\xi}\left(\frac{\partial\sigma_{11}}{g_1\partial\xi}+\frac{\partial\sigma_{12}}{g_2\partial\eta}+\frac{\partial g_1}{g_2\partial\eta}\frac{\sigma_{12}+\sigma_{21}}{g_1}+\frac{\partial g_2}{g_1\partial\xi}\frac{\sigma_{11}-\sigma_{22}}{g_2}\right)\\&\quad+\frac{\partial}{g_2\partial\eta}\left(\frac{\partial\sigma_{21}}{g_1\partial\xi}+\frac{\partial\sigma_{22}}{g_2\partial\eta}+\frac{\partial g_1}{g_2\partial\eta}\frac{\sigma_{22}-\sigma_{11}}{g_1}+\frac{\partial g_2}{g_1\partial\xi}\frac{\sigma_{12}+\sigma_{21}}{g_2}\right)\\&\quad+\frac{1}{g_2}\frac{\partial g_2}{g_1\partial\xi}\left(\frac{\partial\sigma_{11}}{g_1\partial\xi}+\frac{\partial\sigma_{12}}{g_2\partial\eta}+\frac{\partial g_1}{g_2\partial\eta}\frac{\sigma_{12}+\sigma_{21}}{g_1}+\frac{\partial g_2}{g_1\partial\xi}\frac{\sigma_{11}-\sigma_{22}}{g_2}\right)\\&\quad+\frac{1}{g_1}\frac{\partial g_1}{g_2\partial\eta}\left(\frac{\partial\sigma_{21}}{g_1\partial\xi}+\frac{\partial\sigma_{22}}{g_2\partial\eta}+\frac{\partial g_1}{g_2\partial\eta}\frac{\sigma_{22}-\sigma_{11}}{g_1}+\frac{\partial g_2}{g_1\partial\xi}\frac{\sigma_{12}+\sigma_{21}}{g_2}\right)\end{aligned} \tag{5-9}$$

とくに，等測度係数系では $g_1=g_2=g$ として，

$$\sigma_1{}^*=\frac{\partial\sigma_{11}}{\partial\xi}+\frac{\partial\sigma_{12}}{\partial\eta}+\frac{1}{g}\left\{\frac{\partial g}{\partial\xi}(\sigma_{11}-\sigma_{22})+\frac{\partial g}{\partial\eta}(\sigma_{12}+\sigma_{21})\right\} \tag{5-10}$$

$$\sigma_2{}^*=\frac{\partial\sigma_{21}}{\partial\xi}+\frac{\partial\sigma_{22}}{\partial\eta}+\frac{1}{g}\left\{\frac{\partial g}{\partial\xi}(\sigma_{12}+\sigma_{21})+\frac{\partial g}{\partial\eta}(\sigma_{22}-\sigma_{11})\right\} \tag{5-11}$$

$$\begin{aligned}L&=\frac{1}{g^2}\left\{\frac{\partial^2\sigma_{11}}{\partial\xi^2}+\frac{\partial^2}{\partial\xi\partial\eta}(\sigma_{12}+\sigma_{21})+\frac{\partial^2\sigma_{22}}{\partial\eta^2}\right\}\\&\quad+\frac{1}{g^3}\left\{\frac{\partial g}{\partial\xi}\left(\frac{\partial\sigma_{11}}{\partial\xi}-\frac{\partial\sigma_{22}}{\partial\xi}+\frac{\partial\sigma_{12}}{\partial\eta}+\frac{\partial\sigma_{21}}{\partial\eta}\right)+\frac{\partial g}{\partial\eta}\left(-\frac{\partial\sigma_{11}}{\partial\eta}+\frac{\partial\sigma_{22}}{\partial\eta}+\frac{\partial\sigma_{12}}{\partial\xi}+\frac{\partial\sigma_{21}}{\partial\xi}\right)\right\}\\&\quad+\left\{\frac{1}{g^3}\left(\frac{\partial^2 g}{\partial\xi^2}-\frac{\partial^2 g}{\partial\eta^2}\right)+\frac{1}{g^4}\left(-\left(\frac{\partial g}{\partial\xi}\right)^2+\left(\frac{\partial g}{\partial\eta}\right)^2\right)\right\}(\sigma_{11}-\sigma_{22})\\&\quad+2\left(\frac{1}{g^3}\frac{\partial^2 g}{\partial\xi\partial\eta}-\frac{1}{g^4}\frac{\partial g}{\partial\xi}\frac{\partial g}{\partial\eta}\right)(\sigma_{12}+\sigma_{21})\end{aligned} \tag{5-12}$$

となる．

2重 Laplace 式は式(3-4)より，

$$\Delta^2u=\left\{\frac{\partial^2}{g_1{}^2\partial\xi^2}+\frac{\partial^2}{g_2{}^2\partial\eta^2}+\frac{1}{g_1{}^2}\left(\frac{1}{g_2}\frac{\partial g_2}{\partial\xi}-\frac{1}{g_1}\frac{\partial g_1}{\partial\xi}\right)\frac{\partial}{\partial\xi}+\frac{1}{g_2{}^2}\left(\frac{1}{g_1}\frac{\partial g_1}{\partial\eta}-\frac{1}{g_2}\frac{\partial g_2}{\partial\eta}\right)\frac{\partial}{\partial\eta}\right\}^2u \tag{5-13}$$

である．

とくに，$g_1=g_2=g$ ならば，

$$\begin{aligned}\Delta^2u&=\frac{1}{g^2}\left(\frac{\partial^2}{\partial\xi^2}+\frac{\partial^2}{\partial\eta^2}\right)\frac{1}{g^2}\left(\frac{\partial^2u}{\partial\xi^2}+\frac{\partial^2u}{\partial\eta^2}\right)\\&=\frac{1}{g^4}\left[\frac{\partial^2}{\partial\xi^2}+\frac{\partial^2}{\partial\eta^2}-\frac{4}{g}\left(\frac{\partial g}{\partial\xi}\frac{\partial}{\partial\xi}+\frac{\partial g}{\partial\eta}\frac{\partial}{\partial\eta}\right)\right.\\&\quad\left.-2\left\{\frac{1}{g}\left(\frac{\partial^2 g}{\partial\xi^2}+\frac{\partial^2 g}{\partial\eta^2}\right)-\frac{3}{g^2}\left(\left(\frac{\partial g}{\partial\xi}\right)^2+\left(\frac{\partial g}{\partial\eta}\right)^2\right)\right\}\right]\left(\frac{\partial^2u}{\partial\xi^2}+\frac{\partial^2u}{\partial\eta^2}\right)\end{aligned} \tag{5-14}$$

$$=\frac{1}{g^4}\left[\frac{\partial^4 u}{\partial\xi^4}+2\frac{\partial^4 u}{\partial\xi^2\partial\eta^2}+\frac{\partial^4 u}{\partial\eta^4}-\frac{4}{g}\left(\frac{\partial g}{\partial\xi}\frac{\partial^3 u}{\partial\xi^3}+\frac{\partial g}{\partial\eta}\frac{\partial^3 u}{\partial\xi^2\partial\eta}+\frac{\partial g}{\partial\xi}\frac{\partial^3 u}{\partial\xi\partial\eta^2}+\frac{\partial g}{\partial\eta}\frac{\partial^3 u}{\partial\eta^3}\right)\right.$$

$$\left.-2\left\{\frac{1}{g}\left(\frac{\partial^2 g}{\partial\xi^2}+\frac{\partial^2 g}{\partial\eta^2}\right)-\frac{3}{g^2}\left(\left(\frac{\partial g}{\partial\xi}\right)^2+\left(\frac{\partial g}{\partial\eta}\right)^2\right)\right\}\left(\frac{\partial^2 u}{\partial\xi^2}+\frac{\partial^2 u}{\partial\eta^2}\right)\right] \qquad (5\text{-}15)$$

となる．

つぎに，実際の直交曲線座標系の代表例について簡単に述べよう．

6. 極座標系

極座標系は $\xi=r$, $\eta=\theta$ とする直交曲線座標系である(図5)．第2章の諸式により，次の微分幾何定数を定める．

$x=r\cos\theta$, $y=r\sin\theta$, $x_r=\cos\theta$, $x_\theta=-r\sin\theta$, $x_{rr}=\theta$,
$x_{r\theta}=-\sin\theta$, $x_{\theta\theta}=-r\cos\theta$, $y_r=\sin\theta$, $y_\theta=r\cos\theta$, $y_{rr}=0$,
$y_{r\theta}=\cos\theta$, $y_{\theta\theta}=-r\sin\theta$, $g_1=1$, $g_2=r$, $\alpha=\theta$, $\alpha_r=0$,
$\alpha_\theta=1$, $\rho_1=\infty$, $\rho_2=r$

このように極座標系は不等測度係数系で，図6に示される扇形は解析面では図7に示される直交直線格子系に移される．この図でわかるように解析面で格子定数を一定とすると，物理面では網目の形は不揃いとなる．

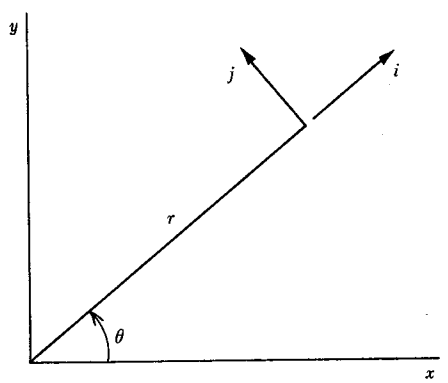


図 5 極座標系

Fig. 5 Poler coordinate system

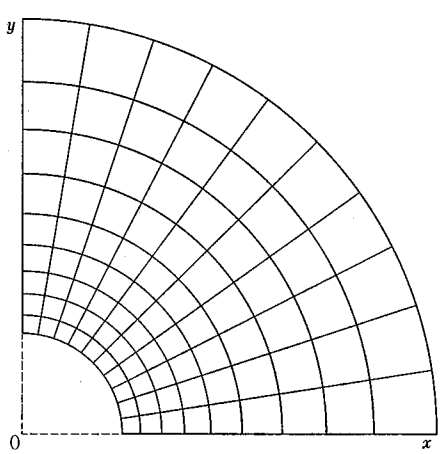


図 6 極座標系での物理面

Fig. 6 Poler coordinate system (plane of physics)

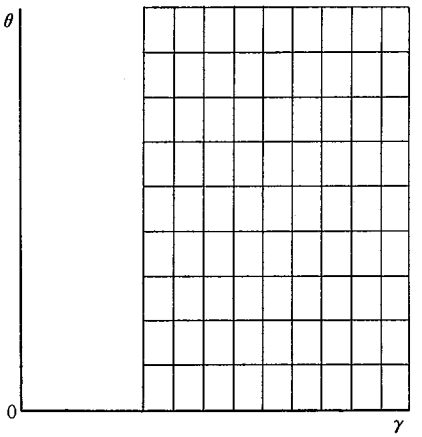


図 7 極座標系での解析面

Fig. 7 Poler coordinate system (plane of analysis)

この座標系で変数がスカラ変数の場合と，ベクトル変数の場合に分けて述べる．

1) スカラ変数 $u(r,\theta)$

歪： $\varepsilon_1=\dfrac{\partial u}{\partial r}$, $\varepsilon_2=\dfrac{\partial u}{r\partial\theta}$

$$\Delta u=\frac{\partial^2 u}{\partial r^2}+\frac{1}{r}\frac{\partial u}{\partial r}+\frac{1}{r^2}\frac{\partial^2 u}{\partial\theta^2}$$

$$\Delta^2 u=\frac{\partial^4 u}{\partial r^4}+\frac{2}{r^2}\frac{\partial^4 u}{\partial r^2\partial\theta^2}+\frac{1}{r^4}\frac{\partial^4 u}{\partial\theta^4}+\frac{2}{r}\frac{\partial^3 u}{\partial r^3}-\frac{2}{r^3}\frac{\partial^3 u}{\partial r\partial\theta^2}-\frac{1}{r^2}\frac{\partial^2 u}{\partial r^2}+\frac{4}{r^2}\frac{\partial^2 u}{\partial\theta^2}+\frac{1}{r^3}\frac{\partial u}{\partial r}$$

2) ベクトル変数 $\boldsymbol{U}=iu(r,\theta)+jv(r,\theta)$

歪： $e_{11}=\dfrac{\partial u}{\partial r}$, $e_{12}=\dfrac{1}{r}\left(\dfrac{\partial u}{\partial\theta}-v\right)$, $e_{21}=\dfrac{\partial v}{\partial r}$, $e_{22}=\dfrac{1}{r}\left(\dfrac{\partial v}{\partial\theta}+u\right)$

$\mathrm{div}\,\boldsymbol{U}=\dfrac{\partial u}{\partial r}+\dfrac{\partial v}{r\partial\theta}+\dfrac{u}{r}$, $\mathrm{rot}\,\boldsymbol{U}=k(-e_{12}+e_{21})$

平板の面内変形

$$\frac{\partial^2 u}{\partial r^2}+\frac{1}{r}\frac{\partial u}{\partial r}-\frac{u}{r^2}+\alpha\frac{\partial^2 u}{r^2\partial\theta^2}+\beta\frac{\partial^2 v}{r\partial r\partial\theta}-\gamma\frac{\partial v}{r^2\partial\theta}+\frac{R}{k}=0$$

$$\beta\frac{\partial^2 u}{r\partial u\partial\theta}+\gamma\frac{\partial u}{r^2\partial\theta}+\alpha\frac{\partial^2 v}{\partial r^2}+\alpha\frac{\partial v}{r\partial r}+\frac{\partial^2 v}{r^2\partial\theta^2}-\alpha\frac{v}{r^2}+\frac{T}{k}=0$$

ただし，$\alpha=(1-\nu)/2$, $\beta=(1+\nu)/2$, $\gamma=(3-\nu)/2$ である．

7. 対数極座標系

$$\xi=\log(r/r_0),\quad (r=r_0e^{\xi}),\quad \eta=\theta$$

により，曲線座標 ξ,η を次のように定義する．

$x=r_0e^{\xi}\cos\eta$, $x_\xi=r_0e^{\xi}\cos\eta$, $x_\eta=-r_0e^{\xi}\sin\eta$, $x_{\xi\xi}=r_0e^{\xi}\cos\eta$,
$x_{\xi\eta}=-r_0e^{\xi}\sin\xi$, $x_{\eta\eta}=-r_0e^{\xi}\cos\eta$, $y=r_0e^{\xi}\sin\eta$, $y_\xi=r_0e^{\xi}\sin\eta$,
$y_\eta=r_0e^{\xi}\cos\eta$, $y_{\xi\xi}=r_0e^{\xi}\sin\eta$, $y_{\xi\eta}=r_0e^{\xi}\cos\eta$, $y_{\eta\eta}=-r_0e^{\xi}\sin\eta$,
$g_1=g_2=g=r_0e^{\xi}=r$, $\alpha=\theta$, $\alpha_\xi=0$, $\alpha_\eta=1$, $\rho_1=\infty$, $\rho_2=r$,
$g_\xi=r$, $g_\eta=0$, $g_{\xi\xi}=r$, $g_{\eta\eta}=0$, $g_{\eta\eta}=0$

以上のように対数極座標系は等測度係数で，図8，図9に示されるように物理平面における格子系の網目は一様である．

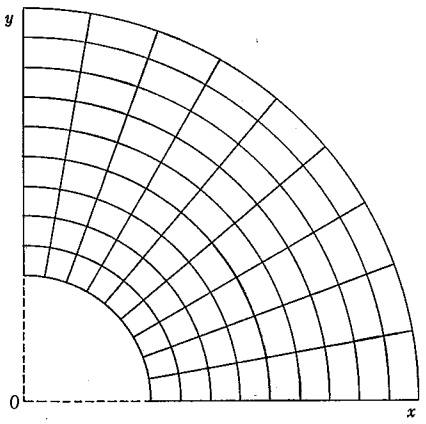


図 8 対数極座標系の解析平面

Fig. 8 Logarithmic poler coordinate system (plane of physics)

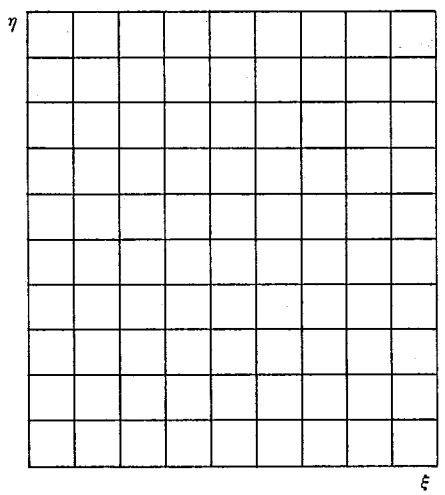


図 9 対数極座標系の物理平面

Fig. 9 Logarithmic poler coordinate system (plane of analysis)

対数極座標系での微分方程式を,変数がスカラーのときとベクトルのときについて述べる.

1) スカラ変数 $u(\xi, \eta)$

歪: $\varepsilon_1=\dfrac{e^{-\xi}}{r_0}\dfrac{\partial u}{\partial \xi}$, $\varepsilon_2=\dfrac{e^{-\xi}}{r_0}\dfrac{\partial u}{\partial \eta}$

$$\Delta u=\frac{e^{2\xi}}{r_0^2}\left(\frac{\partial^2 u}{\partial \xi^2}+\frac{\partial^2 u}{\partial \eta^2}\right)$$

$$\Delta^2 u=\frac{e^{4\xi}}{r_0^4}\left\{\frac{\partial^4 u}{\partial \xi^4}+2\frac{\partial^4 u}{\partial \xi^2 \partial \eta^2}+\frac{\partial^4 u}{\partial \eta^4}-4\left(\frac{\partial^3 u}{\partial \xi^3}+\frac{\partial^3 u}{\partial \xi \partial \eta^2}\right)+4\left(\frac{\partial^2 u}{\partial \xi^2}+\frac{\partial^2 u}{\partial \eta^2}\right)\right\}$$

2) ベクトル変数 $\boldsymbol{U}=iu(\xi, \eta)+jv(\xi, \eta)$

歪: $e_{11}=\dfrac{e^{-\xi}}{r_0}\dfrac{\partial u}{\partial \xi}$, $e_{12}=\dfrac{e^{-\xi}}{r_0}\left(\dfrac{\partial u}{\partial \eta}-v\right)$, $e_{21}=\dfrac{e^{-\xi}}{r_0}\dfrac{\partial v}{\partial \xi}$, $e_{22}=\dfrac{e^{-\xi}}{r_0}\left(\dfrac{\partial v}{\partial \eta}+u\right)$

$$\mathrm{div}\ \boldsymbol{U}=\frac{e^{-\xi}}{r_0}\left(\frac{\partial u}{\partial \xi}+\frac{\partial v}{\partial \eta}+u\right),\quad \mathrm{rot}\ \boldsymbol{U}=\frac{e^{-\xi}}{r_0}\left(-\frac{\partial u}{\partial \eta}+\frac{\partial v}{\partial \eta}+v\right)$$

平板の面内変形を行うと,次のように表せる.

$$\frac{\partial^2 u}{\partial \xi^2}+\alpha\frac{\partial^2 u}{\partial \eta^2}-u+\beta\frac{\partial^2 v}{\partial \xi \partial \eta}-\gamma\frac{\partial v}{\partial \eta}+\frac{k}{r_0}Re^{\xi}=0$$

$$\beta\frac{\partial^2 u}{\partial \xi \partial \eta}+\gamma\frac{\partial u}{\partial \eta}+\alpha\frac{\partial^2 v}{\partial \xi^2}+\frac{\partial^2 v}{\partial \eta^2}-\alpha v+\frac{k}{r_0}Te^{\xi}=0$$

このように極座標系による微分方程式よりも，格段に単純化されていることが了解される．

8. 楕円座標系

ξ, η を曲線座標とするとき，各式は次のように定義される．

$x=\cosh\xi\cos\eta$, $y=\sinh\xi\sin\eta$

$$\left(\frac{x}{\cosh\xi}\right)^2+\left(\frac{y}{\sinh\xi}\right)^2=1,\quad \left(\frac{x}{\cos\eta}\right)^2-\left(\frac{y}{\sin\eta}\right)^2=1$$

$x_\xi=\sinh\xi\cos\eta$, $x_\eta=-\cosh\xi\sin\eta$, $x_{\xi\xi}=\cosh\xi\cos\eta$, $x_{\xi\eta}=-\sinh\xi\sin\eta$,

$x_{\eta\eta}=-\cosh\xi\cos\eta$, $y_\xi=\cosh\xi\sin\eta$, $y_\eta=\sinh\xi\cos\eta$, $y_{\xi\xi}=\sinh\xi\sin\eta$, $y_{\xi\eta}=\cosh\xi\cos\eta$,

$y_{\eta\eta}=-\sinh\xi\sin\eta$, $g_1=g_2=\sqrt{\sinh^2\xi\cos^2\eta+\cosh^2\xi\sin^2\eta}=\sqrt{\sinh^2\xi+\sin^2\eta}=g$,

$g_\xi=\sinh\xi\cosh\xi/g$, $g_\eta=\sin\eta\cos\eta/g$, $\rho_1=-\sin\eta\cos\eta/g^3$, $\rho_2=\sinh\xi\cosh\xi/g^3$

したがって，この曲線座標系は等測度係数である（図10）．

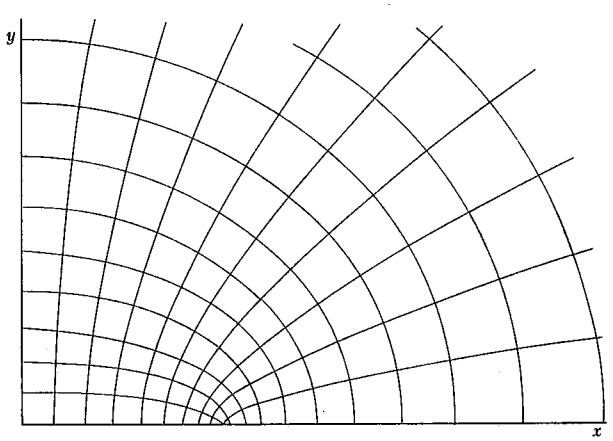


図 10 楕円座標系

Fig. 10 Eliptic coordinate system

9. 任意境界形に応じる座標変換差分法

座標変換差分法の利用を既存の直交曲線座標系に限定せずに，さらにその範囲を拡大するためには任意形状の領域内に直交曲線座標系を確立することが必要である．たとえば，図11, 12 に示すような単重，2重連結領域を想定する．ただし単重連結領域（図11）において各境界 s_1, s_2, s_2, s_4 は，たがいに直交するものとする．非直交のときは，交点は流体力学における澱み点と同様に特異点となり，計算精度の低下を招くこととなる．

たとえば，図11で変数 ξ はその領域内部の Laplace 式を満たし，s_1 上で $\xi=0$, s_3 上で $\xi=1$, s_2, s_4 上で $\xi_n=0$ を満たすものとする．同様に η 変数も領域内で Laplace の式

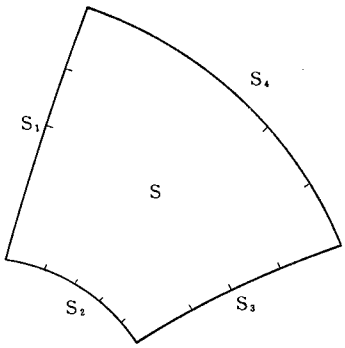


図 11　単重連結領域
Fig. 11　Singly conected region

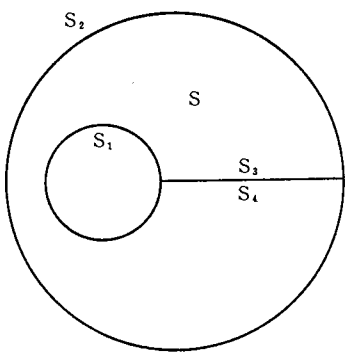


図 12　2 重連結領域
Fig. 12　Doubly conected region

を満たし，境界，s_1, s_3 上で $\eta_n=0$，s_2 上で $\eta=0$，s_4 上で $\eta=1$ を満たすものとする．このとき ξ,η は，領域内で直交曲線座標系となっている．同様に図9で，ξ は領域内で Laplace の式を満たし，境界 s_1 上で$\xi=0$，s_2 上で $\xi=1$ を満たすものとする．この領域を単重連結とするため ξ に直交するように cut を入れ，その両側を s_3, s_4 とする．変数 η も領域内で Laplace の式を満たし，s_1, s_2 上で $\eta_\eta=0$，s_3 上で $\eta=0$，s_4 上で $\eta=1$ を満たすものとする．このようにして定められた ξ,η は，同様に直交曲線座標系を構成する．任意形状の境界を持つ領域について ξ,η を求めるためには有限要素法，または任意節点配置差分法の助けを借りることが必要である．

この他に複素関数論を利用することもできる．よく知られているように複素数 $z=x+iy$ の正則な関数を

$$f(z)=\xi+i\eta$$

とするとき，ξ,η はおのおの調和関数で，ξ,η は直交曲線座標系となっている．

このようにして直交曲線座標系が確立されれば，物理平面の点 (x,y) と解析平面内の点 (ξ,η) は1対1に対応するので，解析平面上の直交直線格子系の各格子点における x, y の値は定まり，下記の手続きにより微分幾何諸定数を算出することができる．

いま格子定数を $\bar{\xi},\bar{\eta}$ として，

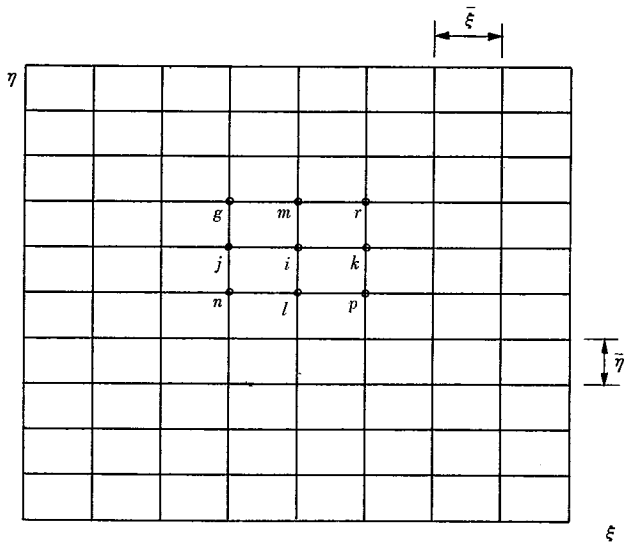


図 13 直交直線格子系

Fig. 13 Orthogonal linear straight lattice system

$$x_\xi=(-x_j+x_k)/2\bar{\xi},\quad y_\xi=(-y_j+y_k)/2\bar{\xi},\quad x_\eta=(-x_l+x_m)/2\bar{\eta},\quad y_\eta=(-y_l+y_m)/2\bar{\eta},$$
$$x_{\xi\xi}=(-2x_i+x_j+x_k)/\bar{\xi}^2,\quad y_{\xi\xi}=(-2y_i+y_j+y_k)/\bar{\xi}^2,\quad x_{\xi\eta}=(x_n-x_p-x_q+x_r)/4\bar{\xi}\bar{\eta},$$
$$y_{\xi\eta}=(y_n-y_p-y_q+y_r)/4\bar{\xi}\bar{\eta},\quad x_{\eta\eta}=(-2x_i+x_l+x_m)/\bar{\eta}^2,\quad y_{\eta\eta}=(-2y_i+y_l+y_m)/\bar{\eta}^2,$$
$$g_1=\sqrt{x_\xi^2+y_\xi^2},\qquad g_2\sqrt{x_\eta^2+y_\eta^2},$$
$$g_{1\xi}=(x_\xi x_{\xi\xi}+y_\xi y_{\xi\xi})/g_1,\qquad g_{1\eta}=(x_\xi x_{\xi\eta}+y_\xi y_{\xi\eta})/g_1,\qquad g_{2\xi}=(x_\eta x_{\xi\eta}+y_\eta y_{\xi\eta})/g_2,$$
$$g_{2\eta}=(x_\eta x_{\eta\eta}+y_\eta y_{\eta\eta})/g_2,\quad \rho_1=-g_1g_2/g_{1\eta},\quad \rho_2=g_1g_2/g_{2\xi},\cdots\cdots$$

のように定められ，与えられた微分方程式，境界条件式もこの格子系により差分表示される．

このように，任意の領域について座標変換差分法を行うためには，直交曲線座標系 ξ,η の確立と，解析面格子系による差分化が必要である．したがって，単発の計算に対しては計算手続きが複雑となり不利と思われるが，同じ形の連続体について荷重条件，境界条件などが種々に変った組み合せ計算のときは有利と思われる．

10. おわりに

本稿で述べたことは筆者が日本ユニバック(株)に入社以来研究した結果をまとめたものである．筆者は三菱重工在社中より専ら有限要素法の研究を行っていたが，ユーザからの問題提起により差分法の利用について考えるようになり，日本ユニバック(株)応用ソフトウェア部マネージャ渡部義維と共に任意節点配置差分法について研究開発を行ってきた．この結果はまとめて本技報[1],[2]にすでに報告した．本稿で報告する座標変換差分法の発想は，むしろ上述のユーザの示唆によるものである．現在，連続体解析の方法論として有限要素法，境界要素法，任意節点配置差分法，さらに今回の座標変換差分法などがあるが，その優劣を早急に論じることは困難であり，今後の研究に待つこととしたい．

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, 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, 1983, pp. 69-83.

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・振動流体力学などの解析法（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) などの著書がある．

分散処理システムのモデリングと分析

松井　節男 ／ 高橋　肇

1. はじめに

分散処理システムの全ライフサイクルを通じて，ハードウェア，データベース，アプリケーション，ワークロードの導入および配置がシステムの性能に及ぼす影響を評価するツールが必要である．そして，性能を予測・分析することによって，システムの設計・構築，アプリケーション・システムのテイラーリング（適合化），データの分散，機器の導入計画，アプリケーション・システムのレベルアップを定量的な尺度にもとづき実施できる．

最近は，分散処理システムに適用可能な性能モデリング・ツールが発表されている．本稿では，積形解法モデルと，使いやすいエンド・ユーザ・インタフェースを採用した AQUADS (Analytic Queue Analysis of Distributed Computer System) システム[1]を中心に紹介する．なお，AQUADS は，Sperry Research Center の R. Heinselman によって開発されたシステムである．

2. 分散システムのライフサイクル

分散システムのライフサイクルは，次の七つのステップに分けられる．一般に性能評価分析システムは，次の 1) および 5) 以外のステップで有用である．

1) 情報分析……自動化対象のアプリケーションの要求仕様を作成する．
2) 実現性調査……考えられる各種の案を検討し，一つの案に絞り込む．
3) システム分析と設計……組織のニーズに適合するよう EDP システムを設計する．
4) プログラム開発……各種の規則およびプログラムを作成する．
5) 購入……必要なハードウェアおよびソフトウェアを購入する．
6) 実現……ハードウェアおよびソフトウェアを設置し稼動させる．
7) 運用と保守……システムを運用し継続的に保守を実施する．

3. コンピュータ性能評価用の一般モデル

性能評価モデルは，設計案の選択，ワークロードの変更，機器の変更などの判断における定量的尺度として用いられる．そして，これらの尺度としては，

1) トランザクションのターン・アラウンド・タイム
2) 機器のスループット
3) キュー形成の速度
4) キューの最大サイズ
5) キューに滞留する平均時間
6) 機器の利用率

がある．

性能評価モデルは，個々の設計案の評価，感度分析，タフなシステム設計案の選択に用いられる．まず高水準モデルによって，ボトルネックを予想したり性能パラメタを予測し候補モデルを絞る．そして，つぎに性能予測を詳細化し感度分析を行い最適化し，コミュニケーションと並行処理プロトコルの正当性を証明するための詳細なモデル群を構築する．

また，与えられた制約を満たす最適設計案を求めるためには，選択および最適化モデルが用いられる．

一般に性能モデルは，一つの設計案を詳細に評価するために利用され，選択および最適化モデルは，多数の設計案の中から最適なものを選択するために利用される．そこで，後者の表現は前者と比較すると，一般的によりあらいものが使われる．

多くの最適化モデルは，発見的アルゴリズムを枠組として用い，性能評価モデルを繰り返し使って評価コスト情報を出力する方式をとっている．最適化モデルで使用される典型的な性能評価モデルは解析モデル[6],[10]であるが，時には離散系シミュレーション・モデルが使われることもある．また，最近は従来の発見的技法に代わって非発見的技法も使われるようになった．

また，検証モデルは，データベースの整合性，節点間のコミュニケーション，回復手順などのプロトコルの正当性を検証するために使用される．これらのすべての検証を一つのモデルで行えるわけではなく，個々の正当性の型に合せたモデルを個別に作成する必要がある．正当性モデルでは，分散型データベースの並行性制御の正当性証明用の離散系シミュレーション技術[9]，正確なプロトコル記述やモデルの操作についての表明を証明するためのペトリネットや，評価ネットの利用などが行われている．

正当性モデルで性能と最適化の両方の問題を考慮することも可能だが[9]，一般にそのようなことは行われていない．

4. コンピュータ性能の解析モデル

モデルの表現の詳細さと実行速度は，トレードオフの関係にある．離散系シミュレーションによるモデルで，システム設計用の比較的正確なデータが得られる．シミュレーション・モデルとしては，GPSS や SIMSCRIPT のようなシミュレーション言語や，RESQ, PAWS/S[3] のようなシミュレーションに準拠した性能評価モデリング・パッケージが用いられている．しかし，シミュレーション・モデルは，マンパワーおよび機械資源を多く使う傾向がある．

シミュレーション・モデルの代わりとして用いられるのが，より安価な数学的（解析的）待ち行列モデルである．しかし，分散処理システムを正確に記述する解析的待ち行列モデルは手に負えないので，分散システムの比較的正確なモデルを解析する近似的解析技法，あるいは近似モデルを解析する正確な解析技法のどちらかが用いられる．分散システム用の大部分の解析的待ち行列モデルは後者のグループに属している．

最近の10年間は解析的待ち行列ネットワーク・モデルの利用が急速に進み，このモデルの発展の歴史や理論を説明するよい資料が多数発表されている．

一般的に待ち行列ネットワーク・モデルはハードウェア，ソフトウェア，ワークロードの変更の影響を調べるための性能評価モデルとして用いられている．なお，これらのモデルでは，スケジューリング方式として，CPU はラウンド・ロビン，入出力機器は FCFS (First-Come-First-Served, 先着順) を採用している．

また，比較的一般的なサービス時間分布関数も用いられており，モデルによっては優先度付き待ち行列や主記憶コンテンションを指定できるものもある．これらのモデルで共通に用いられている入力は，ワークロードの型の数，一つのワークロード中のジョブの数，ワークロードの到着割合，ワークロード・ジョブの一つのハードウェア機器を訪れる頻度，一つのハードウェア機器が一つのワークロードに対してサービスする割合などである．

出力としては，通常，機器の使用状況，ワークロードに対する応答時間，ワークロードの待ち行列，スループットなどである．たいてい入力で使用されるのは平均値であり，出力としても平均値が用いられる．このようなモデルは，理論的に厳密に見ると特定のケースしか成立しないが，経験によると多くの場合，非常によい性能予測を与える（たとえば，通常ハードウェア機器の利用率とスループットの場合で，5パーセント以下，応答時間で25パーセント以下の誤差である[11].）

現在利用可能な解析的モデリング・ツールは数少ない．SNAP[4], PAWS/A[3], MVAP のような汎用パッケージは，中央側のコンピュータ・システム，分散処理システム，コミュニケーション・ネットワークにおける各種の待ち行列ネットワーク問題を解決できる．これらのパッケージの適用範囲は広いが，性能評価に専用化されたユーザ・インタフェースを欠いている．そこで，性能評価専用ツールが登場してきた．これらのソフトウェアの例として，すべての商用の待ち行列モデルの内でおそらく一番有名な BEST/1[5], コミュニケーション・ネットワーク・システムモデルNAP[2], コミュニケーション・ネットワークモデル VANS, 分散システム・モデル AQUADS などがあげられる．

分散処理システムは，コミュニケーション・ネットワークで接続された中央側コンピュータ・システムの集まりとみなせる．そして，原理的にはコミュニケーション・ネットワークをモデル化し，かつ専用の性能モデルを使って個々の中央側のコンピュータをモデル化し，その結果を結合することも可能である．

しかし，このアプローチは，アナリストが異なる数種のツールを知っている必要があり，しかも現在の諸ツールは，ツール間の相互利用を考えて作成されていないためデータの変換が求められる．SNAP や MVAP のようなパッケージは理論的には，問題を分割せずそのまま扱えるが，実際には計算の複雑さが手に負えないことと，分散処理システムのユーザ・インタフェースがないという問題が残されている．

NAP は問題をほぼそのまま処理できるが，メッセージに準拠しており，データ処理と中央側コンピュータ・システムの面ではあまり考慮されていない．

これから説明する AQUADS は，データ処理と中央側システムについてもよく考慮しており，かつ分散処理システム専用のユーザ・インタフェースも有している．しかし，コミュニケーションについては詳細には取り扱っていない．このため AQUADS は，NAP のようなコミュニケーション・モデルのバックアップが必要である．

5. 分散処理システムの評価モデル

AQUADS は，分散処理システムの効率を解析的にモデル化して予測する．このモデルでは，分散処理システムをコミュニケーション・サブネットワー

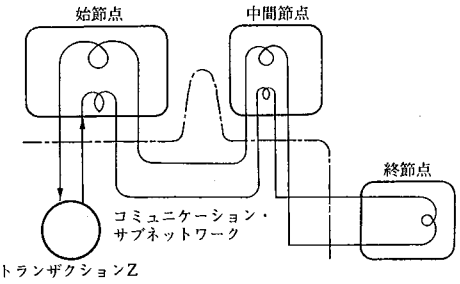


図 1 分散処理システムの概念モデル
Fig. 1 Conceptual model of a distributed computer system

クによって連結されたコンピュータ・システムの集合体とみなしている．使用者は，AQUADS からの一連の応答要求に応えることでネットワークを記述する．この記述により，積型解法で解析される待ち行列のマルチクラス・ネットワークが構築される．解析され計算される効率評価パラメタには，トランザクションの平均ターン・アラウンド・タイムやハードウェア構成要素の使用率などが含まれる．

AQUADS において，モデルを構成する論理的基盤を与えるもの，言い換えれば，分散処理システム内の各種の動きのモデル化された見方を図1に示す．

トランザクションは，モデル化されたワークロードの最小単位であり，実際，それにもとづいてネットワーク効率が計算され記述される．バッチ形式でのラン・ユニットや対話形式でのコマンドおよびデータベースへの問い合せや更新などが，このトランザクションの例である．データは，コンピュータ・システムに当たる各節点で，データベース・セグメントと名づけられた領域に収納される．トランザクションは始節点からネットワークに投入され，終節点でデータベース・セグメントをアクセスする．トランザクションの流れは，始節点から終節点への流れとその逆の流れとから成る．トランザクションが，始節点から終節点まで移動する間に通過する節点を中間節点と呼ぶ．なお，一つの節点が，トランザクションの始節点と終節点を兼ねることもある．

AQUADS はトランザクションに対して，いくつかの仮定を立てている．それは，すべてのトランザクションは同一の優先順位を持ち，各トランザクションは他のトランザクションとは互いに独立していること，一つのトランザクションは同時に複数の資源 (CPU や入出力装置など) を占有することはできないが，異なるトランザクションであれば，異なる資源を並行して占有できるということなどである．

実際のコンピュータ上では，使用者が与えたネットワーク記述は，上述の概念モデルを表す図2のような待ち行列のネットワークを構成するために使わ

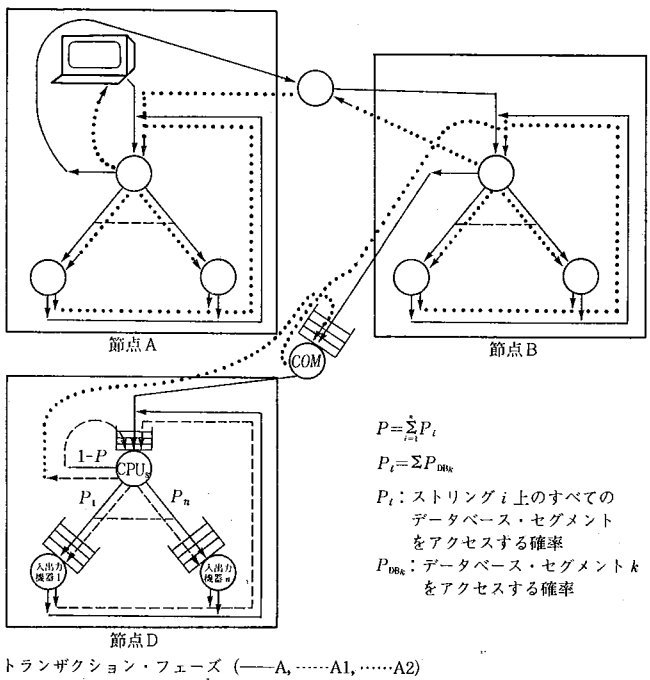


図 2 待ち行列ネットワークの概念モデル
Fig. 2 Conceptual model of queing network

れ，待ち行列ネットワーク分析技法によって解析される．

各節点は開放中央サーバ・モデルによってモデル化され，これらの節点モデルは，コミュニケーション・サブネットワーク内で，リンクにより連結される．このため，トランザクションはマルチクラス開放チェインによって表現される．節点に到着する一つ一つのトランザクションに対し，一つのチェインが存在する．待ち行列ネットワークの効率評価には，マルチクラス積型解析技法を用いている．各節点がプロセッサ複合体のときは，各 CPU をプロセッサ共用 (PS) サーバとみなして構成する．密結合の多重プロセッサで構成される CPU の場合は，マルチサーバ待ち行列理論にもとづく近似解法によって，PS の基準統計値を調整している．また各節点の入出力複合体は，各入出力ストリングに対する FCFS サーバを用い構築されている．このほか，エンド・ユーザ機器は遅延サーバにより表現される．またネットワーク・レベルでは，コミュニケーション・サブネットワークにおけるリンクは，FCFS サーバによって表される．

参考文献

[1] R. C. Heinselman, "Performance Modeling and Analysis of Distributed Computer Systems," Sperry Research Center, Sudbury, MA.

[2] "Network Analysis Program (NAP): Technical and Programming Documentation", Sperry Univac, C & DS, Salt Lake City, Nov. 1981.

[3] "PAWS: Performance Analysis Workbench System", IRA, Austin, Texas.

[4] "SNAP3R0: SNAPL/1 Language Reference Manual", ITR, University of Stellenbosch, South Africa, 1982.

[5] G. Bucci, D. N. Streeter, "A User-Oriented Approach to the Design of Distributed Information Systems", *Measuring, Modelling and Evaluating Computer Systems*, North-Holland, Amsterdam, 1977.

[6] J. P. Buzen, R. P. Goldberg, A. M. Langer, E. Lentz, H. S. Schwenk, D. A. Sheetz, A. Shum, "BEST/1—Design of a Tool for Computer System Capacity Planning", *National Computer Conference* 1978, pp. 447-455.

[7] K. M. Chandy, C. H. Sauer, "Approximate Solution of Queueing Models", *Computer*, Apr. 1980, pp. 25-32.

[8] R. Heinselman, "A Network of Queues Model for Distributed Systems", Technical Report SRC-RP-81-6, Sperry Research Center, Sudbury, MA, Mar. 1981.

[9] K. Lin, "Concurrency Control in a Multiple Copy Distributed Database System", *Berkeley Conference on Distributed Data Management and Computer Networks*, Aug. 1979.

[10] G. M. Schneider, "The VANS System", *COMPCON '78—Fall*, Sept. 1978, pp. 166-174.

[11] K. C. Sevcik, "Data Base System Performance Prediction Using An Analytic Model", *Seventh International Conference on Very Large Data Bases*, Sept. 1981, pp. 182-198.

(英文名：Performance modeling and analysis of distributed processing system)

(ソフトウェア・プロダクト統括部　ソフトウェア一部／技術企画部　テクニカル・パブリケーション室)

特定のコンパイラに対するテストを仕立てる一体験談

H. K. Seyfer

本稿は，Sperry 社の UCS PASCAL のテストについての報告である．ここでは，「テスト」という言葉をテスト・プログラムまたはもっと広いテスト・システムという意味で用いている．テストの収集，開発および UNIVAC テスト制御システム (TCS) のもとでのテスト実施を通して得られた経験について報告している．付録には，TCS のもとでのテスト環境の設定，テストの実行，報告書の自動作成の例があげられている．

以下，章立てにそって本稿の内容を要約紹介する．UCS については文献[10]，TCS については文献[18]を参照されたい（訳者）．

1. はじめに

コンパイラをテストしつくすことは，システムの複雑さという点と膨大な資源を必要とする点から不可能である．テストに費やすことのできる資源の中で，どうして最大の効果をあげるかが問題となる．ここでは，次の順に作業を進めた．

1) 既存テストの収集……テストを構築する手間が省けただけでなく，ソフトウェアのテストをどうするかについての異なった考え方を取り入れることができた．

2) 他言語のテストの変換……シリーズ 1100 独自の機能をテストするために，FORTRAN テ

スト・プログラムからの変換を行った．数学的関数のテスト[13]はFORTRANで書かれているが，これも変換しテストした．

3) テストの開発……上記の収集と変換でテストできない部分についてテストの開発を行った．

2. テスト

2.1 テストの収集

文献[17]の調査をし，次の二つのテストを入手した．

1) PASCAL Validation Test Suite[7]……このテストはオーストラリアのTasmania大学と英国TeddingtonのNational Laboratoryで開発され[21]，PASCALのISO規格案に準拠した318のテストが含まれている．このテストは何をテストするかにより，次の六つのクラスに分類されており，各テストの説明の一部として，ISO規格のどこの章節に関連するかが明示されている．
   - 規格に適当しているか
   - 規格からの逸脱
   - 作成者が定義する部分
   - エラー処理
   - 品質
   - 拡張

2) PASCAL Syntax Error Test……G. D. Ripley と F. C. Druseikis[14] は Arizona 大学の計算機学科の学生の書いたPASCALプログラムの誤りを分析し，共通した誤りのパターン化とエラー・リカバリ・アルゴリズムの効率評価を行った．これらの誤りは分類され，PASCAL Syntan Error Testとしてまとめられた．テストと共に提供されるデータには，誤りの個所と誤りのタイプおよびJensen と W wirth の文献の付録E[11]に掲載されているメッセージ番号が含まれている．3000に近い誤りを127のテストに割り振り，誤りの発生頻度[15]に従って重みづけされている．

3) 無作為抽出……上記以外に，300以上のプログラム（テストのために作られたのではない）を集めた．

2.2 変換したテスト

1) サービス・サブルーチン……日付や時間のような情報を提供するためのサービス・サブルーチンなどのテストは，FORTRANテストから変換された．

2) ELEFUNTルーチン……W. J. Cody, Jr. によってArgonne National Laboratoryで開発された，基本的な数学的関数のためのFORTRANテスト・パッケージELEFUNT[1],[13]をPASCALに変換した．

2.3 開発したテスト

2.3.1 収集したテストの弱点の補填

収集したテストの調査で，明らかになったテスト不十分な点についてテストを開発した．外部手続きやファイルのテスト，UCS PASCALによって翻訳時と実行時に表示される，すべての診断メッセージのテストなどが主なものである．実行時の誤りに関するテストの開発ガイドラインは，Eggertの文献[4]とFischerらの文献[6]に紹介されている．

2.3.2 言語の弱点の補填

PASCALについては議論しつくされている[12]が，とくにWelshらの文献[19]の「PASCALの曖昧性と無防備性」には，型の定義，名前の有効範囲，集合型の構成，可変部分を持つレコード，引数として受け渡される関数や手続き，変数の値が許される範囲を超えること，などをシステムが検査することの有効性が証明されている．これらの言語の弱点の検査に関するテストを作成した．

3. テストの実行

テストは，TCSのもとで実行された．TCSはテストプログラム，実行に必要なJCL，テスト結果を検証するための正解などを管理している．実行すべきテストをインタラクティブに選択し，テスト環境の選択を行うと，テスト・プログラムのコンパイルと実行，結果の検証およびレポートの作成が自動的に行われる．付録にテストの実行の詳細をあげる．

4. 得られた経験と考察

4.1 言語の隅をつつくテスト

言語の隅をつつくようなテストには限界がある[8],[9],[20]．この点について，二つのコメントがある．第1にテスト開発者の気がつかない“隅”がある点，第2は利用者の業種によってプログラミング言語の使われ方が異なる点である．新しいニーズが発生すると，プログラマはテスト開発者，あるいはコンパイラの作成者ですら予測だにしなかった言語の機能を掘り起こすことになる．ユーザからのエラー・レポートが避けられないゆえんである．この2点を同時に解決するには，実際のユーザ環境から種種のテスト・プログラムを選び出すことである．

4.2 テストの公開

テスト・セットをコンパイラ開発者に公開することについて，Wichmannらの文献[20]では，あらかじめ公開すべきではないと結論しながらも，「少数の例を抜き出して公開することは，コンパイラ作成

者に有効なツールを与えることになる」と助言している．テストの最初の段階で，公開テスト・セットの一つ PASCAL Validation Siute のみに頼って90パーセントの合格率であったものが，他のテストも可能となると45パーセントの合格率となったのは，特定のテスト・セットに焦点を合せすぎた証拠である．

4.3 テストのサイズ

Goodenough や Wichmam らの文献[8],[19]と同様，テストの目的を果たすためには，少数の大きなプログラムよりも，数多くの短いプログラムの方がよいことが確かめられた．短いプログラムの欠点——機能どうしの予期しない相互作用，コンパイラの容量などがテストされない点——は，1〜5 パーセントの特定のテスト・プログラムを長くすることによって解決される．Grune の文献[9]にはテストに関する興味ある事実として，「このテストによって発見された誤りの少なくとも 1/4 は，あらかじめ予期しなかったものである」と述べているが，大きいプログラムの方が，テストしようと思っている以外の誤りも発見できる可能性は大きい．

4.4 最適化

数学的なものに関するテストは，コンパイル時に最適化をするか否かで実行結果が異なる．これらについてはテストを別に管理し，必要なだけ正解を準備した．

4.5 内部的検証と外部的検証

内部的検証とは，テスト・プログラム中で，たとえば

IF 〉実行結果〉=〈期待する結果〉

などという形で検証することである．外部的検証とは，結果をファイルに出力し，最初の1回だけ人手により検証し，2回目以降はこのファイルとの照合によって検証する方法である．外部的検証は，誤ってテストを合格にしてしまう問題を最小化できる．コンパイラが言語仕様に合致しているかといったテストは，内部的検証が適当である．

4.6 適合性と効率

標準規格に適合するか否かのテストと，品質のテストを混同してはならない．品質という面で，まず目に見える指標は効率と数値の精度である．効率は時間とスペースで計られる．精度は数量的に把握できるが，標準規格には定められていない．Scowen の文献[16]は精度を取り扱う上での規格の曖昧さに言及している．Cugini の文献[3]は数値の精度について詳しい．ELEFUNT[2]は精度のテストのすぐれた例である．

4.7 テストにおける基本的な前提

テストをするために，必要な機能について前提条件を設けるのが普通である[2],[5]．通常，次のいずれかである．

1) 後続のテストに必要な機能を先にテストできるように，機能の階層化をする．
2) 基本的な前提を確認するテストを作成し，他のテストより前に実行する．
3) 基本的前提を無視する．

第2の方法が妥当な折衷案で，検証を手早く行うテストに向いている．

5. おわりに

テストの収集をした，S. Costello，テストを実施した M. Feuer, T. Kridle に感謝する．K. Sogge からのコメント，P. Klausler のアシスタントに感謝する．

〔付録〕 テストの実行例

ここでは，図1にあげたプログラムを用いたテストを例にする．

図2は，デマンド・ユーザとテスト・コントローラとのやりとりを示す．

1行目と2行目でユーザは，上記のテストを含むテスト・パッケージ PCRT を指定して，コントローラを呼び出す．

4，5行目でデマンド・モードでテストを実行することを宣言する．いくつかのテストを実行するつもりであれば，テストをバッチ・モードで実行するよう指定することもできる．

6行目から29行目までは，コントローラが，ユーザへオプションやシステム機能について問い合せている．テスト・パッケージ・ファイルの中に，問い合せと省略時の標準値がはいったエレメントがある．

31行目にこのテスト・パッケージは，まだ開発中である旨の警告が表示されている．

32行目と33行目で実行すべきテストを指定する．34行目は，テストの選択がうまくいったという確認メッセージである．コントローラは，テスト環境の条件が合致せず，指定されたテストが実行できない旨のメッセージを返すことがある．たとえば，テストが特定の機械の上でのみ実行すべく指定されている場合などである．

35行目と36行目では，標準のファイルを必要としないときや，標準のファイルが何であるかを知りたいときに“no”と答える．コントローラは標準のファイルを表示し，どれを変更すべきかを問い合せてくる．

37行目で，コントローラはユーザにテスト・レポートを受け取るための参照番号を割り当てる．このレポートの例は後で示される．

38行目から44行目までは，テストの実行の進行状況の表示である．38，39行目でテストの開始が示される．40，41行目でパッケージ PCRT のためのステータス・ファイルが，このテストの実行を登録したことで更新されたことが示される．この更新はテストの開始時刻の記録のためで

ある．この情報は，テストされるソフトウェアがループしたり，ハングアップした場合の検出に役立つ．

43行目と44行目はテストの実行が終了し，ベース・エレメントとの比較が始まったことを示している．比較の結果は45と46行目に示される．この例ではコンパイル中に差異があり，実行結果には差異がなかったことを示している．

47行目から51行目で，コントローラは個々のテスト報告のどの情報が必要かを問い合せている．47行目は報告書が必要か否かの問い合せで，ここで“no”と答えればもちろん報告書は出力されない．ここで示すように“yes”と答えると，図3の1から46行目のようなレポートが作成され，報告に追加される．図2の49行目で“yes”と答えると，さらにコンパイルに必要な情報を問い合せてくる．

図2の52行目に答えると，図3の47行目から48行目がレポートに追加される．このレポートのハード・コピーがとられ，テスト・セッションの詳細を検討するのに使われる．

図3は，前にも述べたように，単一テストのテスト結果である．1行目から20行目までが，今回のテストと前回のテストの実行結果の比較である．この例では前回のテストでは，type宣言の矛盾について診断メッセージが出力されなかったことを示している．13行目から16行目の〈T〉は，この行が前回の実行にはなく今回の実行ではじめて現れたことを示す．11，12行目はコンパイラが呼び出されたときに表示される行である．日付と時刻といった情報は常に異なるので，コントローラにはこれらの行の相違を無視するように指示する．このような指定があったことをユーザに示すために，実行されたコントローラ指令文が5行目に印刷される．

7行目と17行目には，引用符とブラケット内の数字で分割された二つの比較が示されている．ファイルを何と比較すべきかが，UPASプロセッサ（UCS PASCALのコンパイラ）の処理部分については3行目に，またXQTに関する部分については4行目に指定されている．

33行目のプロセッサDREDの呼び出しで，図2の6行目から30行目でユーザが選択した機能が挿入される．最少限のコンパイル・リストが要求されているので，図3の35行目から38行目までに診断メッセージのみが生成される．（例題を短くするために，コンパイラの開始と終了を示す印書行は省略してある）．

図2の51行目と53行目でコンパイルの詳細リストを指定しているので，コンパイルのLリストがレポートの46行目以降に出力される．

図4はパッケージ全体のステータス・レポートの例である．このレポートのほとんどの部分は，読んで解るような形で出力されている．ここで述べてきた例の結果が，19行目から21行目に表示されている．このテストは，1982年6月10日10：25：00に実行された．テスト・ベース・エレメントはUPASレベル 0R1T1で作成され，テストはレベル 0R1T2で実施された．テストには 000：22 SUP (Standard Unit of Processing)，すなわちCPU時間と入出力に約22秒を要した．

```
1.  | The language imposes an ordering on the different classes |
2.  | of declaration within a block.  However, there is nothing |
3.  | forbidding declaration after use within  the type         |
4.  | definition.  This should be no problem to a two-pass      |
5.  | compiler.  A one-pass compiler should diagnose correctly  |
6.  | Refer to Welsh, Sneeringer, and Hoare, p688.              |
7.  PROGRAM main (output) ;
8.    type
9.      matrix = array [1..10, 1..10] of complex ;
10.     complex = record
11.                 realpart, imagpart: real
12.               end ;
13.   var
14.     M      : matrix ;
15.     ii,rr : integer ;
16. begin ;
17.   writeln ('For a single-pass compiler, an error should have
18.             occurred in the TYPE section during compilation.') ;
19. end
```

図1 原始プログラム例
Fig. 1 Sample tPst source

参考文献

[1] W.J. Cody, W. Waite, *Software Manual-for The Elementary Functions*, Prentice-Hall, 1980.

[2] J.V. Cugini, J.S. Bowden, M.W. Skall, *NBS Minimal Basic Test Programs—Version 2, User's Manual, Volume 1—Documentation*, NBS Special Publication 500-70/1, November 1980, pp. 15-16.

[3] J.V. Cugini, *Specifications and Test Methods for unmeric Accuracy in Progra-*

```
1.  << user types "@tcs$.controller pcrt"
2.         pcrt is name of test package >>
3.  FTC 3R1.85   06/10/82 15:37:08  CREATED  01/22/82 09:27:20
4.  MODE?
5.  << user types "demand" (for this example) >>
6.          SPECIAL EDITING CONSIDERATIONS FOR  PCRT
7.          PASCAL COMPILER CALL OPTIONS MAY BE SPECIFIED.
8.          COMMA MUST PRECEDE OPTION(S); E.G., ",S"
9.          ' '  ( BLANK IN SINGLE QUOTES ) ==> NO OPTIONS.
10.         ENTER PAS OPTION(S):
11. ----- DEFAULT  IS ' '    ---->
12. << user presses XMIT key for default >>
13.         MAP OPTION MAY BE SPECIFIED.
14.         ' '  ( BLANK IN SINGLE QUOTES ) ==> NO OPTIONS.
15.         ENTER MAP OPTION(S):
16. ----- DEFAULT  IS S      ---->
17. << user presses XMIT key for default >>
18.         TYPE OF LIBRARY MAPPED MUST BE SPECIFIED:
19.              FOR                                  | ENTER
20.            -------------------------------------|------
21.            NON-REENTRANT LIBRARY                |  NR
22.            CONFIGURED COMMON BANK LIBRARY       |  CC
23.            NON-CONFIGURED COMMON BANK LIBRARY   |  NC
24.         ENTER TYPE OF LIBRARY MAPPED
25. ----- DEFAULT  IS NC     ---->
26. << user presses XMIT key for default >>
27.         OPTION KEYWORDS MAY BE SELECTED.
28.         ENTER OPTION KEYWORDS:
29. ----- DEFAULT  IS NOOPTIONS·  ---->
30. << user presses XMIT key for default >>
31. PCRT: DEVELOPMENT
32. TEST NAME?
33. << user types "ai-scop-1" >>
34. AI-SCOP-1 WILL BE RUN.
35. DEFAULT FILES OK?
36. << user types "yes" >>
37. YOUR DEMAND USER NUMBER IS: 05
38. @MSG,N COMPILE, MAP, AND XQT OF AI-SCOP-1 FOLLOW (IN BRKPT)
39. @BRKPT PRINT$/TESTRESULTS
40. @ESTAT:SYSTEM$.STATUS,L  PCRT
41. FTS 3R1.86   06/10/82 15:55:30  CREATED  02/03/82 14:11:47
42.  ***  AI-SCOP-1  FROM  PCRT  ***
43. SDFCOMP 3R1.34   06/10/82 15:55:31  CREATED  02/03/82 14:11:
44. **** END SDFCOMP ****
45. THERE WERE DIFFERENCES IN COMPILATION.
46. THERE WERE NO DIFFERENCES IN EXECUTION.
47. KEEP LISTINGS?
48. << User types "yes" >>
49. COMPILATION LISTINGS?
50. << User types "yes" >>
51. DEFAULT OPTIONS ARE L.  WHAT OPTIONS?
52. << User presses XMIT key for default >>
```

図 2 デマンド・ユーザとテスト・コントローラのやりとり
Fig. 2 Sample of test execution in demand

mming Language Standards NBS Special Publication 500-77, June 1981.

[4] P.R. Eggert, "Runtime Checking for ISO Standard Pascal," *IEEE Transactions on Software Engineering*, Vol. 7, No. 4, July 1981, pp. 447-448.

[5] *FORTRAN Compier Validation Systdm (FCVS 78), 1.0 Detailed Test Specifications*, Federal Compiler Testing Version Center, November 1978, available from National Technical Information Service as publication AD-A 062-038.

[6] C.N. Fischer, R.J. LeBlanc, "The Implementation of Run-Time Diagnostics in Pas-

```
1.  SDFCOMP 3R1.34   06/10/82 11:20:58
2.  SDFCOMP   RESULTS.AI-SCOP-1/BASE,TESTRESULTS.,PTCFILE
3.   COMPILER UPAS
4.   EXECUTION XQT
5.   IGNORE UPAS 1,9 BEGIN UCS
6.  **** SDF COMPARE INITIATED **** - BY SDFCOMP -
7.  """""""""""""""""""""""""""""""""""""""""""""""""< 1 >"""""""""""""""""""""""""
8.                              SDF COMPARE FOR PROCESSOR UPAS
9.           @PAS$.UPAS          SOURCE$.AI-SCOP-1,RB$.REL,,,NOOPTIONS
10.          @PAS$.UPAS          SOURCE$.AI-SCOP-1,RB$.REL,,,NOOPTIONS
11. <BI      BEGIN UCS PASCAL OR1T1 02/10/82 15:52:49
12. <TI      BEGIN UCS PASCAL OR1T2 06/11/82 11:20:47
13. <T>      **ERROR(MAJOR) 10  Error in type     v
14. <T>      **REMARK(CLARIFICATION) 10  Scanning resumes here
15.           after last error with this number          v
16. <T>      3  matrix = array [1..10, 1..10] of complex ;
17. """""""""""""""""""""""""""""""""""""""""""""""""< 2 >"""""""""""""""""""""""""
18.                             SDF COMPARE FOR PROCESSOR XQT
19.          @XQT   RB$.ABS
20.          @XQT   RB$.ABS
21. @HDG ****  TEST OUTPUT FOR PCRT TEST   ****  AI-SCOP-1
22. FTS 3R1.86   06/11/82 11:20:42  CREATED  02/03/82 14:11:47
23. @ADD,LP FESTPAS*PCRTJCL.AI-SCOP-1/JCL
24. :DOCUMENTATION
25.  << documentation appearing in the test's JCL >>
26. @ASG,T   RB$.,F/10//500
27. READY
28. @ERS    RB$.
29. FURPUR 28R2T2 S74T11 06/11/82 11:20:45
30. END ERS.
31. @SYSTEM$.DRED
32. DRED 3R1.47   06/11/82 11:20:46  CREATED  07/16/81 09:01:51
33. @ADD,LP    DRED$.
34. @PAS$.UPAS          SOURCE$.AI-SCOP-1,RB$.REL,,,NOOPTIONS
35. **ERROR(MAJOR) 10  Error in type     v
36. **REMARK(CLARIFICATION) 10  Scanning resumes here
37.  after last error with this number          v
38. 3  matrix = array [1..10, 1..10] of complex ;
39. @MAP$.MAP,F    SOURCE$.MAPNC,RB$.ABS
40. END MAP    ERRORS: 0  TIME: 7.081
41. @XQT   RB$.ABS
42.  << The execution of the test is performed as a check >>
43.  << on runtime error recovery and diagnostics.          >>
44. @FREE    RB$.
45. READY
46. @BRKPT PRINT$
47. @HDG ****      LISTING(S) FOR PCRT TEST ****  AI-SCOP-1
48.  << A long listing of the source element appears here >>
```

図 3 単一テスト結果
Fig. 3 Sample individual test report

cal", *IEEE Transactions on Software Engineering*, Vol. 6, No. 4, July 1980, pp. 313-319.

[7] R.A. Freak, A.H.J. Sale, "Pascal Validation Suite— Version 3.0," Department of Information Science, University of Tasmania.

[8] J.B. Goodenough, *ADA Compiler Validation Implementers' Gmide* (Waltham, Mass.: SoftTech, Inc., 1980), available from National Technical Information Service as publication AD-A 091-760, October, 1980.

[9] D. Grune (ed.), *The Revised MC ALGOL 68 Test Set*, IW-122/79-November (Amsterdam, The Netherlands: Mathematisch Centrum 1979).

[10] H.C. Gyllstrom, R.C. Knippel, L.C. Ragland, and K.E. Spackman, "The Universal Compiling System", *SIGPLAN NOTICES*, Vol. 14, No. 12, December 1979,

```
1. TEST CONTROLLER SYSTEM STATUS PRINTED AT 13:40:06 ON  6-10-82
2. *** ON THE FLY STATUS -- DEMAND MODE ***
3. *** USER NUMBER 62 - RUNNING PASCAL CRITIQUES
4.     (PCRT: DEVELOPMENT) ROUTINES ***
5.    1 TEST(S) PASSED.
6.    2 TEST(S) FAILED.
7.   73 TEST(S) WERE NOT RUN.
8.    3 TOTAL TESTS RUN.
9.  *** TESTS THAT HAVE FAILED ***
10. AI-RCRD-EQ18  AI-SCOP-1
11. NO TESTS ACTIVE.
12. # FOLLOWING TESTS WERE RUN ON AN 1100/60 UNDER EXEC 38R2 #
13.  AI-NAME-EQN3 PASSED.                              10:24:15
14.     ON  6-10-82 ***  BASE UPAS  OR1T1 ; TEST UPAS  OR1T2
15.     ELAPSED SUPS = 000:21
16. >AI-RCRD-EQ18 VARIED IN COMPILATION-EXECUTION< 10:24:30
17.     ON  6-10-82 ***  BASE UPAS  OR1T1 ; TEST UPAS  OR1T2
18.     ELAPSED SUPS = 000:25
19. >AI-SCOP-1    VARIED IN COMPILATION               10:25:00
20.     ON  6-10-82 ***  BASE UPAS  OR1T1 ; TEST UPAS  OR1T2
21.     ELAPSED SUPS = 000:22
22. TESTS WERE STARTED BETWEEN 10:24:15 ON  6-10-82
23.      AND 10:25:00 ON  6-10-82.
```

図 4 パッケージ全体のステータス・レポートの例
Fig. 4 Sample test package status report

pp. 64–70.

[11] K. Jensen and N. Wirth, *Pascal User Manual and Report, Second Edition*, New York, Springer-Verlag, 1974.

[12] D. V. Moffat, "Index to the Periodical Literature—1981 Pascal Bibliography (June, 1981)", *SIGPLAN NOTICES*, Vol. 16, No. 11, November 1981, pp. 7–21.

[13] *ELEFUNT (FORTRAN Elementary Function Tests)*, NESC Abstract 881 (Argonne, Illinois : National Energy Software Center, Argonne National Laboratory, 1980.

[14] G. D. Ripley, F. C. Druseikis, "A Staistical Analysis of Syntax Errors", *Computer Languages*, Vol. 3, 1978, pp. 227–240.

[15] G. D. Ripley, "Pascal Syntax Error Data", report accompanying test programs, 1981.

[16] R. S. Scowen, Z. J. Ciechanowicz, "Compiler Validation—A Survey", *NPL CSU Technical Report* No. 8/81, National Physical Laboratory, December 1980.

[17] H. K. Seyfer, "Compiler Test Sets", *SIGPLAN NOTICES*, 1982.

[18] *User Guide, Sperry Univac Series* 1100 *Test Controller System (TCS) Level 2R1*, RRD-A 446. 2, Sperry, 1980.

[19] J. Welsh, W. J. Sneeringer, C. A. R. Hoare, "Ambiguities and Insecurities in Pascal", *Software Practice and Experience*, Vol. 7, 1977 pp. 685–696.

[20] B. A. Wichmann and B. Jones, "Testing Algol 60 Compilers", *Software Practice and Experience*. Vol. 6, No. 2, April-June 1976, pp. 261–270.

[21] B. A. Wichmann, A. H. J. Sale, *A Pascal Processor Validation Suite*, Report CSU 7/80, (Teddington, England : National Physical Laboratory, 1980).

(英文名 : Tailoring Test to a Specific Compiler)

編集部注 : 本稿の著者 H. K. Seyfer は, 現在 Sperry 社でシステム・プログラマとして言語プロセッサおよびその関連ソフトウェアのテスト・システムの開発やパフォーマンス評価に従事.

(ソフトウェア・プロダクト統括部
ソフトウェア一部 藤村 光 訳)

——浦　昭二/土居範久/原田賢一 共訳——

プログラミング原論

——いかにしてプログラムをつくるか——

サイエンス社，A5判，xii＋254 pp.，
1983年，3,500円

「以前は，少しは上手な人が書き，普通の人がそれを読んだ．今は，普通の人が書いて，誰も読んでいない．」と，ある作家が100年程前に嘆いたそうだ．今日，プログラムについても同じようなことが言えそうだ．

かつて，コンピュータ・プログラミングは高度な理工学的技芸であった．この仕事に携わる人は，高い専門性と鋭い洞察力を備えた知的エリートであった．プログラミングという仕事もプログラミングをする人も，社会からは畏敬の目でながめられていた．

ところが，コンピュータがたくさん使われるようになり，人々にとってプログラミングという言葉が身近なものとなるにつれ，変化がおこった．プログラミング経験者がふえる一方，プログラミングという行為は疎ましく感じられるようになり，粗末に扱われるようになった．

作文の大衆化が文章のインフレを招いたように，プログラミングの偏よった普及がプログラムのインフレを引き起こした．

このような現象を苦々しく思っている人々がいる．彼らはコンピュータ・プログラミングの本質に対する根本的で鋭い洞察と，Discipline に裏打ちされた確かな技能によってこそプログラムが意味を持つ，と主張する．本書の著者 Dijkstra も，またこの主張者の1人である．

本書は，Dijkstra の A Discipline of Programming の日本語版である．原著は1976年に発行され，その翻訳が待たれていたが，このたび約6年目にして手にすることができた．

著者自らが言っているように，本書は初学者のために書かれたものではなく，特定のプログラミング言語に関する入門書でもない．プログラミングの本質に対する彼の哲学と，そこから生まれたプログラミングの方法を述べたものである．すなわち，前半においては，正しいプログラムを作るための道具立てとして，プログラム実行の過程で不変に保つべき関係に着目た数学的基礎を与え，後半でいくつもの例題を通してその有効性を示している．

ちなみに，章立てを以下に引用しておく．

0　実行の抽象化
1　プログラミング言語の役割
2　状態およびその特徴づけ
3　意味の特徴づけ
4　プログラミング言語の意味の特徴づけ
5　二つの定理
6　適列に停止する構造の設計について
7　ユークリッドのアルゴリズムの改訂
8　いくつかの小さな問題の形式的処理
9　制約を受ける非決定性について
10　“変数の有効範囲”という概念について
11　配列変数
12　線形探索定理
13　次の順列を求める問題
14　オランダ国旗の問題
15　逐次ファイルの更新
16　併合問題の見直し
17　R. W. Hamming の問題
18　パターン照合の問題
19　二つの平方の和として数を書くこと
20　大きな数の最小素因数の問題
21　最も孤立した村の問題
22　最短の展張木の問題
23　同値クラスを記録するための Rem のアルゴリズム
24　3次元における凸包の問題
25　有向グラフにおける極大強成分を求めること
26　説明書と実現について
27　いままでをふり返って

著者がたどった考えの経過を述べているだけに，難解で読みこなすには努力も必要と思われるが，幸い，こなれた翻訳となっているので，プログラマであることに誇りを持ち，有能なプログラマであろうとする人にはすすめたい．

(Edsger W. Dijkstra, “A Discipline of Programming”, Prentice-Hall, 1976)

Alfred V. Aho, John E. Hopcroft, Jeffrey D. Ullman 著
“Data Structures and Algorithms”

Addison-Wesley, xi＋427 pp., 1983.

本書は，Pascal のような命令型言語によるプロ

グラミングでよく用いられるリスト，スタック，木，グラフなどの基本的データ構造と，それらを操作する算法についてやさしく述べた入門書である．すでに著者らには，算法の設計と解析の方法から計算量論までを包括的に概説した，

（1） A. V. Aho, J. E. Hopcroft, J. D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, 1976.（「アルゴリズムの（「設計と解析Ⅰ，Ⅱ」野崎昭弘，野下浩平訳，サイエンス社）

がある．本書は，序文に述べられているように，前書のはじめの6章にほぼ相当する内容を super Pascal を用いてプログラマ向きに述べている．すなわち，内容的には，CPU とフラットなメモリ・セルの配列から成る，いわゆる von Neumenn 型コンピュータを仮定して，いろいろなデータ構造の実現法とそれらを扱う算法を網羅的に説明している．また，1章では算法の時間量を導入し，前書の第2章に対応して，9章では算法の時間量解析を述べ，10章では算法の設計法を述べている．とくに，新たに外部記憶に関する算法と Garbage Collection について11章と12章が加えられている．したがって，やさしい時間量の解析を加味した一般的なデータ構造とそれに関する標準的算法を述べたプログラマ向きのやさしい入門書（あるいは教科書）となっており，とくにパソコンを含めた“システム”のソフトウェアを製作するプログラマには，手短かな算法集ともなろう．

同型の著書としては，すでに次のようなものが出版されている．

（2） N. Wirth, Algorithms+Data Structures=Programs, Prentice-Hall, 1976.（「アルゴリズム+データ構造=プログラム」片山卓他訳，日本コンピュータ協会）

（3） D. E. Knuth The Art of Computer Programming, Vol. 1, 2, 3, Addison-Wesley.（「1. 基本算法=基礎概念」広瀬　健訳，「2. 基本算法=情報構造」米田信夫・筧　捷彦訳，「3. 準数値算法=乱数」渋谷政昭訳，「4. 準数値算法=算術演算」中川圭介訳，サイエンス社）

（4） T. A. Standish, Data Structure Techniques, Addison-Wesley, 1980.

（5） L. I. Krönsjö, Algorithms; Their Complexity and Efficiency, John-Wiley, 1979.

Pascal-like 言語で書かれている点では，Wirth[2] と同様であるが，プログラミングの過程を述べる点では本書は Wirth[2] より劣っている．しかし，タイトルの示すように網羅する算法のレパートリーでは，本書は Wirth[2] の2章から4章までよりはるかに広いし，時間量の解析の記述もすぐれている．本書の著書らはコンパイラにきわめて詳しいが，Wirth[2] の5章に相当する記述は本書にはない．Standish[4] は Knuth[3] と同様の記法で算法を記述しているが，内容的には本書に最も近い．Tom Standish は元来データ構造で学位を取得している専門家であるが，そのためか，Standish[4] の方が諸算法を網羅するという点で本書より優れており，プログラマ向きの算法の参考書としては（記法を気にしないならば）Standish[4] の方が適当であろう．冒頭に本書をやさしい入門書であると述べたゆえんである．やや古くはなったが，Knuth[3] は算法の解析の点で最良であり，MIX 言語を気にしないならば，算法の書として現在でも最もすぐれている．Knuth[3] 以降については，本書や Standish で補足するのがよいであろう．Krönsjö は数値算法も含めた Knuth[3] のコンパクト版であり，算法とその解析を簡単にレビューするのに適している．各著書の特色に応じて，使い分けるのがよいであろう．

さて，プログラミングでは，言うまでもなく処理効率のよい算法を工夫し設計する必要がある．とくに，反復利用する“ソフトウェア”においては使用する算法の処理効率の問題が重要になることは言をまたない．しかし，一方，与えられた問題に対して，それを解く“よい”算法を発見したり，工夫・改良することは実際にはむずかしい部類の問題であることも真実である．実際，例としてはやや穏当を欠くが，5次以上の一般の代数方程式の根号による解法の存在を否定する Galois の理論や Hilbert の第10問題に対する Matijasevic の結果を想起すれば，算法の発見が人間の高度の知的活動の所産であることを推量できよう．さらに，算法の処理効率の改良についても，たとえば単純な行列の乗算算法についてさえ，V. Strassen の算法が発見されている．S. Winograd の定理によれば，行列の乗算算法の時間量（乗算回数）の一つの下界は行列のサイズの平方のオーダ（位数）であるが，現在でもどこまで平方に近づけうるかは未解決の問題である．

歴史を振り返るならば，人間が工夫し，発見した諸算法は，たとえば，Rhind のパピルス，ユークリッドの原論，あるいは九章算術として集大成されてきた．コンピュータの出現以降，計算が高速化すると共に，とくに時間量の小さい算法の工夫に対する要求は一段と大きい．それゆえ，コンピュータ用

算法の設計と解析は，いわゆる情報科学に固有の最も重要な研究分野の一つであり，その所産は正当に評価され，集成されなければならない．この意味で，これらの著書の存在意義は大きい．

ところで，Knuth[3]からも，うかがえるように，算法の解析には組み合せ数字と同様の有限を扱うむずかしさがある．この方面については，

(6) C. L. Liu, Intoroduction to Combinatorial Mathematics, Mc-Graw Hill.（「組み合せ数字入門Ⅰ，Ⅱ」伊理正夫他訳，共立出版）．

(7) E, S. Page, L. B. Wilson, An Introduction to Computational combinatorics, Cambridge University Perss, 1979.

(8) R. A. Brualdi, Introductory Combinatorics, North-Holland, 1977.

(9) D. H. Greene, D. E. Knuth, Mathematics for the Analysis of Algorithms, Birkhäuser, 1981.

などがある．本書ではこの点の記述が浅いので，Knuth[3]や，これらを同時に参考にされることを薦めたい．

新刊紹介

このたび，「情報システム部門のマネジメント」および「意思決定支援システム」が日本ユニバック・グループの社員らによって発刊された．

萩原忠雄（日本ユニバック情報システム(株)）著

情報システム部門のマネジメント

共立出版刊／A 5 判・260 頁／3500 円

多くの企業の情報システム部（コンピュータを利用したデータ処理システムの開発・運用部門）が，共通の役割として持っている計画・開発・運用・保守・費用管理などの業務をどう管理していくかについて，主に中間管理者の立場から具体的に書かれている．また，本書では受益者負担の原則に従って，情報システム部の費用をすべて利用部に付け替える場合を想定して，どのように業務別に費用を積み上げ，これらをどのような考え方で各利用部へ配賦していくかについて，その一例を説明している．

［目次］ 1. 情報システム部の役割と組織 2. 情報システム部と利用部の関係 3. 情報システム部の管理 4. 仕事と要員の管理 5. システム計画 6. 機械化費用見積 7. 詳細設計 8. システム開発 9. システム運用 10. 費用管理 付録. 各種フォーマットの記述要領

広内哲夫（文教大学情報処理学部）
小坂 武（日本ユニバック株式会社） 共著

意思決定支援システム ——DSS 構築の方法論——

竹内書店新社刊／菊判・232 頁／2800 円

マネジメントの問題解決能力を増大するシステムとして，意思決定支援システム（DSS）が注目されている．柔軟性に豊んだ DSS の構築により，企業内組織を横断する情報ネットワークが促進されるからである．

本書は，この DSS を構築したいと考えているマネジメント，システム・エンジニア，上級プログラマのために実践的立場から執筆された，本格的な DSS の解説書である．まず，DSS の歴史，概念から始まり，筆者の実践経験と研究成果にもとづいた事例およびシステム運用の実際までが述べてある．

［目次］ 1. DSS と MIS 2. 意思決定と DSS 3. DSS アーキテクチャ 4. データベース 5. ユニット・モデル 6. DSS パッケージ 7. システム操作 8. モデルの作成 9. システム設計 10. DSS の実施

●第13回画像工学コンファレンス——1982年12月7日～8日，東京

本年の招待講演は，塚原進氏（福島医大），藤尾孝氏（NHK），宮原諄二氏（富士フィルム），鈴木義二氏（浜松テレビ），内海厚氏（大日本電線），江尻正員氏（日立中研），長谷川三郎氏（西洋美術館），山中正宣氏(阪大)によって，それぞれ「CRTディスプレイの見やすさの研究」，「高品位テレビと将来の画像システム」，「新しいメモリ型蛍光材料を用いたコンピューテッド・ラジオグラフィー」，「画像計測用イメージ・デバイス」，「石英系イメージ・ガイドの現状」，「産業用ロボットと視覚」，「絵画の科学的調査とその可能性」，「コーデド・アパーチャ・イメージングのレーザー核融合研究への応用」と題して行われた．ポスター・セッションは，視覚と画質，表示装置，新しい画像技術，光技術，画像処理，画像計測，符号化が行われた．

●第24回プログラミング・シンポジウム——1983年1月11日～13日，箱根

森政弘教授（東工大）による「ロボットづくりのむずかしさ」の招待講演が行われた．

一般講演の主なテーマは，PASCAL 数値データに関する機械独立コード生成法，設計システム記述言語 ADL，パソコン用言語 SPICA，抽象データ型言語によるストリームの実現，推論関係型 DMBS Adbis の会話型データ操作言語 Tsuno，推論機械 SIMP，形式的仕様記述の道具としての自然言語，数式処理言語 GAL，TAO—Lisp，Prolog，Small talk の調和平均，英文テキスト清書システムの改良，表現された知識として見たソフトウェアなど，でっあた．

●COMPCON Spring '83——1983年2月28日～3月3日，San Francisco

今回の基調講演は，「超 LSI システム入門（Introdurtion to VLSI Systems）」（培風館刊，1981）の著者として知られる L. Conway (Xerox PARC) 氏による「情報化社会における知的挺子」．チュートリアル・セミナーのテーマは，高級 VLSI 実装技術，プログラミング言語 Ada，プログラマブル・アレイ・ロジックの利用，Josephson 接合素子とその応用．このほか，スペシャル・セッションとして，昨年8月に IBM を除く米国の大手のコンピュータ・メーカと半導体メーカ合わせて10社によって設立された，米国コンピュータ業界の協同研究会社 MCC (Microelectronics and Computer Technology Corporation) が紹介された．

また，パネル討論では，コンピュータ倫理，日本の高技術（第5世代コンピュータ，機械翻訳），音声と文書との結合，高技術（High-Tech）ベンチャー・キャピタル，会話型プログラミング環境などが論じられた．

●情報処理学会第26回全国大会（前期）——1983年3月15日～17日，東京

今回の特別講演は，山本七平氏による「日本人の創造性」，招待講演は梅谷陽二氏（東工大）による「ロボット工学の現状と将来」であった．また，パネル討論のテーマは，「論理型プログラミングとオブジェクト・オリエンテッド・プログラミング」および「ソフトウェア・メトリックスの現状と課題」で，前者では Prolog に代表される論理型プログラミングと Smalltalk に代表されるオブジェクト・オリエンテッド・プログラミングの是非，後者ではソフトウェアの信頼性・複雑性などの特性を計量的に把握する技術が討議された．このほか，約760件の一般講演が，言語関係，データベース関係，ネットワークおよびオフィス・システム関連，日本語および機械翻訳関連，人工知能およびエキスパート・システム関係，ハードウェアおよびアーキテクチャ関係，ソフトウェア工学関係，などの会場に分かれて発表のがあった．

●昭和58年度電子通信学会総合全国大会開催——1983年4月2日～4日，仙台

今回の特別講演は，外島 忍氏（東北大）学による「電池のはなし」および，小田 稔氏（宇宙科学研究所）による「X線天文学発達の歴史—すだれごしに見た宇宙」であった．また，パネル討論は，榎本肇氏（東工大）を座長として「画像サービスの将来とそれを支える技術」をテーマに行われた．

そして，最近の話題を取り上げ集中的に複数の論文が発表されるシンポジウム講演では，集積回路におけるデバイス・シミュレーション，移動通信におけるディジタル信号伝送，情報源符号化の新しい展開，Josephson 計算機用ディジタル回路，VLSI チップ上の高速信号伝送，単一モード光ファイバ伝送，記号処理・データベース・高級言語処理用の専用計算機，アモルファス半導体，自然言語処理，複合交換，薄膜磁気記録媒体，文字認識，技術革新時代の大学院像，LAN (Local Area Network) のサービス総合化の諸問題などが取り扱われた．

●NCC '83 (National Computer Conference)——

1983年5月16日～19日, Anaheim (Calif.)

今年は「情報化時代の登場：コンピュータ，通信および人々 (The Emerging Information Age: Computers, Communications, and People)」をテーマに，ADAPSO (Association of Data Processing Service Organizations) の前会長である J.P. Imlay 氏 (MSA, Management Science America)を基調講演者として迎え行われた．また，この大会の初日に，Harry Goode Memorial 賞が，パターン認識研究のリーダーとして知られる K.S. Fu 氏 (Purdue University) に授与された．なお，同賞は情報処理分野における卓越した業績に対して与えられるもので，昨年の受賞者は C.A.R. Hoare 氏 (Oxford University) であった．

このほか，5月18日の恒例のパイオニア・デー(情報処理分野のパイオニアを記念して，その貢献の歴史的意義を討論する日) には，Mark I，II，III，IV を開発した H.H. Aiken 氏と Harvard Computation Laboratotory の業績を取り上げ，二つのセッションが開かれた．

また，今回のプロフェッショナル育成セミナーでは，ソフトウェア・プロジェクトの失敗例のケース・スタディ，EDP プロフェッショナルの評価，EDP マネージャの時間管理，コンピュータ・コンサルタントの役割，コンピュータと法律，ワード処理とデータ処理の統合，ビジネス・コンピュータの選択法，電子郵便システムの動向，データ解析技術，COBOL プログラマの生産向上，1980年代のエンド・ユーザ・ファシリティ，データ処理プロフェッショナルの動機づけ，災害からの回復，コンピュータ・システムの利用における人間工学，コンピュータ・グラフィックス MIS (Management Information System)，ストレスの解消法，非言語的コミュニケーション (Non-verbal Communication) の重要性，品質管理サークル活動の効用などであった．

また，一般講演は，ソフトウェア工学，マネジメント/教育，データベース/分散システム，人間および社会的問題，オフィス・オートメーション，意思決定支援システム，ハードウェア，通信/アプリケーション，パーソナル・コンピュータなどに分かれ，発表が行われた．

●20 th DAC (Design Automation Confererence) **——1983年6月27日～29日, Miami Beech**

本年の基調講演は，S. Mayo 氏 (Bell Laboratories) によって「設計自動化—過去の教訓と未来への挑戦 (Design Automation—Lessons of the Past, Challenges for the Future)」と題して行われた．また，パネル討論およびワークショップでは EWS (Engineering Work Station)，論理回路の合成，IC マスクの設計製造，IC の配置アルゴリズム，ロボット工学，CAD 用関係データ・モデルの将来，シミュレーション支援システムなどがテーマとして取りあげられた．このほか，グループ・テクノロジー，機器独立グラフィックス・システムの CAD/CAM 産業への影響と題してチュートリアル・セミナーが開かれた．

なお，Sperry 社からは，L.F. Todd，P.G. Kovijanic が，配置設計アルゴリズムおよびシミュレーション・エイドのセッション・チェアマンを務めるほか，H. Sumada 氏がパネラとして参加した．

●10 th Annual International Symposium on Computer Architecture——1983年6月13日～17日, Stockholm

本年の基調講演は，「コンピュータ・アーキテクチャの課題 (Current Issues in Computer Architecture)」と題して Manchester 大学の M.V. Wilkesda 氏によって行われた．なお，今回はチュートリアル・デーと称し，データベース・マシン，シリコンへの機能の移動の二つのセミナーが開かれた．このほか，最終日には，「応用人工知能およびそのコンピュータへの影響」と題しセッションが特設され，第5世代コンピュータのアーキテクチャに関する淵一博氏 (ICOT, 新世代コンピュータ技術開発機構)，内田俊一氏 (ICOT)，P. Treleavan 氏 (Newcastle 大学) の講演やパネル討論が開かれた．

●SIGPLAN '83 (ACM Special Interest Group on Programming Languages)**——1983年6月27日～29日, San Francisco**

本年は，「ソフトウェア・システムにおけるプログラミング言語」と題して開かれた．なお，今回のパネル討論では，「1980年代において，プログラミング言語のどんなアイデアが重要となるか」が論じられた．

また，一般講演の主なテーマは，分散型環境におけるソフトウェアの構成法，Von Neumann マシン用の開数型言語コンパイラ，属性文法とデータフロー言語，データ抽象化および制御抽象化のアプローチ，分散型プログラミング用言語，会話型ディスプレイ・インタフェース用プログラミング言語 Descartes，構文指向グラフィック・インタフェース，GLISP におけるデータ抽象化，高水準プログラミング言語とコマンド言語，UNIX 上の分散的コントロール・フローなどであった．

●ローカル・ネットワークの開発計画——Sperry社は，直接結合 (DC, Direct Connect) というベースバンド・バスによるローカル・ネットワークの技術を用い，柔軟性に富む多くのメーカの製品を持続できるローカル・ネットワークを開発中である．

これは，構内自動交換機 (PABX, Private Automatic Branch Exchange) と上述のトークン・アクセス方式のベース・バンド・バスやブロード・バンド・バスを組み合せたものである．このほか，Ethernet, ARCnet, IBM 型のリングなどの他のローカル・ネットワークとの接続を可能にするインテリジェント・ライン・モジュールの開発も現在計画されている．これらのすべてのローカル・ネットワーク関連の製品の設計のガイドラインとして，DCA (Distributed Communication Architecture) が用いられている．(E. D. Feist, "Local Network Developments", Proc. of the National Electronics Conference, Chicago, Illinois)

●障害分離へのシステム・アプローチ——本稿で紹介される IUCT (In Unit Card Test) システムは，U 1100/90 の可テスト性および保守性の要件を満たすために開発された．IUCTの目的は，①テストできない論理素子の数を最小化すること，②テスト・リストの生成時間の最小化，③障害チップの発見，④障害分析ルーチン (Post Fault Analysis Routine) の自動開始，などである．

IUCT は，IC およびカードにおけるピン接続の問題を発見するシステムで，Sperry 社の CAD システム UCADS のファイルにもとづき生成された刺激/反応パターン (Stimulus/Response Patterns) とスキャン・セット論理回路を使用している．筆者は，このシステムの採用によって 70 万論理ノードを越える大規模プロセッサの場合でも，90 パーセントの確率で 6 分間以内に 1.1 個のチップにまで障害個所を特定できると述べている．(G. Engel, "A System Approach to Fault Isolation", Sperry Spring, 1982)

●リアルタイム・プログラムのテスティング・システムを開発——システムの中核は，テスト記述言語で定義されたテスト・スクリプト（テストの台本）中のメッセージをあたかも端末機から投入されたように解釈するインタプリタである．このほか，同システムではスクリプトの作成や処理結果の比較を行う補助的プログラムも用意されている．テスト記述言語は高水準，かつ構造的なものである．また，一つのスクリプトから複数個のスクリプトを並列処理アクティビティとして実行したり同期をとったり，共通するテスト内容をスクリプトとして定義しておき，他のスクリプトから利用することも可能である．

現在，このシステムは Sperry 社のアプリケーション開発センターで，オンライン・プログラムのデバッグ，単体テスト，機能テスト，総合テストなどに用いられて成果をあげている．(R. D. Schlotfeldt, "Transa ction Testing System", Sperry, Spring, 1982)

●ソフトウェア仕様記述および開発支援ツール RDL/RDP——RDL/RDP は，要求仕様および開発用言語 RDL (Requirements and Development Language) とその処理プロセッサ RDP で構成されるシステムである．

大規模ソフトウェア開発における現在の課題は，ソフトウェア・プロダクト自体およびその開発を担当する組織の大きさと複雑性にある．

従来のソフトウェア開発ツールは，そのライフサイクルの各段階ごとに用意され，各段階を横断的に取り扱うものはなかった．RDL/RDP システムは，これらの各段階における情報を結びつけるもので，要求仕様がシステム定義によって，つぎにシステム定義が設計によって，設計がコーディングによっておのおの満足されているかどうかを確認できる．なお，本稿は RDL/RDP システムの概念および 1100 EXEC プロジェクトへの適用例について紹介している．(W. L. Claudle, "RDL/RDP: A Tool for Managing Software Complexity", UUA/E Conference, Geneva, Switzerland 1981)

●よいオペレータ・インタフェースの設計指針——複雑な自動システムの登場と共に人間・機械インタフェースの改良の重要性が増大してきた．よいオペレータ・インタフェースを設計するためには，①ユーザのクラス分けおよび各クラスで必要とされる知識水準の把握，②システムの代表的なユーザが用いているシステム評価法の理解，③システムの応答方法（例外処理の程度およびトランザクション処理の速さなど）の決定が重要である．

オペレータ・インタフェースの設計ではこれらを踏まえ，①システムの好感性 (System Sympathy)，②グラフィック・インタフェース，③応答時間/性能の基準，④コマンド/コントロール言語，⑤例外処理，などを考慮する必要がある．

一方，オペレータがシステムを熟知していれば，判断を要求される業務にもより積極的に取り組めるし，「システムに使われているのではなく，システムをユーザが自由に駆使している」という感覚が得られてシステム好感性が増大する．このため，ユーザの知識レベルに合せたオペレータ・インタフェースの設計が肝要である．

たまの利用者（カジュアル・ユーザ）に対するインタフェースは，通常 HELP あるいは DIALOG モードとして多くのシステムに組み込まれている．このレベルのインタフェースは，マニュアルの厚さが厚くなればなるほどカジュアル・ユーザ・インタフェースがより不親切な結果になる」という経験法則が存在する．

定常的利用者（プロダクション・ユーザ）へのインタフェースでは，そのコマンド言語は，省略形，ディフォールト値，コマンド複合文（1文で複数のコマンドの記載が可能）などの簡便記法へ向う傾向にある．

応答時間については，一般的に長すぎる場合に問題にされるが，短かすぎてもオペレータにストレスを与えることになる．このため，端末機での応答時間は，適切かつ一定であることが望ましい．

コマンド/コントロール言語は，システムの対話が1行づつ交互に行われる単一フレーム・インタラクション方式，メニュー形式の計画的順序方式，自然言語に類似した自由形式コマンドを用いる自由形式順序方式に分けられる．システム・メッセージについては，マニュアルを見なくても誤りの原因がわかるような情報を提供する必要がある．(D. K. Trombley, "A Description of the Requirements for Good Operator Interface", Sperry, Spring, 1982)

☆

技　　報

UNIVAC TECHNOLOGY REVIEW

No. 5

発　行　日	昭和 58 年 8 月 31 日
発行人兼編集人	富　田　和　夫
発　行　所	日本ユニバック株式会社 東京都港区赤坂 2-17-51 〒107 TEL (03) 585-4111 (大代表)
頒 布 価 格	1,500 円
印　刷　所	三美印刷株式会社