技報

UNIVAC TECHNOLOGY REVIEW

1982年2月　第2号

日本ユニバック

E. J. Weyuker の**プログラム・テストにおける神託の仮説については**，プログラム・テストにおいてしばしば取りあげられる仮説，「テスターまたは外部機構は，プログラムが作り出した出力結果が正しいか否かをいつでも正確に決定できる」との仮説の妥当性をエッセイ風に論じている．

プログラム・テストの分野での最大の研究目標は，テストについての健全な理論的基礎を発展させることと，プログラム中の誤りを検出したり，プログラムの正当性に関する確信を高める実践的な方法論を開発することである．もし，テスト結果の正否が決定できないとすれば，テストは合理的な工程であるとはいえないであろう．本稿は，上記の仮説の妥当性を検討し，その結果，この仮説は多くの場合に現実的なものでないとの結論に到達することを示す．さらに，この仮説を承認したときの影響も検討し，この仮説に代わる考え方を調査している．このようにして，本稿は，神託の存在仮説が多くの問題の潜在的な根源であることを気づかせる．

J. Nelson の**米国における商用暗号化システムの標準化**は，米国において現在行われている標準化の努力について述べ，近い将来使用できる商用暗号化システムを明らかにする．

端末間，端末とコンピュータ間，コンピュータ間における機密情報の転送に用いる商用暗号化システムを求める声が高い．多くの商用情報は非常に価値があり，保護を必要とする．一方，暗号化は費用の面から見て情報の暴露を容易に防ぎ，情報の改変をわけなく見つける唯一の効果的な方法であり，プライバシーを守るためにも使用される．暗号化システムを実現する製造技術はすでにあるものの大量生産を可能にするための標準化が遅れている．

日本でも進められているこの問題を考える際，米国でのこの資料は有効なものとなるであろう．

B. S. H. Wang の**コンピュータ・システムの構造的モデル化技法と性能評価**は，近年発展しているコンピュータ・システムのモデル化やその性能に関する解析技法のツールの１つである待ち行列網モデルを用いて，一般システムの観点から統一的なモデル化と評価技法を採用する試みを示す．ここでの試みとは，複雑な全体システムをいくつかのサブシステムに分解することによって管理可能な複雑さに軽減することであり，全体システムに対しては巨視的な分析を行い，サブシステムには微視的な分析を行う．

E. J. Weyker らの**プログラム・テストの理論と等質な部分領域の応用**は，Goodenough と Gerhart によるテスト・データ選択についての理論を拡大し洗練するために，等質なテスト基準と部分領域の考えを提案する．この考えの意図するところは，入力領域を部分領域に分割して，その同値クラスのデータはどれも正しく処理されるか，またはどれも正しく処理されないかの，どちらかになるようにすることである．プログラム・テストに関しては“プログラム・テストにおける神託の仮説について”と本稿をおさめた．

J. R. Jordan らの**マイクロプロセッサに支援されたデータベース管理**は，マイクロプロセッサを利用したデータベース・コンピュータのアーキテクチャとその操作についての報告である．この設計のポイントは，大きなデータ・ブロックを並列に転送し，マイクロプロセッサ群により連想記憶方式で並列処理する点である．なお，本システムは CODASYL 型やリレーショナル型データベース管理に必要な機能すべてをそなえている．

N. Relles らの**オンライン援助への統一的アプローチ**は，オンライン援助を効果的にするためにエンド・ユーザと援助提供者との必要性に立脚した枠組を明らかにする．すなわち，会話型ユーザが必要とする援助の型，わかりやすい援助機能を提供するために必要なデータ構造およびデータの関係，効果的な援助形態を促進・支援するソフトウェア・アーキテクチャ，オンライン援助を組み込み，それを維持していくために必要なプログラミング努力を取り扱っている．また，ソフトウェアのライフサイクルの各過程で求められる援助を統合する方法を提案している．

F. P. Mehrlich らの**移植性――高水準言語の作成事例**は，Salt Lake のソフトウェア開発センターで行った経験を報告している．これは処理系に依存しない単一の原始コードの ANSI COBOL コンパイラの作成報告である．この原始コードは，シリーズ 1100，シリーズ 80 の上で翻訳され，UTS 400 および UTS 4000 端末群で実行可能な目的コードとなる．

☆

論説 米国における商用暗号化システムの標準化

The Development of Commercial Cryptosystem Standards

Jim Nelson

要 約 端末間，端末とコンピュータ間，およびコンピュータ間における機密情報の転送を行うための商用暗号化システム (commercial cryptosystem) を求める声は多い．多くの商業情報は非常に価値があり，暗号化による保護に費用をかけるのも当然といえよう．本稿では商用暗号化システムの実現が遅れているのは，標準化が遅れているためであると考える．使用者が暗号化システムとしての製品を広く享受できるためには，この標準化が必要である．また，広く享受できるためには暗号化製品価格が妥当な値となるだけの生産量を保証する必要がある．さらに，本稿では，標準化に対して現在行われている努力について述べ，近い将来使用できる商用暗号化システムを明らかにする．

Abstract There are many requirements for commercial cryptosystems for transmission of intelligence between terminals, between terminals and computers, and between computers. Much commercial information has great value and can warrant the cost of cryptographic protection. This paper claims that the slow rate of implementation of commercial cryptosystems is due to the lack of standardization. This standardization is necessary to achieve general acceptance of cryptosystem products by users. General acceptance is required to permit production volumes which lead to reasonable crypto product prices. This paper discusses the standards efforts that are now ongoing, and projects the types of commercial cryptosystems which will be available in the near future.

1. 商用暗号化の必要性

コンピュータ・データバンクに入っている商業情報は，価値あるものである．実際，このような情報の所有を権力の一様態として見る見方もある．情報のように価値があり，権力を生むものは，それが攻撃されやすいときにはとくに，保護されなければならない．それは，情報があるところから別のところへ転送されるときである．転送は，ケーブル，無線，人工衛星を使って行われる．また単に，磁気テープ・ファイル等の媒体の物理的な輸送であったりする．

電子資金転送や店頭売上 (point-of-sale) 転送の分野では情報の流出や改変の恐れのあることが非常に重大な問題となる．暗号化 (encryption) は，これらの分野での保証の手段やメッセージ改変に対する保護の手段として使用される．暗号化の価値はこれに留まらず，顧客ファイルとか営業計画のように機密度の高いデータベースを所有する大企業においてもまた，非常な利用価値を持つ．近い将来，米国政府は，すでに制定されている“プライバシー法”[1]により，ある種の情報の保護を制度化するだろう．暗号化はコストの面から見て情報の暴露を容易に防ぎ，情報の改変をわけなく見つける唯一の効果的な方法なので，こうしたプライバシーを守るためにも使用されるであろう．

2. 暗号化システム標準化の必要性

商用暗号化システムの市場における必要性は明白であるが，まだ大量生産が可能な段階

Reprinted from CRYPTOLOGIA, (October 1980), pp. 213-224.

には至っていない．商用暗号化システムは，費用が低く，誤り率も低くなければならず，人間の介在をほとんど必要としないほどに自動化されていなければならない．

製造技術はすでにあるが，大量生産は作り出される暗号化システムの種類が非常に少ない場合にのみ可能となる．そしてこのときにのみ，製造企業は必要な低費用で実現するために欠かせない大市場を持つことができる．人間の介在を最小限にし，誤りが生じにくい操作を可能とするには，誤りとステータスの処理およびキーの交換方法に対して，前もってプロトコルを定めておく必要がある．このようにして，世に広く受け入れられるような商用暗号化システムは暗号化製品の大量生産を可能にする標準化を待っている状態である．

3. 標準化の進展

どんな標準化でも，その進展はゆるやかに進行するものである．暗号化システムの標準の作成も，その問題の複雑さと産業界になじみの薄いことから，なかなか実現しえなかった．従来，書き物が専門化されていて一般に広められていないような領域において，標準化を推進する人々はうまく立ち回らなければならなかった．標準が採用されるためには標準化の構成員のうちの 2/3〜3/4 の承認を通常必要とする．構成員が新しい標準の技術的および経済的効果をそれぞれ理解するまでは，この承認を得ることはむずかしい．さらに，標準を作成するグループがいくつかあれば，ある標準が採用される前に，これらのグループ間での調整にも多くの時間が費やされる．

4. 暗号化システムの標準化に関する団体

いま，5つの主要な標準化団体が，データ処理およびデータ通信業界で使用するための暗号化標準を作成している．その団体とは，U. S. National Bureau of Standards (米国政府標準局：NBS)，U. S. National Communication System (NCS)，U. S. National Security Agency (国家安全保障局：NSA)，American National Standards Institute (米国規格協会：ANSI)，および International Standards Organization (国際標準化機構：ISO) である．

商用暗号化方式に関するアメリカ合衆国の標準は，政府標準局 (NBS) によって出される合衆国情報処理標準 (FIPS PUB) と，U. S. General Service Administration によって出される連邦標準 (FEDSTD) からなる．これらの標準は，米国政府にシステムを納入する際に適用される．

米国規格 (American National Standards) は，民間企業グループの代表者によって作られ，商用の製品や適用業務の開発に適用される．国際的な標準は，スイスの Genéva にある国際標準化機構 (ISO) で作られる．その構成会員はそれぞれ，自分の標準化のための組織を代表している．

5. 米国の標準化の動向

5.1 データ暗号化（アルゴリズム）規格 (DES)：FIPS PUB 46

データ暗号化（のアルゴリズム）規格，すなわち DES は，国家安全保障局 (NSA) の助けを借り，政府標準局によって選択され，1977年1月に FIPS PUB 46 として制定された[2]．この合衆国情報処理標準は，データ暗号化のアルゴリズムが変更されることのないように，ソフトウェア・プログラムではなく，ハードウェアで実現することを要求している．

5.1.1 電子コード・ブック・モード

DES は，64 ビットの入力(通常 8 文字の英数字)を 56 ビット・キーのコントロールのもとで 64 ビットの出力に変換するという，暗号化の“電子コード・ブック”モードを定義している．交換アルゴリズムは 2 つの形式をとる．最初の形式の暗号化は，2 番目の形式の復号化のちょうど逆の変換である．キーKのもとで平文を暗号化した後，同じキーKのもとで暗号文の復号化を行い，もとの平文を作り出す．

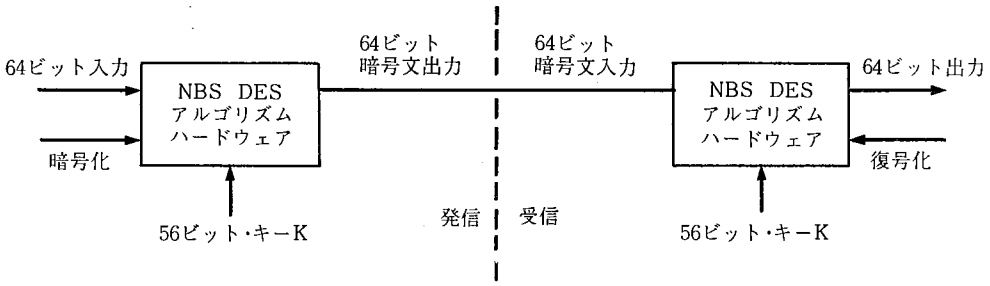


図 1 電子コード・ブック・モード
Fig. 1 Electronic code book mode

このモードは，“電子コード・ブック”という名前が付けられている．なぜなら，どのようなキーに対しても，同じ 64 ビットの入力でいつも同じ 64 ビットの出力が作り出されるからである．これはちょうど巨大な電子コード・ブックによって，平文または暗号文を捜し出すのに似ている．他のコード・ブックと同様に 64 ビット，つまり 8 文字のグループごとの対応づけが可能となる．したがって，電子コード・ブック・モードは平文を暗号化する方法としてはあまり勧められない．

5.2 テレコミュニケーション暫定規格：FEDSTD 1026

その結果，米国政府は，“データ暗号化規格と共に使用するテレコミュニケーション暫定要求”と呼ばれる，まだ採用されていない規格 FEDSTD 1026[3] を別に起草した．

この規格はフィードバックの技法を使ってコード・ブックにおける予測可能性を排除した 2 つの暗号化モードを定義している．これらの暗号化のモードは，“暗号フィードバック(CFB)”と“暗号ブロック連鎖(CBC)”といわれる．

5.2.1 暗号フィードバック・モード

暗号フィードバック・モードは，64 ビット以下のデータ・グループ(1 つのテレタイプまたはコンピュータ文字)を暗号化の単位とする．当然，電子コード・ブックで可能であった平文と暗号文の間の対応付けは不可能となる．

暗号フィードバック・モードでは，発信側および受信側の暗号化装置は両方とも“暗号化装置”として働き，同じ 56 ビット・キーと初期値を使って，まったく同一の暗号ストリームが生成される．暗号ストリームは発信側の暗号化装置でビットごとに(つまり mod 2 で)平文に加えられ，暗号文が作り出される．暗号文は，受信暗号化装置でビットごとに(つまり mod 2 で)暗号ストリームに加算されて，もとの平文が得られる．

発信される暗号化文字と受信された暗号化文字はフィードバック・レジスタの最下位部分に移され，次のビットごとの加算に使用するストリームを作り出すために使用される(図 2 参照)．

暗号フィードバック・システムは，同期を取るために暗号化されないで転送される初期値の転送の安全度に依存する．初期値の転送は通常，各方向の最初の転送を行う．2 方向転送システムでは，初期値は回線ごとにやりとりされる．

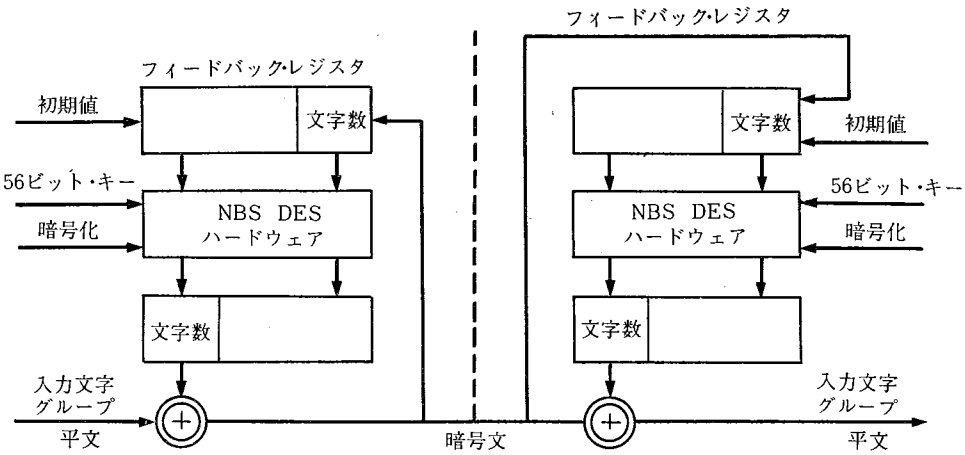


図 2　暗号フィードバック・モード
Fig. 2　Cipher feedback mode

受信側の暗号化装置で暗号化ビットに誤りが検出されると，誤りのビットとそれ以降の数文字分のビットは誤りのビットがフィードバック・レジスタからシフトされて出てくるまで捨てられる．その後，受信側の暗号化装置は発信側暗号化装置と共に再同期化され，正しいデータが受け取られるようになる．

5.2.2　暗号ブロック連鎖モード

DES のアルゴリズムを利用する暗号ブロック連鎖モードは，通常，文字の長さを独立して扱うことができるような“知的な”コンピュータと端末において使用される．暗号ブロック連鎖モードは，64ビット(通常8文字の英数字)データ・ブロックを暗号化する．もちろん，フィードバック技術を使っているので，電子コード・ブックで可能であった平文と暗号文との間の対応付けはなくなる．

同一の56ビット・キーを用いてデータを暗号化したり復号化する前に，まず発信側および受信側の暗号化装置のフィードバック・レジスタに，同一の初期値を入れておかなければならない．発信側は暗号化モードで，受信側は復号化モードで稼動する．

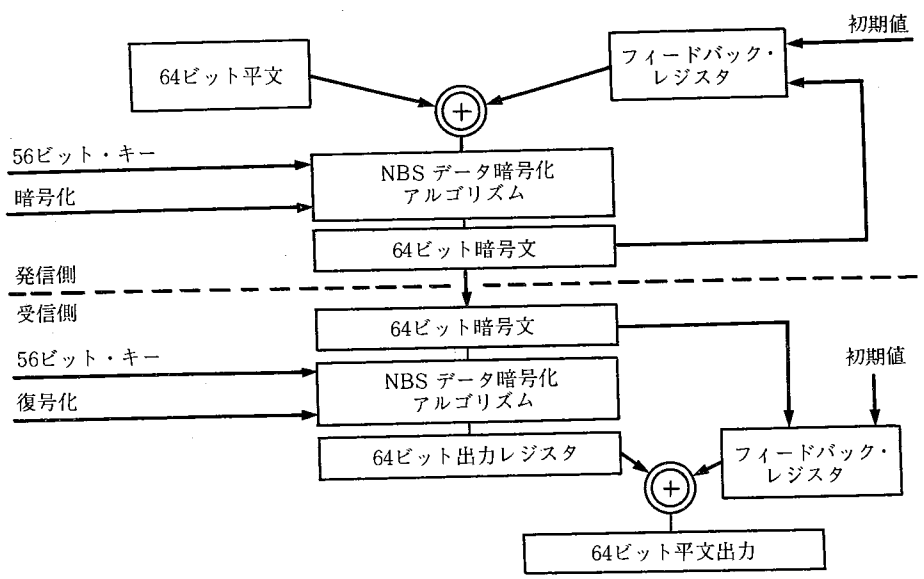


図 3　暗号ブロック連鎖モード
Fig. 3　Cipher block chaining mode

データを暗号化するには，まずフィードバック・レジスタに初期値をセットする．64ビットの平文は，64 ビット・フィードバック・レジスタの内容と (mod 2で)加算される．その結果生じる64 ビットは，データ暗号化アルゴリズムに送られ，56 ビット・キーの下で，暗号化モードで処理される．生成された64 ビットの暗号文が，受信先に転送される．当然ながら，暗号文の64 ビットは，暗号化装置の次のサイクルにおいて，その後のデータを暗号化するために，フィードバック・レジスタにセットされる．

受信側では，フィードバック・レジスタに最初に初期値がセットされる．受け取った64ビットの暗号ビットは，直接，データ暗号化アルゴリズムに入れられ，56 ビット・キーのもとで復号化される．受け取った暗号文も当然次のサイクル用にフィードバックし，レジスタにセットされる．データ暗号化アルゴリズムで処理されて得られた出力はフィードバック・レジスタの値と(mod 2 で)加算され，64 ビットの平文が作り出される．受信された暗号文は，次の復号化のためにフィードバック・レジスタに入れられる．これらを図示したものが図3 である．

受信側暗号化装置で暗号ビットに誤りが検出されると，誤りのビットを含む64 ビット平文ブロックとそれに続く64 ビットの暗号文に対応する64 ビットの平文が捨てられる．その後，受信側暗号化装置は発信側と再同期化され，正しいデータを受信する．

計画中の FEDSTD 1026 のその他の部分ではリンクやそれ以上の高レベルでの暗号化されたコミュニケーション用のプロトコルを規定する．とくに，これらのプロトコルは，主要なリンク・コミュニケーションにおいて，暗号化の開始と終了の時点を規定している．現在計画されている規格では，メッセージの挿入や削除，またはメッセージの改変を検出できるように(リンク・レベルより)高レベルのプロトコルを規定することが考えられている．

5.3 テレコミュニケーション安全保障規格：FEDSTD 1027

この規格は，米国政府の非軍事的アプリケーションで暗号化システムを使用する際に必要な安全保障の要求を示している[4]．この標準に対しての最終的な責任を負う機関は国家安全保障局である．この標準は，キーの記憶，ステータスと故障の表示，封印物に対する侵害排除および手動のキー・ローディングに要求される安全保護策を規定している．また，電磁気による妨害行為および無線周波数干渉による妨害の抑制も必要としている．オプションとして標準的な携帯用キー・ローダ・インタフェースも規定されている．

6. 米国規格協会の活動状況(商用利用)

6.1 コンピュータと情報処理委員会 X3

1978 年8月に，コンピュータと情報処理に対する米国規格協会 (American National Standard Committee) X3 の SPARC (Stardards Planning And Requirements Committee) が，暗号化の製品に対する標準化計画作業を，X3/SPARC/ENCRYPT と呼ばれる研究グループに割り当てた．2年後，このグループは，ソフトウェアによる実行も認めた点を除いては，米国の DES とまったく同一のデータ暗号アルゴリズム規格 (DEA) を提案した．

このグループはまた，暗号化オペレーションのモードに関する別の規格を作成するための開発プロジェクトを発足させた．この作業は新しい専門委員会 X3T1 に割り当てられた．“オペレーションのモード”の規格とは FEDSTD 1026 におけるモード記述と似ているが，出力フィードバックと呼ばれるモードが1つ追加指定されている点が異なる．このモードは，順方向誤り修正を人工衛星システムで使用したり，ファクシミリ・システム

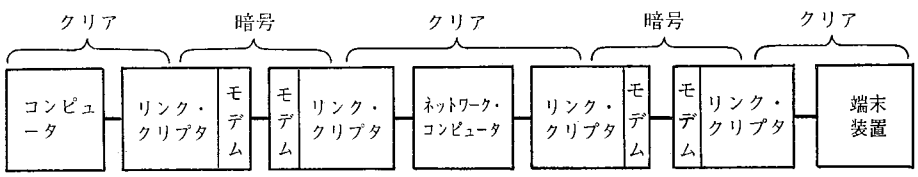
で，望むならば，単一ビットの誤りを無視できるようにするために，ビットの取捨(garble extension) を禁止している．

その審議において，X 3/SPARC/ENCRYPT では，暗号化および復号化の機能はコンピュータと端末装置との操作において，いくつかの異なる場所に置くことができると記している．

6.1.1 リンク間の暗号化

暗号化または復号化機能を置くのに最も簡単な場所は，端末装置と通信回線，またはコンピュータと通信回線の間である．暗号化機能(または装置)は，回線(またはリンク)の各終端で必要であり，このため，この暗号化装置はリンク・クリプタ (link cryptor) とも呼ばれる．

回線交換システムの場合



メッセージまたはパケット交換網の場合

図 4 リンクの暗号化

Fig. 4 Link encryption

このリンク・クリプタの機能は，コンピュータまたは端末装置を変更する必要がまったくなく，通信リンク上を転送されるメッセージ・テキストをすべて暗号化(または復号化)する．このようなシステムでは，コンピュータと端末装置は暗号化に対して特別な配慮をまったく払う必要はないといえる．

しかしながら，独立した“リンク・クリプタ”が，どのリンクの終りでも必要であり，数百以上のリンク上を転送されるデータの暗号化を要求されるような大きなコンピュータ・システムにおいては，このような暗号化システムは経済的でないかもしれない．

さらに，リンク・クリプタは今日の多種多様な回線交換網においては十分に効果的であるが，コンピュータと端末間通信に採用されつつある新メッセージおよびパケット交換網においては十分なデータの安全保護を与えるものではない．

回線交換システムで，使用者は必要な時間，端から端までの全回線を借り受ける．したがって，ある意味で，メッセージは全システムを通じて安全に保護される．

しかし，メッセージおよびパケット交換網は非常に多くの独立したリンクまたは回線から作られている．各リンクまたは回線の間に置かれるネットワーク・コンピュータは，暗号化された情報を受け取り，その暗号を復号化し，チェックし，記憶して再構成し，再び暗号化して(恐らく別のキーを使って)，最終的にその暗号メッセージを目的地に向けて次のリンクに再転送する．明らかに，転送中の情報は各ネットワーク・コンピュータで危険にさらされる(暗号文でなく平文である)．さらに，キーの管理はメッセージまたはパケッ

ト交換網に委任されており，このこと自体が安全保護上の別の問題になってくる．

6.1.2 終端間の暗号化

メッセージとパケット交換網で実際に望まれるのは，転送に使われる回線とは無関係に，発信側コンピュータですべての情報を暗号化するのに使用され，受信に使われた回線あるいは多数の通信リンクとは無関係に，受信側コンピュータで受け取られたすべての情報を復号化するために，使用される単一の暗号化装置，または機能である．

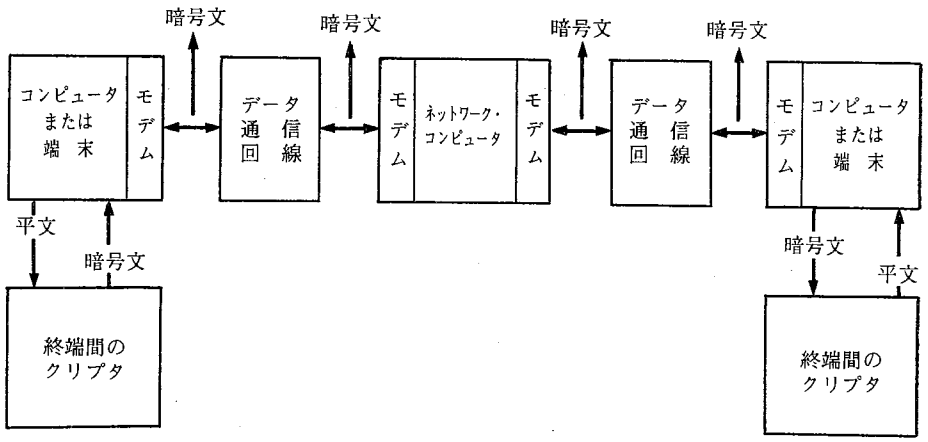


図 5 終端間の暗号化
Fig. 5 End-to-end encryption

この両機能を有する暗号化装置は，端末/コンピュータ使用者のそれぞれ異なる組合せに対し，異なるキーが使用できなければならない．さらに各端末には，それ自身の個別キーを持つ(おそらく組み込まれた)暗号化機構が要求されるであろう．このようなシステムになってはじめて，本当の“終端間”の暗号化を提供することが可能となる．

“終端間”暗号化に対しては“リンク間”の暗号化に対して必要である以上に，より複雑なキーの管理プロトコルが必要とされる．なぜなら，すべての新しいメッセージまたはメッセージ・グループに対して，端末のワイヤ式マスタ・キーの下で暗号化された新しいキーが転送されなければならないからである．

本当の“終端間”の暗号化を実現することは，それがコンピュータと端末のハードウェアおよびプログラムの修正，または新しいデザインを必要とするので，“リンク”の暗号化を実現するよりもむずかしい．

6.1.3 物理的データ交換媒体の暗号化

同時に，このような暗号/復号化両機構を持つ装置は，他のコンピュータ・システムとの物理的データ交換用に使われる磁気テープ・ファイルを暗号化および復号化するのにも使用できる．暗号化した後で，データは通信システムではなくテープ制御装置に送られる．復号化は，通信装置から情報を得てからではなく，テープ・ファイルを読み込んでから行われる．

これらの可能性を考慮した後，X3/SPARC/ENCRYPT は，3つの通信と1つのデータ交換媒体(テープ)の暗号化規格を開発することを提案した．最初の規格は，リンク間で，暗号化をどのように行うかを規定する．2番目の規格は，パケットまたはメッセージ交換網で，終端間の暗号化をどのように行うかを規定する．また，物理的データ交換媒体(磁気テープ)の暗号化を行うために，同じ暗号化装置の使用も認める．さらに，3番目の規格は，リンク間と終端間の中間レベルでの暗号化を規定するものとして，提案された．こ

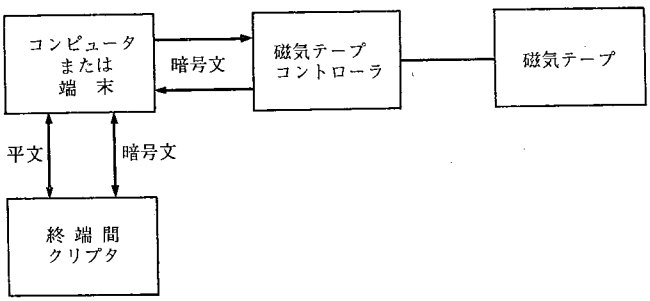


図 6 物理的データ交換媒体（磁気テープ・ファイル）の暗号化
Fig. 6 Physical interchange (magnetic tape file) encryption

の規格は，各リンクで復号化および再暗号化されることなく，暗号化されたメッセージがネットワーク全体に転送されることを認めている．ただし，ユーザ間でキーの共用がいくらかあるであろう．この暗号化は，テープ・ファイルを暗号化するのに使用することはできない．

第4の規格は，データ交換媒体(磁気ファイル)の暗号化をどのように行うか規定するために提案されている．この種の暗号化には別のキー管理の方法が必要であり，独立した規格を設けるのが最も良いからである．

現在(1980年4月)，X3. S38 データ通信規格グループによって，リンク暗号化の規格の仕事は，順調に進んでいる．通信およびデータ交換媒体の暗号化を含む，その他の規格の原案はすでに作成されており，規格文書化グループで処理されるのを待つばかりである．

6.2 金融サービス規格委員会 X9

金融サービスおよび銀行界で使用する標準は，米国規格協会（ANSI）の金融サービス規格委員会 X9 によって開発されている．現在，2つの専門委員会が，それらの業種に対する暗号化システム・アプリケーションの標準化に携わっている．これらの専門委員会とは，銀行カード規格を作る X9. A3 と，資金転送メッセージ認証規格を作る X9. E8 である．

6.2.1 銀行カード安全保障規格

ANSI X9. A3 は，銀行カード安全保障に対する規格の立案に責任を負うグループである．銀行カードは，機械で読取り可能な磁気ストライプを張りつけたプラスチック・カードである．これらのカードは盗まれたり偽造されるかもしれないので，PIN (private identification number) と呼ばれる秘密の個人識別番号を使って確認の手段とする規格が立案された．PIN は，4～12 桁の 10 進数で，顧客が銀行自動窓口装置に取り付けられた電話形式のキー・ボードから入力するものである．提案された規格は，窓口装置でも，中央コンピュータでも，データ暗号化の規格を使用することで本人の確認をすることを考えている．

自動窓口装置で確認が行われる場合は，PIN は磁気ストライプ上に暗号形式で記憶されるか，または何らかの方法でその情報から導出できなければならない．記憶された情報は，磁気ストライプ上に記憶された他の顧客データと組み合わされた秘密のキーを使って，窓口装置で暗号化された顧客の入力した PIN と比較される．

窓口装置とデータ回線または電話回線で接続されたコンピュータで本人の確認を行う場合は，おそらく磁気ストライプ上に記憶された他の情報とともに，PIN は暗号化され，確認のためにコンピュータへ転送される．

本人であるとの確認が得られ，全科目の銀行バランスが満たされるというような，他の条件が満たされると，トランザクションが処理され，記録され，現金が顧客に支払われる．

PIN の長さを短く(4〜6 桁の 10 進数)制限すべきかどうかについては，銀行界で意見の相違がある．いくつかの銀行協会メンバーは普通の顧客が 4〜6 桁以上の 10 進数を記憶するのは困難であると主張している．56 ビットの DES キーは自動窓口装置内に安全に保持され，遠隔地で本人の確認を行う方式では遠隔地側コンピュータ内にも保持されている．

個人識別番号(PIN)の管理と安全保護に関する規格原案，および暗号化で使用するキーの管理に関する規格原案は，X9. A3 から出されている．この規格案は，近い将来，論評を得るために配布される予定である．

6.2.2 資金転送メッセージ認証規格

ANSI X9. E8 は金融機関の資金転送メッセージ認証のための米国規格案を作成する作業グループである．提案された規格は，金融機関の間，および金融機関とその顧客間における通信メッセージの紛失，重複，悪意のある変更，または偶然の改変から保護する方法を規定するためのものである．

提案された規格は，暗号化技術と双方が承認したキーに基づいて，メッセージを発信する機関側で厳密な認証を行うものである．

資金転送メッセージそれ自身は暗号文でなく平文で転送される．56 ビット・キーを使った CFB モードで稼動する DES 規格は，資金転送メッセージに付加される認証テスト・ワードを作り出すために使用される．

この規格は，メッセージの安全保護をより強めるために，他の暗号キーとか暗号モードを使用して，より強力な暗号化を行うことを禁止してはいない．

X9. E8 委員会は，資金転送メッセージ認証に対する規格案を完成させるために，1980 年末を目標にして作業中である．

7. ISO 標準化の状況

国際標準化機構(ISO)内の専門委員会 97(TC 97)は，国際暗号化システム規格に対する必要性を調査する責任を持っている．国際暗号化システム規格に要求される条件を決定する責任のあるこの作業グループの委員長には，英国 Middlesex の Teddington にある英国物理学研究所の Donald W. Davies が任命されている．

8. その他の標準化グループの状況

米国電気電子学会(IEEE)には，Wisconsin 大学の George Davida 教授が指導して現在活動中の専門委員会がある．この専門委員会は，現在データの安全保護と暗号化システムに関するデータの収集を行っているが，いまだに規格の作成までには至っていない．

9. おわりに

今まで述べた活動からわかるように，暗号化システム標準の作成は非常に複雑で，相互に絡み合っている．しかし，この問題は時節に合うように処理しなければならない．もし，暗号化システムの標準が適時にこれらの標準化作成団体から出されなければ，事実上の規格化が実際に起こるまで，相互に相入れないシステムが乱立し，その結果はこの業界での“バベルの塔”(言語の世界において似た言語が多数開発され，その結果，混乱が生じ

る現象)の出現となるであろう.

(システム統括一部 システム推進部　河上 一郎 訳)

参考文献

[1] J. Nelson and D. Reisman, *Consideration of privacy and encryption in personal, national and multinational communications*, Pacific Telecommunications Conference, Hawaii, 1980.

[2] National Bureau of Standards, *Data encryption standard*, FIPS PUB 46, U.S. Department of Commerce, 1977.

[3] National Communications System, *Telecommunications: interoperability requirements for use of the data encryption standard in data communications (Draft)*, Proposed FEDSTD 1026, 1978.

[4] National Security Agency, *Telecommunications: security requirements for use of the data encryption standard (Draft)*, Proposed FEDSTD 1027, 1979.

[5] J. Nelson, *Implementations of encryption in an open systems architecture*, Computer Networking Symposium Gaithersburg, Md., 1979.

執筆者紹介　Jim Nelson

Illinois 大学卒業後, Sperry Univac 社に入社, 論理設計に従事した後, システム工学に従事し, Department of Transportation の Air Traffiic Control Advisory Committee の仕事を行う. ANSI X 3. SPARC/ENCRYPT 規格委員会の副委員長を務め, 現在, ローカル・ネットワークの仕事に従事し, Roseville Development & Manufacturing, Communications and Distributed System Group の Staff Engineer である. IEEE 会員.

報告 マイクロプロセッサに支援されたデータベース管理

Microprocessor-Assisted Data Base Management

J. R. Jordan, H. A. Freeman

要 約 データベース・コンピュータは，豊富な機能を提供する一方，現在のソフトウェアであるデータベース管理システムの効率を改善するためのハードウェア技術の1つとみなされている．本稿は，この領域における Sperry Univac 社の研究の1つであるマイクロプロセッサを利用したデータベース・コンピュータのアーキテクチャとその操作についての報告である．この設計の主眼は大きなデータ・ブロックを並列に転送し，マイクロプロセッサ群により連想記憶方式で並列処理することである．本稿で報告するデータベース・コンピュータは，CODASYL 型やリレーショナル型のデータベース管理システムを支援する際に必要なすべての機能を持つ．

Abstract Data base computers have been identified as one means of using hardware to improve the performance of current Data Base Management (Software) Systems while offering increased functionality. This paper describes a microprocessor-based data base computer architecture and its operation that was developed during one of Sperry Univac's research efforts in this area. Parallel transfer of large blocks of data which are then processed in parallel on a content-addressable basis by a series of microprocessors is the key to this design. The data base computer described in this paper performs all of the functions required to support a CODASYL or a relational deta base management system.

1. はじめに

1980年代において，ソフトウェアとしてのデータベース管理システムは，使いやすさと性能という2つの大きな要求に応えなければならない．

現に，データベースはますます大きくなっており，それらの利用も急速に増え続けている．その一方で，データベースの呼出し方はますます複雑化している．さらに，大規模なシステムの傾向として，大容量の情報処理や利用者の問合せに処理効率上の問題が生じてきている．使いやすさの追求からリレーショナル・データモデルが提案された[1]が，現在の逐次処理型のコンピュータでは，まだ十分な効率が得られていない．このため，このモデルへの切換えが余り積極的に行われていない．現在のデータベース管理システムとリレーショナル型のデータベース管理システムの効率を改善するには，これ専用のハードウェアを利用すると効果的である．現在，特別な目的のコンピュータはコミュニケーション処理やアレイ処理，保守処理等に利用されているが，さらにデータベースの管理のためにデータベース・コンピュータが提案されている[2]．

ハードウェアであるデータベース・コンピュータ・システムは，豊富な機能を提供する一方，ソフトウェアとしての現在のデータベース管理システムの効率を改善する．LSI, VLSI, マイクロプロセッサ，磁気バブル記憶，CCD のような新技術を利用すれば，データベース・コンピュータは，1980年代の情報の蓄積と処理によく適合したものになるだろう．このような特殊目的のコンピュータは，効率の改善に加え，汎用のホスト・コンピュータの資源を他の仕事のために解放し，機密保護の問題を支援するハードウェアを提供

し，ネットワークでデータを共用するのにもっと効果的な方法を提供することになる．

2. 従来の方法

データベース・コンピュータ・システムに対して2つの異なった方法が提案され，実現されてきた．その1つは，データベース管理システムをホスト・コンピュータから汎用のミニコンピュータへ移す方法である．この型のシステムは，十分実現可能であることが証明されたが，効率の改善までには至らなかった[3]．効率の改善が果たせなかった主な理由は，データ管理機能の実行を，伝統的な逐次処理型のプロセッサで行ったことによる．

第2は，データ管理の仕事の一部分，またはすべてを実行するために特別な媒体を使用する方法である．この領域においては，ICL 社が CAFS[4]と呼ばれる製品を公表している．ICL 社は，内容による番地付けを使用したレコードの高速選択のために，ミニコンピュータに特別な選択ロジックを結合した．CAFS は，3つの専用プロセッサ CASSM[5], RAP[6]および DBC[7]と結合している．

3つの基本的なデータベース・コンピュータのアーキテクチャの中で，DBC が商用コンピュータ会社の観点からは最も魅力的なものである．CASSM は，任意の回転記憶装置に適合するものである固定ヘッド・ディスク技術を使用している．この方法では，各トラックごとに読み書きヘッドを必要とし，大容量データベースを支援するには非常に高価なものになるであろう．RAP は大量のステージングを必要とするため，効率改善の障害となるだろう．これらの問題を避ける試みに加えて，DBC はリレーショナル・データベース管理システムと同様に CODASYL 型をも支援することを目指した唯一のデータベース・コンピュータである．

3. 現在の方法

DBC の方法の利点を確認し，実際にデータベース・コンピュータに要求される機能は何かを知るために，まずアプリケーションの調査と分析をデータベース・アプリケーションの設計者と利用者に対し行った．アプリケーションの範囲は，トランザクション処理で，速い応答時間で単純なデータ呼出し手法を行うエアライン予約アプリケーションから，大量のバッチ向け処理の報告書作成のアプリケーションまでを含んでいた．調査結果とその後の分析は，データベース・コンピュータのための要求をまとめるのに，極めて有効であった[9]．すなわち，DBCの方法(移動ヘッド・ディスクの1つのシリンダに対応した大きなブロックにデータを集めることを提案している)は，ほとんどのアプリケーションに対し最も効果的であり，また，ほとんどのアプリケーションは，更新処理を許す利用者の数に制限を置いていない事実が，明らかになった．さらにこの調査で，大部分のデータベース利用者が，1980年代に100億～500億バイトのデータベースを持つことになるであろうことが明かになった．

DBC の設計は，データベース・コンピュータ・システムへの要求をもとに，改訂・拡張された．辞書情報とデータを高速に検索する基本的な方法は，もとの DBC と同じである．大きなデータ・ブロックの並列転送と，連想記憶方式での処理がこの改訂設計の重要な点である．この機能を実現するには，完全結合処理と完全整列処理が必要となる．このためにプロセッサ間のコミュニケーションを追加した．また探索キーを一意に識別し，属性値で整列できるように，固体連想処理要素が考案された．4章で報告するように，この改訂設計により，著しく効率が改善された．

4. データベース・コンピュータの設計

4.1 アーキテクチャ

拡張されたデータベース・コンピュータのアーキテクチャを図1に示す．従来の DBC の設計では，命令あるいは問合せの処理のために2つのループを使用している．1つは構造ループである．このループは求めるレコードを含んでいるシリンダを一意に決定し，基本的な機密保護の検査を行い，データベース中に挿入されるレコードをひとまとめにする．もう1つはデータ・ループで，データベースを呼び出し，格納し，検索したレコードの後処理を行う．構造ループもデータ・ループも同じ並列処理の技術を使用するので，この改訂設計では，上記の機能を1つの構造にまとめている．

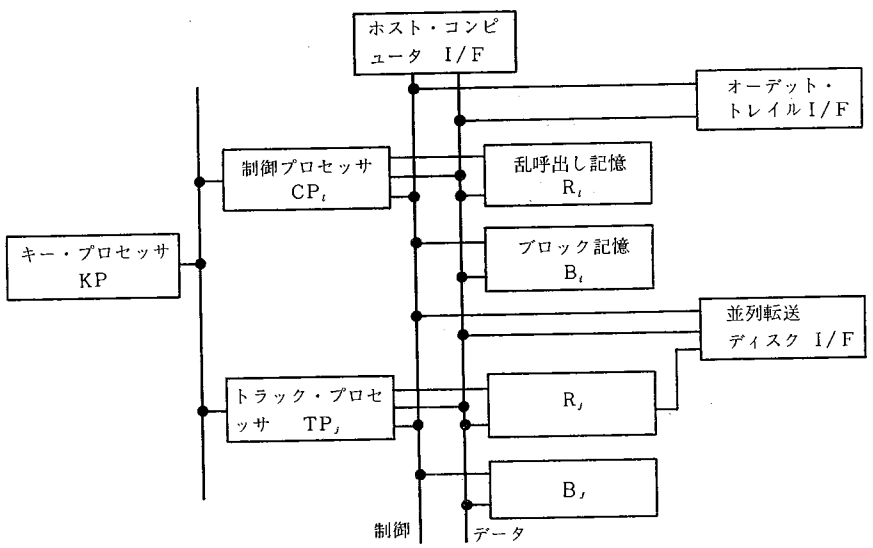


図1　データベース・コンピュータのアーキテクチャ
Fig. 1　Database computer architecture

データベース・コンピュータの設計は，次の5つの要素からなっている．

1)　制御プロセッサ (CP_i)
2)　乱呼出し記憶モジュール (R_i)
3)　ブロック記憶モジュール (B_i)
4)　トラック・プロセッサ (TP_i)
5)　キー・プロセッサ (KP)

これらの要素，相互の関連，外部的に結合された媒体とのインタフェースについては，以下で説明する．

マイクロプロセッサで実現される制御プロセッサは，ホスト・コンピュータからの命令を受け付ける．それらの命令はソフトウェア・インタフェースに従って，CODASYL 型のデータ操作レベルの命令であったり，Sequel[10] あるいは QLP[11] の命令のようなリレーショナル型の高水準の問合せ命令である．命令は制御プロセッサによって処理され，対応するパラメタと命令の集まりが，トラック・プロセッサ，キー・プロセッサ，オーディット・トレイルあるいはディスク操作のために生成される．

常駐化することのできないデータベースは，辞書情報と同様に，複数のディスク装置あるいは並列転送ディスク装置上に置かれる．一時に，1つのディスク装置から並列転送することは，効率や信頼性上好ましい．しかし，DBC 操作の基本は並列に複数のトラックを処理することにあるから，これらのトラックを複数のディスク装置から得ることもできる．ディスク制御インタフェースは，情報を媒体から乱呼出し記憶に直接転送したり，あるいはデータ・バスを通して間接的にブロック記憶に転送するといった2つのケースの情報転送をつかさどる．このインタフェースにおいて，ディスク制御装置に今日共通に見られる通常のエラー修復，縮退処理やその他の機能が提供される．1つのディスク装置の複数のトラックからデータを並列に転送する機能は技術的に新しいものではない．そこで300 メガバイトのディスクの1つを，最高9ディスク・トラックが並列に転送できるように改造した[12]．もっとも他のディスクであっても，簡単な改造によって，この機能が得られる．

現設計では，乱呼出し記憶モジュールには次のものが含まれる．

1) トラック・プロセッサ・プログラムとその局所変数
2) システム・テーブル
3) 大容量記憶装置との間で転送される情報のためのバッファ

乱呼出し記憶の正確な大きさはまだ決められていない．現在研究が進められている効率のシミュレーションの研究結果に従って決められることになるだろう．トラック・プロセッサや制御プロセッサによって実行されるプログラムは，基本的には単純であり，それほど大きな領域を必要としない．このプログラムを区分化することは許されないので，乱呼出し記憶モジュールはこのプログラムを常駐させることのできる大きさでなければならない．これに対し，システム・テーブルの情報は区分化することができる．現在の 112 語あるいは 448 語長のブロックのかわりに，たとえば1シリンダに相当する情報の大ブロックを使用すると，辞書情報を非常に削減できる利点がある．したがって，乱呼出し記憶モジュールの大きさを決めるのに，トラックのバッファリングに対する要求が有力な要因となる．いいかえれば，このことはトラック・プロセッサやディスクの相対的速度，2次記憶領域の存在に依存するということになる．現在のマイクロプロセッサとトラック当たり 1.2 メガバイト/秒のディスク比率 (Ampex 社の並列転送ディスク PTD-930 X) のパラメタを使用し，バッファ領域として3トラック分の大きさ(60 キロビット)があれば効率上十分であることが予測された．したがって，乱呼出し記憶モジュールの適切な大きさは 96 キロビットとなる．このブロック記憶には，乱呼出し記憶モジュールからのオーバフロー・データ，ホスト・コンピュータあるいはディスクに転送されるために選択されたレコード，使用者が認可した変更済みのデータが格納される．

DBC の多くの操作は，データに対し乱呼出しを必要としないので，DBC で提供される2次記憶領域(ブロック記憶)は，逐次型の記憶媒体で実現できる．磁気バブル記憶(MBM)や CCD，固定ヘッド・ディスク(FHD)の転送率は移動ヘッド・ディスク(MHD)よりも速く，その記憶域は RAM より低価である(図2参照)．

1980 年代にどれが最も費用対効果がよくなる技術かは，まだはっきりしない．しかし，バブル記憶の不揮発性は魅力のあるものである．また，RAM の費用は引き続き減少していく可能性を持っている．RAM が使えれば，ブロック記憶に対して記憶モジュールごとに2メガビット削減できる．

一般に，あるディスク・トラックからの情報は，同時に(すなわち1回のディスク回転

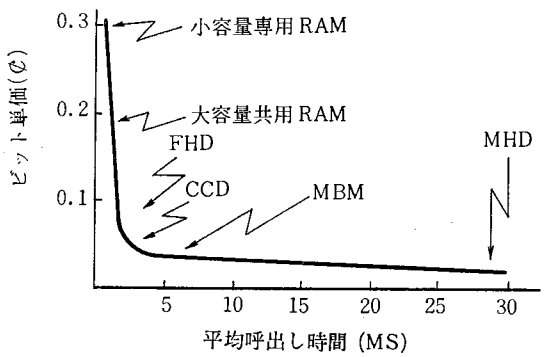


図 2 記憶媒体技術
Fig. 2 Storage technologies

時間内に)対応するいくつかの乱呼出し記憶モジュールに転送される．乱呼出し記憶モジュールに1トラックの大きさのデータが読み込まれると，要求されている機能を実行するために，トラック・プロセッサが非同期にデータを操作しはじめる．これらの機能の割当ては，制御プロセッサによって，乱呼出し記憶に対応したトラック・プロセッサ(TP)中のタスクの待ちを経て行われる．これらの機能については次節で説明する．TP を開発するために選ばれるマイクロプロセッサは，優れた縮退機能を提供している制御プロセッサのマイクロプロセッサと同じであるべきである．

マイクロプロセッサの適切な選択は，DBC の効率にとって重要な意味を持つ．マイクロプロセッサの処理速度とディスクからのデータ転送率の適合性をみることがこの選択の鍵となる．たとえば，1ディスク・トラックの大きさが 50 キロビットで，ディスクからのデータ転送に 20 マイクロ秒(MS)かかるとすれば，マイクロプロセッサの平均速度は，20 マイクロ秒ごとに 50 キロビットを越えなければならない．

トラック・プロセッサに要求される主な機能は，1つのディスク・トラックのレコードすべてにわたって1つの問合せを実行することである．この処理を NS 16032, MC 68000, Z 8000 等の 16 ビット MOS マイクロプロセッサの一群で実行したところ，要求される予測値 20 マイクロ秒よりもいくらか遅いことがわかった．たとえばトラックが 150 バイト長の約 350 件のレコードで 70 パーセント満たされているとする．このレコードに対する 4 述語の問合せに NS 16032 では総計 40 マイクロ秒かかっている．したがって，MOS は TP のマイクロプロセッサとしては不適当であろう．

2極 TTL では，費用は少々高くなるが，十分すぎる速度が得られる．この十分すぎる速度は，TP が実行しなければならない処理に用いることができる．たとえば，整列，更新，領域管理，ロック規約によるデータベースへの同時呼出しの整合性を保証する等，多くの機能の実行に使用できる．さらに，MOS 技術に対し，2極 TTL の使用による費用の増加は TP の全費用を多少増加させるだけである．

キー・プロセッサ(KP)は，属性値空間に広がったファイルを一時的に分割し，特定のデータベース操作を速くするために使用される特別な要素である．図3で示されているように，KP はさらに次の5つの要素からなっている．

1) 制御ロジック
2) 探索要素(SE_i)
3) 記憶モジュール(M_i)

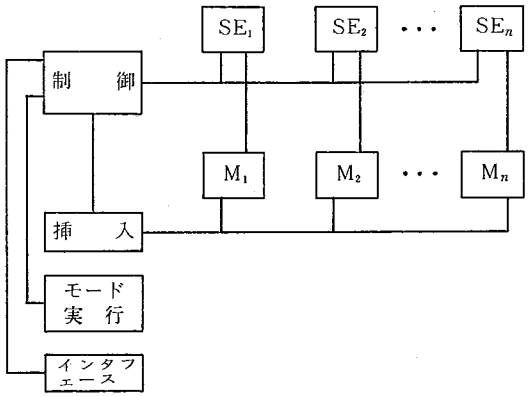


図 3　キー・プロセッサ
Fig. 3　Key processor

4)　挿入ロジック
5)　インタフェース・ロジック

KP は通常の RAM チップを使用するように設計されているが，現在，特注のチップを使用する可能性を研究中である．1つの可能性は，16 キロビットのチップを内部的に 256 ビットの 64 語のグループにすることである．これらのチップの各々には領域に加えて操作に必要となる探索ロジックのすべてが含まれることになる．

要求された仕事を実行するために，KP 中には次の 4 つの操作モードがプログラムされている．

1)　挿入 (insert)：与えられた引数 X がすでに KP 中に存在すれば，KP は，TP を呼び出すために“発生の順番”あるいは索引番号 k を返す．もし X が KP 中になければ，それを KP に挿入し，値 k を割り当て，TP の呼出しに戻る．
2)　照合 (match)：KP は，与えられた引数がすでに KP 中に存在するかどうかを判断する．もし存在すれば，非零値 k を返す．
3)　刻印 (mark)：与えられた引数 X に対応する KP 中のビットをセットする．
4)　検索 (retrieve)：2 つの型の検索が可能である．1つは，まだ刻印されていない引数 Y を検索し，その対応する刻印ビットをセットする．もう1つは，刻印されている引数 Z を検索し，その対応する刻印ビットをリセットする．

最初の 3 つの操作モードで KP に与えられる引数の列 X は，また計数閾値 C_t を伴っている．この値 C_t は，ブロック記憶に転送することを保証するために用いられ，十分な数のレコードが乱呼出し記憶内にあることを決定するのに使用する．DBC で実行される機能のうち，これらの操作の使用については次節で説明する．

使用者あるいはプログラムの誤りにより，不正データはいつでも発生しうる．しかし，ハードウェアの故障の影響は，耐故障性技術の使用により減らすことが可能である．DBC は，別々のハードウェア・モジュールの集まりから構成されるので，既存のモジュールに故障が発見されると，別のモジュールを追加し，操作を切り換えることができる．システム中に故障が存在しえないようにするため，インタフェース・ロジックとバス結合は，すべて置き換えられるようになっている．置き換える必要のない唯一の要素は KP である．もし KP が故障すると，制御プロセッサが，KP の機能に対応したソフトウェアで，同じことを実行する．しかし処理は遅くなる．

4.2 DBC の操作

一般に DBC のすべての要素は，ここで説明する機能を実行するためにともに作動する．

1) 探索 (search)：この操作は，次の順序で行われる．まず，特別な並列転送ディスクあるいは伝統的なディスクの集まりから，1つのシリンダが共用記憶モジュールに一度に読み込まれる．つぎに，制御プロセッサが始動し，すべてのトラック・プロセッサ (TP) が探索条件に合致するレコードを乱呼出し記憶モジュールから探索する．この情報は TP のマイクロプロセッサとバッファを使用して，任意のブール式の探索引数を得る．探索されたレコードは，続く処理のためにホスト・コンピュータあるいはブロック記憶のどちらかに転送される．

2) 射影 (projection)：この操作は，選択されたレコードの一部分のみを取り出す．限定された情報の1つによって，結果が重複レコードになることがある．重複を排除する射影を望むなら，キー・プロセッサ (KP) が呼び出される．この場合，選ばれたレコードの引数列は，挿入モードで動作している KP に渡される．その引数列がすでに KP 内にあれば，そのレコードは重複しているため捨てられる．そうでなければ，新しい引数列が KP 中に格納され，レコードはホスト・コンピュータあるいはディスクへの出力として準備される．あるいは後に続く処理のためにブロック記憶に転送される．

3) 完全結合 (full join)：完全結合は，1つのリレーション (あるいはファイル) A のレコードと同じ属性値を持つ別のリレーション (あるいはファイル) B のレコードと結合する．完全結合処理はいくつかの段階を経て実行する．まず，A の各レコードに対し，ある TP が引数列を挿入モードで作動している KP に転送する．数値 k が KP より戻されると，レコードは対応する乱呼出し記憶中のリスト番号 k に登録される．A のレコードがすべて取り扱われた後に，B のレコードが処理される．この段階では，B のレコードからの各引数列が，照合モードで動作している KP に転送される．レコードの引数列が KP になければ，そのレコードは捨てられる．引数列が KP にあれば，結合するレコードを含むリスト番号 k がわかる．TP は，B のレコードをリスト k 中にある A の全レコードと結合し，レコード A の新しい集合を作成する．B のすべてのレコードをこの方法で処理して，完全結合は完了する．

4) 半結合 (implicit join)：半結合は，リレーション (あるいはファイル) B のレコードと共通な定義域を持つリレーション (あるいはファイル) A のレコードを選び出す．このとき，結合処理は不要である．したがって，半結合は完全結合より単純な操作である．半結合の第1段階では，ファイル B の引数列が KP に格納される．第2段階で，ファイル A からの引数列が照合モードで動作している KP により検査される．KP の値と合致したすべてのレコードが A からの出力となる．KP に対応する値を持たない A からのレコードは，単純に捨てられる．

5) 変更 (modification)：変更したいレコードのキーを，まず挿入モードの KP に挿入する．つぎに，変更されるレコードを含むシリンダをディスクあるいはブロック領域から RAM に読み込む．さらに，このシリンダ中のレコードのキーを，刻印モードの KP 中のキーと比較する．最後に，検索モードの KP で刻印されている項目を TP により得て，対応する変更を行う．未刻印のまま残される KP 中の項目は，検索されたシリンダ中にないレコードの変更要求か，あるいは存在しないレコードの変更要求の

どちらかである．前者の場合には，別のシリンダを読み込み，処理する．後者の場合には，使用者に誤り条件を知らせる．

6) 削除 (deletion)：この操作は，変更の場合と類似しており，刻印されている項目に対応したレコードを削除する．

7) 追加 (addition)：レコードをファイルに追加する最初の2段階の処理は，レコードの変更あるいは削除の処理と類似している．3段階目で，未刻印の項目が得られ，対応するレコードが追加される．KP中に未刻印のまま残されている項目は追加を要求されたレコードがすでにファイル中に存在していることを示しており，このレコードをファイルに挿入することは重複を引き起こすことになる．

8) 整列 (sort)：DBCは並列処理要素を持っているので，多くの型の並列な整列処理が可能である．1つの例は，各TPがまず共用の乱呼出し記憶モジュール(R)中の自分自身の内容について標準の整列あるいは併合を行う．その後，この記憶の低部分の内容と，隣接するTP記憶の高部分とを併合する．これら2つの操作は，ファイルが完全に整列するまで繰り返す．どんな場合も，並列な整列処理の効率改善は，少なくともTPの数 n のオーダでなければならない．

複数の結合処理や射影処理を含む集合の割算や，共通集合のようなもっと複雑な操作もまた可能である．複数の結合や射影を実行するとKPの記憶領域があふれるかもしれない．このような状態になれば，各TPは自分のファイルを終りまで処理し続けるが，KP中に記憶できないレコードは，オーバフロー・リストに置かれる．すべてのTPがこの処理を終了すると，KPの記憶領域はクリアし，オーバフロー・レコードが処理される．キー値あるいは属性列が大きすぎてKPにはいらない場合は，十分な大きさになるまで分割する．この場合KPは挿入モードになり，分割した数を要求元に知らせる．実際の値はKPが与えた数に対応するリスト上に置く．続く操作で実際の値が必要となると，その値を得るために対応リストを探索する．このようにキー・プロセッサを分割の目的で使用することもできる．

5. ソフトウェア

DBCのソフトウェアと対応するホスト・コンピュータのソフトウェアは，命令，データ，応答等の規約を経由して互いに連絡しあうソフトウェア・サブシステムのネットワークからなる．

このネットワークは次のような特徴を持っている．

1) 処理される命令，データ，応答を定義する外部インタフェース

2) 外部インタフェースが変更されない限り，被る変更が局所的にすむ内部処理ロジック

3) 外部インタフェースが変更されない限り，被る変更が局所的にすむ内部データ構造

4) データ辞書データベースに対する要求命令

5) 他のサブシステムを呼び出す要求命令

あるサブシステムは，それ自身入れ子型のサブシステムで構成されてもかまわない．

高水準のソフトウェア・サブシステムのネットワークを図4に示す．それらは次節以降で述べる要素からなっている．

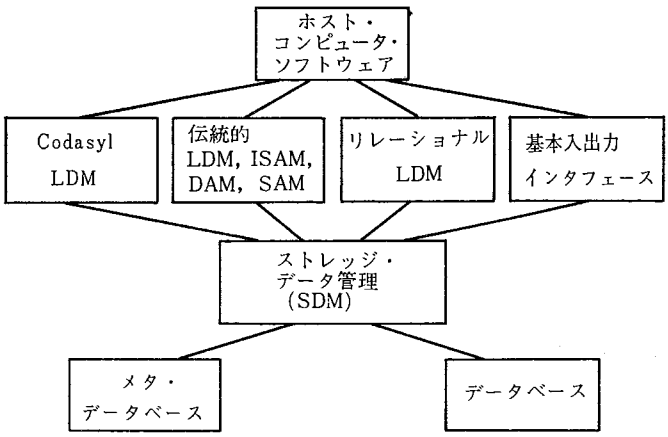


図 4 ソフトウェア構造
Fig. 4 Software structure

5.1 ホスト・コンピュータのソフトウェア

1) 使用者プログラム：これらは論理データ管理(LDM, logical data manager)に対するデータ操作言語とファイル入出力を実行する使用者のアプリケーション・プログラムである．これらは，データの格納と検索のためのデータ管理ソフトウェアを使って使用者のアプリケーションを処理する．

2) 一般ソフトウェア：このカテゴリのソフトウェアは，コンパイラ，ユーティリティおよびデータの格納や検索のために DBC を呼び出す実行ファンクション等を含んでいる．

3) データベース管理ソフトウェア：このカテゴリのソフトウェアは，種々の論理データ・モデルのデータ定義プロセッサや転送のためのファイル管理ソフトウェア，データベースの検証ユーティリティ，データベースの設計支援ユーティリティ，データ辞書プロセッサ等を含んでいる．

5.2 論理データ管理(LDM)

論理データ管理サブシステムは，特定のデータ・モデルを処理するシステムである．これにより任意の論理データ・モデルを処理することができる．現在，考慮しているデータ・モデルは，網状モデル(CODASYL, DMS 1100[8])，MISAM，直接編成，リレーショナル，OS 1100 基本入出力である．

階層モデルやその他のデータ・モデルについては，今後検討する．

LDM は，使用者プログラムにデータの論理的視野 (logical view) を与えるものであり，論理的 DML 命令を変換して，DBC で作動するストレッジ・データ管理 (SDM, storage data manager) サブシステムのためのデータ・ストレッジ言語(DSL)命令にする動きを持つ．

DSL は，レコードの集団を対象とするレコード・レベルの命令あるいは問合せである．データ・モデルとサブスキーマとの参照に基づき，LDM は使用者プログラムにデータを渡す前に，データを変換し写像する．LDM は SDM のレコードの集まりを要求し，内部的にそれらを処理する．

5.3 ストレッジ・データ管理(SDM)

ストレッジ・データ管理(SDM)サブシステムは，データベース・コンピュータ上で動く

ソフトウェアであり，データの格納や検索だけでなく，その他のデータ管理制御機能もつかさどる．

SDM は，次のことを実行するために，必要な制御プログラムと機能ルーチンからなっている．

1) 格納と検索
2) 領域管理
3) メタ・データベースの保守
4) 記憶媒体制御
5) 呼出し制御と機密保護
6) 完全性の管理
7) アカウンティング
8) 修復

SDM は複数のスキーマを管理できる．データベースが複数の DBC に広がっているところでは，複数の SDM が，完全性や修復処理のためにインタフェースをとる．この際，複数の SDM は分散データベース管理制御のために定められた規約に従う．

5.4 メタ・データベース

メタ・データベース(MDB)は，使用者データベースを定義しているデータを含むデータベースである．MDB は，データ管理ソフトウェアによりもっぱら使われ，保守される．これは，データ管理サブシステム間のインタフェースの一部であり，次のものを含んでいる．

1) データ辞書あるいはデータ・ディレクトリ
2) 論理スキーマ
3) ストレッジ・スキーマ
4) サブスキーマ

6. おわりに

Sperry Univac 社は，機能を追加しながら費用対効率を改善し，既存の使用者のデータベースとアプリケーションを支援したいと考えている．この考えに最も適している方法は元にした DBC の方法[7]であることが最初の調査で明らかになった．現在の使用者アプリケーションの分析と，今後の要求の調査により，データベース・コンピュータに対する機能要求をまとめた．提案されていた設計にこれらの要求を適用して，本稿で報告したアーキテクチャに改訂した．ここで報告したデータベース・コンピュータは，CODASYL 型あるいはリレーショナル型のデータベース管理システムの支援に必要なすべての機能を持つ．

現在，研究は第2段階にはいっている．複数の DBC アーキテクチャを実現するために必要な変更を行っている．また，DBC によるデータのクラスタリングに関する重要な領域を研究している．さらに，データベース・コンピュータの効率を解析的に確認するシミュレーション・モデルを開発している．これは汎用コンピュータ上での現データベース管理システムとの比較を行うためのものである．

この特殊目的のコンピュータは，現代技術社会において増大し続けるデータをうまく管理していくことに大きく貢献することになるだろう．

（システム・ソフトウェア開発部　原　　潔 訳）

参考文献 [1] E. F. Codd, "A Relational Model of Data for Large Shared Data Banks", *Communications of the ACM*, Vol. 13, No. 6, June 1970, pp. 377-387.
[2] O. H. Bray and H. A. Freeman, *Data Base Computers*, Lexington Books (D. C. Heath and Co.), Lexington, Mass., 1979.
[3] D. Leavitt, "Two-Year Project Pays Off: Back-End DBMS Succeeds", *Computerworld*, Feb. 20, 1978, p. 1.
[4] *Computer Weekly*, Sept. 13, 1979.
[5] S. Y. W. Su and G. J. Lipovski, "CASSM: A Cellular System for Very Large Data Bases", *Proceedings International Conference on Very Large Data Bases*, Sept. 1975, pp. 456-472.
[6] E. A. Ozkarahan, S. A. Schuster and K. C. Smith, "RAP—Associative Processor for Data Base Management", *AFIPS Conference Proceedings*, Vol. 44, NCC, June 1975, pp. 379-388.
[7] J. Banerjee and D. K. Hsiao, "DBC—A Database Computer for Very Large Databases" *IEEE Transactions on Computers*, C-28, No. 6, June 1979.
[8] Sperry Univac, *Data Management System (DMS 1100)*, UP-7907 Rev. 3, 1977.
UNIVAC シリーズ 1100 データベース管理システム DMS 1100 データ記述言語解説書スキーマ編, 481205620-1, Apr. 1981.
[9] O. H. Bray and H. A. Freeman, "Data Usage and the Data Base Processor", *Proceedings ACM '78*, Dec. 1978, pp. 234-240.
[10] D. D. Chamberlin and R. F. Boyce, "SEQUEL: A Structured English Query Language", *Proceedings ACM Workshop on Data Description, Access, and Control*, 1974, pp. 249-264.
[11] Sperry Univac, *Query Language Processor (QLP 1100)*, UP-8231 Rev. 1, 1977.
UNIVAC シリーズ 1100 データベース管理システム QLP 1100 解説書, 会話処理機能編 481205630-0, May 1979.
[12] Ampex Corp., *PTD-930x Parallel Transfer Drive*, Product Description 3308829-01, Oct. 1978.

執筆者紹介 John R. Jordan
1963年に数学で B. S., 1970年と 1974年にコンピュータ・サイエンスでそれぞれ M. S. と Ph. D を Iowa 州立大学より取得. 1963年から 1965年までは Ames 研究所で技術計算プログラミングに従事. 1965年と 1966年は Boeing 社でプログラミングに従事. 1966年から 1974年には Iowa 州立大学で教鞭をとる. 1974年に Sperry Univac 社に入社, 現在 Consulting Computer Scientist.

Harvey A. Freeman
1966年に Pennsylvania 大学より B. S. E. E. で, 1968年と 1970年に Illinois 大学より電気工学で, それぞれ M. S. と Ph. D. を取得. RCA 社から Sperry Univac 社に移り, データベース・コンピュータとローカル・ネットワークの研究に従事. その後, 分散処理の研究を行う. この間, Minnesota 大学のコンピュータ・サイエンス学部の准教授. 現在, Architecture Technology Corp. の技術担当副社長. IEEE, ACM 会員.

報告 オンライン援助への統一的アプローチ

A Unified Approach to Online Assistance

N. Relles, N. K. Sondheimer,
G. Ingargiola

Help! I need somebody.
Help! Not just anybody.
Help! I need someone. Help!
...Won't you please, please help me?
Help me! Help me!

John Lennon and Paul McCartney

要 約 多くの会話型コンピュータ・システムは，いくつかの形式の HELP プロセッサや援助コマンド(assistance command)を持っている．オンライン援助(online assistance)を効果的にするには，エンド・ユーザと援助提供者(assistance provider)，両者の必要性に立脚した枠組を明らかしなければならない．本稿はそのような枠組について述べる．すなわち，会話型コンピュータ・システムの普遍性と使いやすさは，適用分野によらない援助プロセッサと援助情報を持つ，高度に構造化されたデータベースに依存することを述べる．本稿の主な項目は次のとおりである．

1. 会話型ユーザが必要とする援助の型
2. わかりやすい援助機能を提供するために必要なデータ構造およびデータの関係
3. 効果的な援助形態を促進し，支援するソフトウェア・アーキテクチャ
4. オンライン援助を組み込み，それを維持していくために必要なプログラミング努力

その他，本稿では，オンライン援助機能を有効でかつ経済的なものとする，1つの方法を提案した．すなわち，ソフトウェア・ライフサイクルの各過程で求められる援助を統合する方法である．

Abstract Many interactive computer systems have some form of HELP or assistance commands. Effective online assistance requires a well-defined framework that addresses the needs of both the end-user and the assistance provider. This paper presents such a framework, whose generality and usefulness come from an application-independent assistance processor and a highly structured database of assistance information. Major considerations are (1) the types of assistance interactive users need, (2) the data structures and relationships required to provide comprehensive assistance, (3) software architectures that encourage and support effective forms of assistance, and (4) the programming effort required to include and maintain online assistance. To make online assistance effective and economically feasible, the paper proposes a way to integrate assistance into other phases of the software life cycle.

1. はじめに

使いやすさは，いまや会話型ソフトウェアの開発における最も重要な目標であると考えられている．会話型システムの典型的な使用者は，もはや問題解決をする人（エンド・ユ

This article is reprinted, with permission, from AFIPS Conference Proceedings (1981 National Computer Conference). pp. 383-388.

ーザ）とコンピュータとの仲介者としてのコンピュータの専門家ではない．システムは，むしろエンド・ユーザ自らによって使用されつつある．これらのユーザは特別な訓練やプログラミング技術を必要とせずにコンピュータと会話できることを望んでいる．これらの要望に応える1つの方法が，有効な参照情報，可能なアクションの記述，結果の説明，エラーの認識，リカバリ方法の指摘等を含むオンライン援助(online assistance)の方法である．

オンライン援助は，参照マニュアルやユーザ・ガイドによる伝統的な方法に比べて次の点で優れている．第1に，他の方法よりタイムリにかつ経済的に行える点である．これは，システムとユーザが物理的に分散していること，定期的な更新要求があることを前提としている．第2点は，ドキュメント上に書くことが困難かあるいは不可能な援助を，オンラインでは提供できる点にある．たとえばユーザはオンライン援助を用いて，容易にクロス・リファレンスの鎖をたどることができる．これをマニュアル上で行うには，物心両面での器用さが要求される．最後の利点として，オンライン援助を使うことによりシステムの改良に役立つことである．すなわち，システムの有用性を監視し，頻繁にユーザを当惑させているものは何かを容易に見いだせる点にある．

多くの商用システムは，いくつかのタイプの HELP プロセッサや援助コマンドを持っている[1-6]．しかし，これらのシステムは一般的に参照援助(マニュアル中で普通に見いだせるようなサマリや詳細説明の提供)にすぎない．実験的システムではメニュー選択，知的介入，ユーザ依存プロトコルおよび自然言語インタフェースのような形で，より会話的な援助が行えるようになってきている[7-12]．

効果的にオンライン援助を行うには，エンド・ユーザと援助提供者との要請に基づいた明確な枠組が必要である．本稿は，ソフトウェア・ライフサイクルとは別の要素として，オンライン援助を統合しようとして発達してきた枠組について述べている．システムの一般性と使いやすさには，アプリケーションに依存しない援助プロセッサ，および高度に構造化された援助情報のデータベースから得られる．その際，主に考慮すべき点が4つある．すなわち，会話型ユーザの必要としている援助の型，わかりやすい援助を提供するためのデータ構造とデータの関係，効果的な援助形態を促進し支援するソフトウェア・アーキテクチャ，そしてオンライン援助を組み込みそれを維持していくためのプログラミング努力である．

2. 問題点と要求

いくつかのオンライン会話型デバッギング・システムの存在にもかかわらず，ユーザはしばしば従来どおりの情報源(マニュアル，相談相手，端末機やその周囲の壁に張られたビラ等)に頼ることを余儀なくさせられている．その原因の1つに，指針やガイド・ラインの欠如がある．この欠如が，システムの実行と保守を簡単にするオンライン援助やソフトウェア・ツールの準備の遅延や，オンライン援助機能の効果的な使用法の妨げになっている．効果的に使うためには，オンライン援助は次のような特徴を持っていなければならない．

1) 頑健さ……広範囲(使用しているシステムについてのみならず関連システムを含めて)の質問に答える能力．たとえば，あるプログラムを編集している場合に，ユーザが編集手続き，編集されるプログラミング言語，あるいはオペレーティング・システムについて，同じ方法で質問できなければならない．
2) 融通性……ユーザの要求に応じたきめ細かなレベルで援助を与える能力．すなわ

ち，いくつかのパラグラフを通して見なければ，たった1つの必要な事実を見つけだすことができないのでは困る．簡潔な記述，あるいはそれに引き続くより詳細な説明が容易に得られなければならない．

3) 対話の継続性……ユーザの現在の状況に適合した援助を与える能力．たとえば，あるエラーが生じた時，エラーやその原因となった状態をユーザが確認しなくても，援助が得られなければならない．

4) サービスの非強要性……実行中のタスクを中断することなしに援助を要求する能力．多くの援助システムは独自のジョブ制御文として与えられている．援助を受けるために，現在のタスクを終らさせなければならないのでは繁雑なばかりでなく，作業環境を保存したり元に戻したりすることにより，効率が落ちる．実際に，中断できないプログラムはオンライン援助の恩恵を受けられずにいる．

5) 一貫性……同じ操作形態で，比較的大きなシステムを構成しているすべての会話型プログラムを一貫して援助すること．この特性を実現させる最良の方法は，オペレーティング・システムのレベルで援助機能をまとめることである．しかしながら，現実の援助機能はそのような一貫性が欠如している．その最大の理由は，援助機能が，大きなシステムを構成するアプリケーション・プログラムのためではなく，オペレーティング・システムのコマンド言語のために用意されることが多いことである．その結果，アプリケーション開発者に残された道は，オンライン援助をまったく排除するか，一貫性のない独自のオンライン援助を行うか，あるいはオペレーティング・システムが提供する援助機能に最もよく似たものを作らざるをえないかのいずれかであった．

6) 協調性……援助に対するある要求に続いて，ある操作を自動的に行う能力．たとえば，ユーザがあるファイルの属性の変更方法を質問し，その方法が提示された後，ユーザが「OK！ そのとおりに実行せよ」と指示すれば，実行に移されるような機能を提供すべきである．

これらの特性が重要なことは，あまり効果のない援助システムを使ったことのある人(あるいは他の人が使っているのを見たことのある人)なら，だれにでも明白である．もしオンラインで自分の質問の答えを得ようとしてうまくいかないと，ユーザはすぐにより確かな情報源に戻り，援助システムは使われなくなる．これまでの経験から，援助システムがユーザに歓迎されるか否かは上の6つの特性をどの程度そなえているかにかかっているといえる[8]．

3. 統一的枠組

3.1 外　観

上記の問題点と要請を踏まえた上で，オペレーティング・システムは1つの援助データベース(assistance database, ADB)と1つの援助プロセッサ(assistance processor, AP)を含むことを提案する．援助データベースは，高度に構造化されたデータベースで，アプリケーション・プログラムやオペレーティング・システム自身のコマンド，概念，そして機能を表すのに使われる．援助プロセッサはアプリケーションに依存しないプロセッサで，援助データベースに対する援助の問合せを解釈する．1台のプロセッサで一貫性の要求を満たすことができる．また，適切なソフトウェア・インタフェースを持つことによって，そのプロセッサは対話の継続性とともに端的にユーザのペースで作業が中断できて協調的

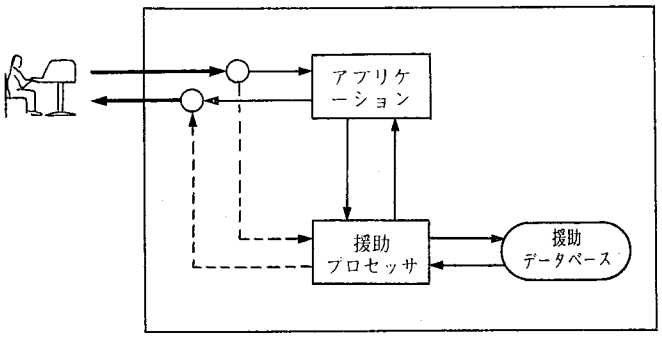


図1 援　助　環　境
Fig. 1 The assistance environment

となりうる．適切に構造化されたデータベースを作れば，援助情報を強力で融通性のあるものとすることができる．

図1はユーザ，アプリケーション・プログラム，援助プロセッサ等の相互のつながりを示している．ユーザが入力するコマンドはアプリケーション・プログラムあるいは援助プロセッサのいずれかに渡される．援助要求はそのアプリケーションにそのままの形で渡される．援助プロセッサは，適宜あいまいさを解消したり，冗長となりうる説明を回避するために，メニュー選択や追加情報の要求を行うことがある．アプリケーション・プログラムと援助プロセッサは，次のような目的のために，メッセージをやりとりすることがある．

1) ユーザの状態の確認．たとえば，生成された目的コードや使用されるコマンドの流れや最後に発生したエラー等．
2) 援助プロセッサの表示説明，省略時の値，回復手順等の入手．
3) ユーザの援助要求についての情報の記録．
4) ある援助要求に対応する特定の処理をアプリケーション・プログラムにさせること（すなわち，「援助の返答どおりに実行せよ」）．

援助データベースは図2に示すような3つの大きな部分からなっている．すなわち固定援助データ，変動援助データ，長期（監視）データである．

固定援助データはパラメタで主として援助管理者（assistance administrator）によって生成され保存される不変データを含んでいる．援助管理者といっても，人間としてではな

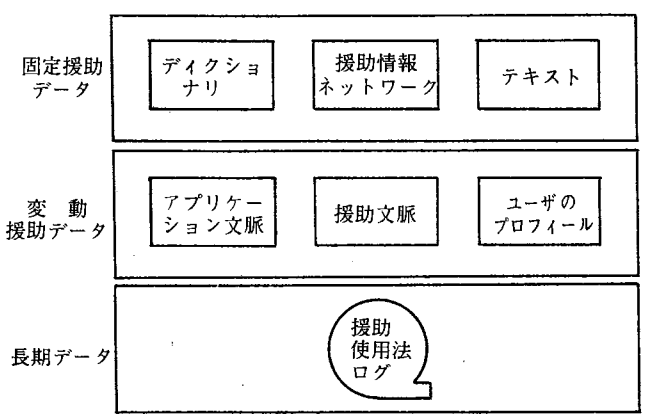


図2 援助データベース（ADB）
Fig. 2 The assistance database

く，役割としてとらえなければならない．それは1人でもよいし，援助提供者の集団でもよいし，プログラマ，テクニカル・ライタ，アプリケーションの専門家の集団であってもよい．固定援助データとは，その大部分は，標準的にユーザ・ガイドや参照マニュアルに見いだされる情報に当たる．固定データのディクショナリ部は援助の問合せに現れる用語を含んでいる．援助情報のネットワークは，あるアプリケーションに関する概念やその相互関係を表現している．このような概念や相互関係はさらに，個人指導用の説明文，説明のメッセージ，コマンドの記述，あるいは例題，といったテキスト部を伴っている．

変動援助データは，各ユーザ・セッションの始めでは通常空であり，アプリケーションやその援助情報とユーザとの間の対話状態を表現している．この文脈上の情報により援助プロセッサは，現在の問合せ，1つ前の問合せさらにアプリケーションとの間で行われた対話から糸口を見つけて，望む情報に焦点を合わせることが可能となる．図2で示されるように，変動援助データはまたユーザの経験レベル，使用頻度，そして対話のスタイル等のプロフィールも含んでいる．アプリケーション・プログラムと援助プロセッサはともに，構造の一部が援助データの不変なネットワークに連結している変動援助データを更新することができる．

長期データは答えられなかった援助問合せ，参照メッセージやネットワークのノードの頻度表，そしてその他の援助プロセッサの使い方についての統計表からなっている．この情報は，援助プロセッサのためというよりは援助管理者のためのもので，援助データベースを維持改善するのに使用される．

3.2 援助データベース

効果的な援助をするには，援助ネットワークにおける情報の豊かな構造が不可欠である．知識の表現に関する研究が，われわれにそのようなネットワークを構築する基礎を与えてくれたと信じる[13-15]．この節ではわれわれが開発してきたネットワーク表示の構造を簡略に示してみる．

援助の最も基本的な形は，互いに無関係なノードからなるデータベースによって与えられる．各ノードは，コマンド，パラメタ，あるいは定義のような概念を表し，テキストのいくつかの部分と連結されている．図3では，概念とそれらを表示するテキストの断片との連結関係を TEXT というラベルを付けたアークで表している．この単純な構造により，ユーザが知りたい概念に関する援助情報を表すことができる．たとえば，乱編成ファイルに関する情報を要求すると，援助プロセッサは，そのノードに連結されたテキストを表示する．

もっと多様な援助を提供するように，各ノードを階層的に連結することができる．既存のいくつかの HELP システムは，援助データベースの中にそのような階層構造を取り入れている[1,3,4,6,7]．たとえば Wisconsin 大学における@@HELP システムは，システム・ライブラリ内の各プログラムに対して，1つのノードを持っている．さらに，そのようなノードは使用可能なコマンドの形式を説明する複数の下位のノードを各々持っている．この形式を説明する各ノードは，パラメタやオプションを細かく表す複数のさらに下位のノードを持っている[6]．

現状の援助データベースでも階層関係を改良することによって，援助システムの重要な改良が行われる．すなわち，別々の階層関係を持つことによって，ユーザはすみやかに求める情報に焦点を合わすことができる．1つの概念に関するすべての下位情報を表示することなく，ユーザは独自の方法で関係する情報部分だけが得られる．明確に定義された関

係の固定セットは，またそのネットワーク中で一貫性を持たせ，援助情報の回復を指示する．連想ネットワークの中で，これらの階層はそれぞれ異なったラベルを付けたアークで表される．

もっとも役立つ階層関係の1つは IS-A 関係である．IS-A 関係はいくつかの概念を，より一般的な包含した概念に結び付ける．たとえば図3で IS-A 関係は，すべての RANDOM FILE は FILE であり，すべての SEQUENTIAL FILE も同様であることを示している．サブセットの関係として考えると，IS-A 関係は多くの他のタイプの概念（コマンド，パラメタ，許容値等）を表現するのに使える．このように関係というものを用いることにより，ユーザは一般的解説の異なるレベルでの説明を得ることが可能となる．

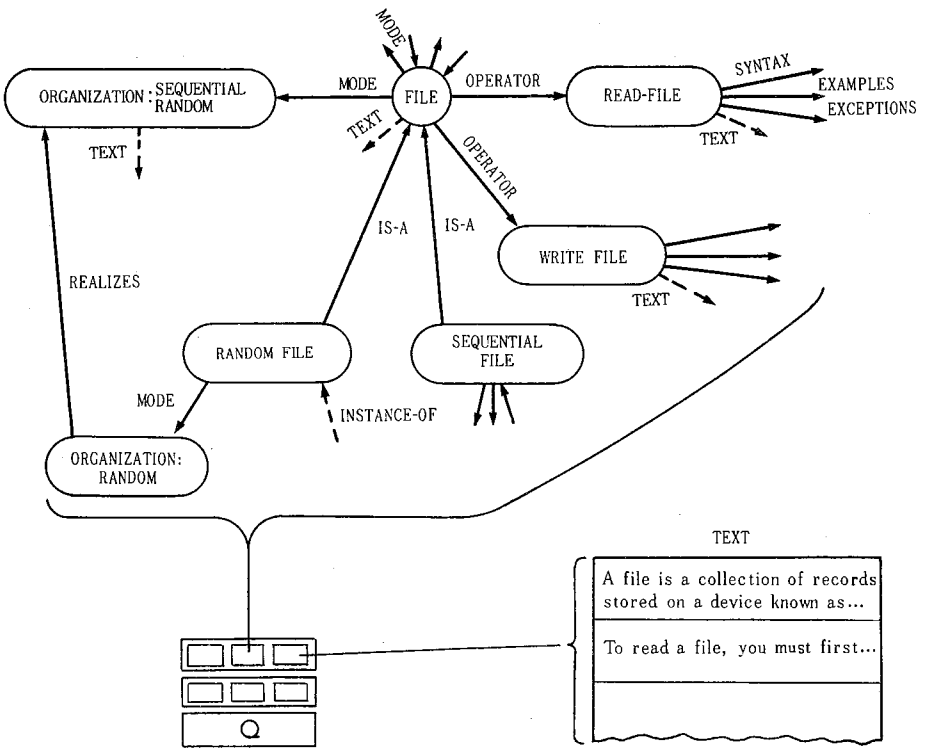


図3 援助データの見本

Fig. 3 Sample of assistance data

IS-A 階層(関係)に伴う他の2つの関係に MODE と REALIZES がある．MODE 関係はエンティティの属性を導入するのに用いる．これらのラベルはこれらの属性の具体化方法の制限を示している．たとえば，図3で MODE 関係は FILE の ORGANIZATION 属性の値として SEQUENTIAL か RANDOM が許されることを示している．REALIZES 関係は，あるエンティティの属性が，より一般化された同じ型の属性の上で，どのような範囲で選択できるかという制限を示している．図3の例では，RANDOM FILE の ORGANIZATION 属性は一般的なファイル編成の具体化として示されている．これらの関係によって，ユーザは，エンティティの属性，それらの許容値，および関連エンティティとの間の相違を説明することができる．また，REALIZES 関係がなければ，性質の引継ぎが行われる．たとえば，RANDOM FILE では，そのファイルの概念が持つことのできるすべての他の MODE の属性と値を許容するとみなされる．

図3には，他の階層関係も示されている．OPERATOR 関係はあるエンティティ上で作用可能なコマンドを定める．たとえば，図3の FILE では READ-FILE コマンドと WRITE-FILE コマンドが許される．SYNTAX 関係はコマンドの構文の説明本文への索引として使われる．EXAMPLES は有益な例をアクセスするため，EXCEPTIONS はありうる例外的行動を表すために，それぞれ使われる．

INSTANCE-OF 関係は，援助ネットワークにおけるもう1つの重要な改良点である．ほとんどの HELP システムにはないが，この関係はユーザの実際の対象を，援助ネットワークの中で表現された一般的で抽象的な概念に対応させるのに使う．これは親子組と成員の関係と考えることができる．たとえば，図3では RANDOM FILE ノードは，援助データベースの変動部分にある例題のファイルに INSTANCE-OF 関係を通して連結されている．それゆえ，ユーザの特定の乱編成ファイルに対して援助を受けることが可能となる．

オンライン援助がマニュアルの巻末にある索引よりもっと効果的であるためには，普通の木構造や格子構造以上のものを導入しなければならない．関係する概念の間の自由な相互の参照を可能にする構造でなければならない．たとえば図3の全ノードは，異なったプログラミング環境に対するデータベース中の同様のノードと関連づけられるであろう．同様の内部連絡はエラー条件を表現する場合にも必要である．各エラー状態は単一のノードとして表現される．このノードに結合されたテキストはエラーを記述し，このノードに付けられた他の関係はエラーの原因，予防法および訂正を定めるのに使われる．たとえば，乱編成ファイル上に書こうとしているときに発生したエラー状態は図3の RANDOM FILE と WRITE FILE の両方のノードに結合されなければならない．

上に述べたようなタイプの援助ネットワークは，多くの異なるシステムの機能を表現するのに使える．ここで述べた関係はすべてをつくしているわけではないが，これらの関係は，現在の適用システムで提供されている援助の形をほとんど網羅し，まだどこでも実現されていない援助の形を提供するにも十分であろう．そのような構造に基づく援助は強力で一貫しており，融通が効き，そして対話をうまく継続できる．

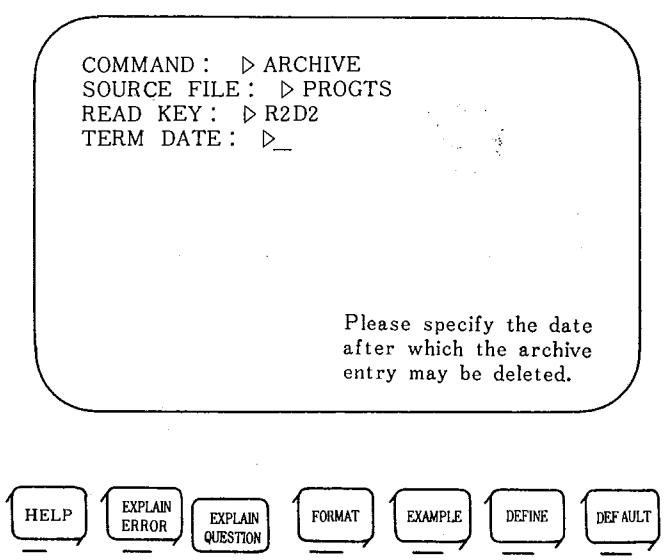


図 4　対話をうまく継続させるオンライン・デバッギング・システムの例
Fig. 4　Context sensitive online aids

3.3 ユーザの観点

援助ネットワークは援助プロセッサに用いられ，多くの異なった形態のオンライン援助ができる．ネットワークをたどったり連結してある援助情報を表示したりすることは，ユーザ，アプリケーション・プログラム，援助プロセッサ，援助データベース間のやりとりによって管理される．ネットワークのデータ上にユーザの“自由領域”をとることもできるが，ユーザの現在の状態を反映しているデータベース中に，数ポジションを設定するほうが有用である．

図4はユーザ側から見た援助環境の一例を示している．作動中はいつでも，アクセス可能な援助情報にはラベルを付けたファンクション・キーが連結している．これらのキーを繰り返し押すことにより，ユーザは現在の状態に関係する詳細な情報をつぎつぎに得ることができる．

4. 援助をソフトウェア・ライフサイクルに組み込む方法

援助の枠組の有効性は，それが効果的に使われることはもちろん，それが使用されるということをも保証しない．良いドキュメンテーションと同様に，良い援助システムは簡単に手に入るものではなく，かなりの資源を必要とする．使いやすくするための費用はどのくらいになるのだろうか．最近の推定値がPROMISシステム[7]から得られる．その報告によると，1人年あたり約500の割合でメニュー・スクリーンを作っている．他の会話型システム[8]では，援助メッセージの作文と保守にかかる時間はシステム総作成時間の1/3以上となっている．もし，いかなるソフトウェア製品の開発においても援助情報を含めるとすれば，明らかに援助情報の開発・維持費は，もっと経済的でなければならない．

また，オンライン援助情報が書かれたドキュメンテーションと一致していなければならない．明らかに書かれたドキュメンテーションとオンラインによるドキュメンテーションの双方が正しくなければならないのはいうまでもない．すなわち，1つのシステムにおける機能要求と仕様は一致していなければならない．したがって，各々のアクティビティ(要求分析，システム仕様，高度設計，ドキュメンテーション，援助情報の開発)で，情報が重複していては不経済である．

これらのアクティビティを効率的に1つにまとめるためには，アクティビティ各々のデータベースをデータの，より一般化された結合のサブセット(ソフトウェア・プロダク

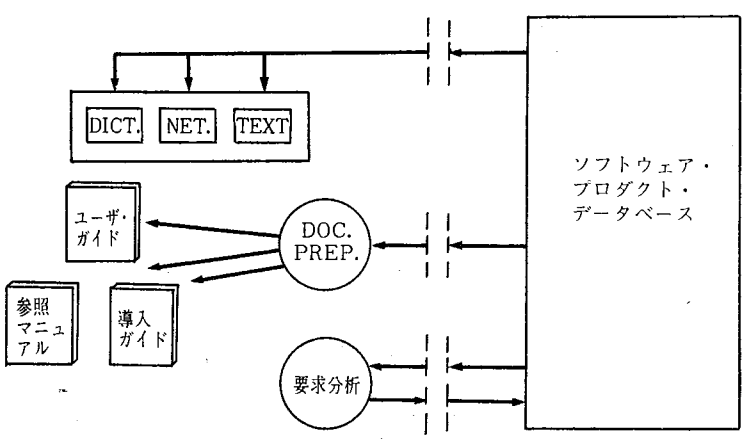


図5 ソフトウェア・ライフ・サイクルにおける援助
Fig. 5 Assistance in the software life cycle

ト・データベースと呼ぶ)としなければならない．なぜならこのようにすると，援助情報，ドキュメンテーション，および設計仕様における重複を1つにまとめることが可能と思われるからである．要求分析とドキュメント準備の統合についてはすでに実現している[16]．オンライン援助機能とドキュメントの準備とを統合することについても進展が見られる[17]．援助情報とこれらのアクティビティを全体的にまとめた概念図を図5に示す．対になっている破線はソフトウェア・プロダクト・データベースとのインタフェースを表している．あるインタフェースはソフトウェア・プロダクト・データベースの一部を要求分析やシステム設計に使用可能な構造に変換する．このインタフェースは，また逆に要求と設計のデータベースからソフトウェア・プロダクト・データベースへの対応する変換も行う．同様に，1つのドキュメント準備システムにおいて使われるデータベースのサブセットを作り，維持していくためのインタフェースも存在する．その他に，援助データベースを構成するディクショナリ，ネットワークおよびテキストを作るためにデータベースから情報を抽出するインタフェースがある．

ソフトウェア・プロダクト・データベースはオンライン援助に対する中心的構造をなす．ソフトウェア・プロダクト・データベースは簡単にアクセスできる形で，必要な情報を提供しなければならない．これは，援助要求に対して人が納得するような応答を組み立てるために必要となる．そして，すでに述べたようにソフトウェア・プロダクト・データベースは，ソフトウェア・ライフサイクルの他の過程との間で必要な情報と一致していなければならない．このデータベースの定義は，相互に関連するユニットの一連のものとして発展してきた．その各構成要素は抽象的考えや具体的対象を表している．われわれは，そのような構成要素と関係とを“module”という考えで与える Ada[18] のような言語にとくに興味を覚える．モジュールはタスク，抽象データ型そして関連する宣言のライブラリを表現するのに使うことができる．そのようなモジュール適正に拡張することによって，要求分析，高級設計仕様，援助情報等をまとめる基礎が得られる．

次に示すプログラム例は，ファイルについての援助情報が Ada のような言語でどのように表現されるかを示している．

```
PACKAGE  [file;
              TEXT "A data structure for storing and retrieving" &
                   "objects to and from secondary storage"] is
  IS-A data-structure;
  SYNONYM dataset, data-set;
  RELATED secondary storage, peripheral;
    ⋮
  TYPE kind-of-organization is (
        [sequential;
        TEXT "A form of organization that allows only" &
             "sequential access to records"],
        [random;
        TEXT "a form of organization that allows both" &
             "sequential and random access to the" &
             "records of a file"]);
  organization: CONSTANT kind-of-organization := random;
    ⋮
```

図3は同じ情報のネットワーク表現である．PACKAGE モジュールは1つのファイルの概念を与えている．IS-A, SYNONYM, そして RELATED 句はファイルが一種のデータ構造であることを示しており，1つのファイルがまた “dataset” あるいは “data-set” として認められること，補助記憶装置と周辺装置が関連していることを示している．TYPE 句はあるファイルの属性とその編成を表している．この句は1つのファイルが順編成か乱編成であること，その編成が1つのファイルの一生を通して一定であること，とくに編成を指定しない場合，乱編成であることを示している．TEXT 句のいくつかは，それぞれの概念に対する説明をしている．“record”, “data structure”, “secondary storage” 等の考えの記述は同じ構造単位中に表れるであろう．

5. おわりに

オンライン援助機能は，多くの商用コンピュータ・システムでは常識となっているが，しばしば使用上の制限と煩わしさを伴う．今日，ユーザはサービスが押しつけがましくなく，非常に頑健で融通性があり，対話をうまく継続できてかつ一貫性のあるオンライン援助を必要としている．また同時に，オンライン援助機能へ入り込む複雑な情報を簡単に，かつ経済的に保守できなければならない．さらに効果的なオンライン援助を得るのに，特別な技術が必要であってはならない．

本稿は，オンライン援助を提供することにより発達させられる1つの枠組を紹介した．ただし，そのオンライン援助は援助プロセッサを通してアクセス可能な援助情報の関連ネットワックに基づいている．ネットワークの一般化した生成構造によって，援助をアプリケーションの広い範囲に対して与え，また経験者にも未経験者にも利用できるようにした．援助プロセッサは，情報に一貫性があり，容易にかつユーザの現在の環境に特有の表現で情報が得られるように，オペレーティング・システムの不可欠な要素でなければならない．援助情報の生成や保守を単純化するために，中央集中的なデータベースをソフトウェア開発のいくつかの過程(要求分析，高度設計仕様，ドキュメンテーション)で使用できる．そのようなデータベースの情報を提供する効果的な方法として，Ada の拡張と変更を述べた．

現在の技術でも多くのオンライン援助の改良を行うことができるが，ここで述べたような多くの議論はさらに研究に役立つであろう．連想ネットワークの一般性と使いやすさは，適度に完成していてそれほど複雑でない関係の集合に依存する．このような最適セットが容易に得られるか否かは，いくつかの会話型セッティングでユーザの要求する援助の種類の研究成果に依存する．われわれは，またユーザの資質について，そしてユーザの経験が深くなるにつれてユーザの要求がどのように変化するかについて，もっと知る必要がある．知的“援助”やシステム開始“援助”の準備にもまた，経験や使用頻度，プログラミングのスキル等，ユーザの特質を十分理解していなくてはならない．現在，そのための知識が，実験により，いくらか得られてきている[19-20]．援助機能の利用状況が単に監視するだけでもシステムの改良に役立つ見方が得られる．他の開発作業を伴いながら援助データベースを組み立て，それらを統合することは複雑な仕事である．たゆまぬ研究と試作システムの実現を通してはじめて，その仕事がいかに複雑であるか，どのような改良で使いやすくなるかがわかる．

(システム・ソフトウェア保守部　元山 裕之 訳)

参考文献 [1] *SPERRY UNIVAC 1100 Series Conversational Time Sharing (CTS) System: Programmer Reference*; UP-7940, Blue Bell, PA: Sperry Univac Computer Systems, 1977.
[2] Holg, Chloe, "*The Joy of TENEX and TOPS-20...in Two parts*," University of Southern California: Information Sciences Institute, Technical Report ISI/TM 79-15, January 1979.
[3] *Interactive Facility Version 1 Reference Manual, CDC Operating System NOS 1*, St. Paul: Control Data Corporation, 1978.
[4] K. Thompson and D. M. Ritchie, *UNIX Programmer's Manual: Sixth Edition* Murray Hill, NJ: Bell Laboratories, 1976.
[5] *IBM System/38 Technical Developments*, ISBN 0-933186-00-2, IBM Corporation, 1978.
[6] J. Anderson, "@@HELP: Online Documentation System," in *Technical Papers, USE Spring Conference*, 1979, Bladensburg, MD: USE, Inc., pp. 215-235.
[7] G. Robertson, A. Newell, and K. Ramakrishna, *ZOG: A Man-Machine Communication Philosophy*, Pittsburgh, PA: Carnegie-Mellon University, Dept. of Computer Science, August 1977.
[8] N. Relles, *The Design and Implementation of User-Oriented Systems*, Computer Sciences Technical Report # 357, University of Wisconsin-Madison, July 1979.
[9] R. Roberts, "HELP-A Question Answering System," in *Proceedings, Fall Joint Computer Conference*, 1970, pp. 547-554.
[10] S. C. Shapiro and C. K. Stanley, "Interactive Consulting via Natural Language," *Communications of the ACM*, 18: 8, 1975, pp. 459-462.
[11] W. Ash, R. Bobrow, M. Grignetti and A. Hartley, *Intelligent On-line Assistant and Tutor System*, Technical Report No. 3607, Bolt Beranek and Newman, Inc., January 1977.
[12] R. R. Burton and S. B. John, *An Investigation of Computer Coaching for Informal Learning Activities*. BBN Report No. 3914, ICAI Report No. 12, Cambridge, Massachusetts: Bolt Beranek and Newman, Inc., August 1978.
[13] S. Fahlman, *NETL: A System for Representing and Using Real-World Knowledge*, MIT Press, 1979.
[14] N. V. Findler, *Associative Networks: Representation and Use of Knowledge by Computers*, New York: Academic Press, 1979.
[15] R. J. Brachman, *A Structural Paradigm for Representing Knowledge*, Ph. D. Dissertation, Harvard University, 1977.
[16] S. Funk, "Putting PSL/PSA to Work," in *Technical Papers, USE Spring Conference*, Bladensburg, MD: USE Inc., 1979, pp. 57-78.
[17] L. A. Price, *Representing Text Structure for Automatic Processing*, Computer Science Technical Report # 324, Madison: Computer Sciences Department, University of Wisconsin-Madison, May 1978.
[18] J. D. Ichbiah, "Preliminary Ada Reference Manual," in *SIGPLAN Notices*, 14: 6, June 1979.
[19] B. Shneiderman, *Software Psychology: Human Factors in Computer and Information Systems*, Winthrop Publishers, 1980.
[20] L. H. Miller, "A Study in Man-Machine Interaction," in *Proceedings National Computer Conference*, 1977, pp. 409-421.

執筆者紹介 Nathan Relles

Wisconsin 大学においてコンピュータ・サイエンスで BA (1968年), MS (1973年), Ph. D (1979年) をそれぞれ取得. 1968年から1971年まで Sperry Univac 社の Defence Systems Division に所属, また, Lufthansa 航空の CAI システム構築に参画. 大学院時代は大学のコンピュータ・センターに籍をおき, いくつかのデータベース管理システムの開発・保守に従事. 1979年に Software Research Department に加わる. 現在は Human-Machine Interaction グループのマネージャであり, オンライン援助, ヒューマン・ファクタおよび自然言語処理について研究, 現在に至る. ACM および IEEE コンピュータ学会の会員. Temple 大学の准教授. 最近, オンライン援助で

全国的規模の組織を作った.

Norman K. Sondheimer
1968年 Carnegie-Mellon 大学において，数学および英語学でBSを取得．Wisconsin大学において，コンピュータ・サイエンスで1970年にMSを，1975年にPh. Dをそれぞれ取得．現在の研究分野は自然言語理解およびオンライン援助である．自然言語によるデータベースのアクセスを目標とするプロジェクトを主宰している．Sperry Univac社入社以前は，Ohio州立大学の学部に籍を置き，Bell社のコンサルタントであった．現在，Temple大学および Delaware 大学の准教授，Computational Linguistics 学会会長.

Giorgio Ingargiola
1967年に Pennsylvania 大学でコンピュータ・サイエンスのPh. D を取得．1968年から1975年まで California 工科大学で教鞭をとる．1976年より，Sperry Research Canter に所属，現在，Temple 大学の准教授．プログラミング言語，オペレーティング・システム，オペレーション・リサーチについての著作がある．現在は知識表現の研究に従事.

論文

コンピュータ・システムの構造的モデル化技法と性能評価

Structured Modeling and Performance Evaluation of Computer Systems

B. S. H. Wang

要 約 新規システムにしろ既存システムの変更にしろ，そのシステムの性能を予測する際に，コンピュータ・システムのモデルが重要な手段となる．近年，コンピュータ・システムのモデル化やその性能評価に関する種々の解析的技法がめざましく発展している．待ち行列網モデル (queueing network model) は，そうしたツールの1つである．これまでにも多くのモデルが開発され，コンピュータ・システムおよびオペレーティング・システムの様々な要素の性能が予測できるようになった．

本稿では，一般システムの観点から統一的なモデル化と評価技法を採用する試みを示す．その試みとは次のようなものである．

1. 複雑な全体システムのモデルを複数のサブモデルの集合に分解することによって，複雑さを軽減する．そのようなサブモデルは，ほぼ完全な分解可能性があるという意味で，管理可能な程度の複雑さになる．
2. 対象システム（多くのソフトウェア・モジュールを伴った階層化されたオペレーティング・システム，互いに結合しているハードウェア資源，あるいはその両者の組合せ，のいずれか）を，水平方向には結合したシステム要素で構成され垂直方向には階層的に構成されたシステムにモデル化して，精確さを達成する．
3. モデル化技法と解析方法とを統合し，全体システムに対しては巨視的な分析を行い，それぞれのサブシステムに対しては微視的な分析を行う．実際的な事例によってこのアプローチを述べる．

Abstract Models of computer systems constitute a very important means of investigating computer system performance, whether for predicting the performance of proposed new systems or proposed changes to existing systems. In recent years, noteworthy progress has been achieved in development of various analytic techniques pertinent to computer system modeling and performance evaluation; queueing network models are one such tool. Many successful models have been developed which enable the performance of different components of computers and operating systems to be predicted.

From the general system's point of view, this paper presents an attempt to adopt a unified modeling and evaluation strategy, which will:

1. Reduce the complexity by decomposing a complex total system model into a set of submodels. Such submodels are of manageable complexity in the sense of near-complete decomposability.
2. Achieve accuracy by matching the target system structure (whether it is a layered operating system with many software modules, interconnected hardware resources, or a combination of both) with a model structured vertically in hierarchical levels as well as horizontally in interconnected system components.
3. Integrate fragments of modeling techniques and method of analysis to perform microscopic analysis of a total system and microscopic analysis of subsystems. Practical examples are given to illustrate such an approach.

This article is reprinted, with permission from proceedings of CMG XI International Conference 1980, pp. 1-5.

1. はじめに

新しいコンピュータ・システムを計画・設計したり，現存のコンピュータ・システムを評価し機器構成を変更して性能向上を図るときに，個々の選択枝が与える全体のシステム性能に及ぼす影響を予測することがある．そのときには，何らかの効果的なツールが必要となる．これらのツールは，システムの生産性に大きく影響する要因を他の要因から区別できなければならない．現在，コンピュータ・システムの性能を評価し，予測するために採用されている主なアプローチが2種類ある．その1つはシミュレーション・モデルであり，もう1つは解析モデルである．一般的に，モデル化するシステムの姿は，解析モデルよりもシミュレーション・モデルによるほうが，忠実かつ詳細に表現できる．しかし，シミュレーションは計算時間が長いため費用が高く，計算結果の解釈もむずかしい．一方，解析モデルは，最近のモデル化技法の進展によって，より強力で豊富なものとなりつつある．

本稿では，全体システムの観点から，統一したアプローチを採用する．このようなアプローチにより，解析モデルの新しい技法を統合し，それらの技法を新しいコンピュータ・システムの計画・設計に適用することが可能となる．

Jackson 型待ち行列網モデルによって，伝統的な Markov 待ち行列網理論に基づく厳密解法が可能となった．この待ち行列網モデルは，Baskett ら[1]によって一般化され拡張された．この結果，異なったスケジューリングの方式，ジョブのクラス分け，指数分布でないサービス時間を含むようになった．これらの待ち行列網モデルについて数年間にわたって研究が続けられたが，得られた結果は常に積形の解であった．これらの待ち行列網は，扱いやすい(tractable)待ち行列網[2]として特徴付けられる．しかし，その解法が適用できるジョブの振舞いや資源の型には制限が設けられていた．より現実的でより複雑なモデルを扱うためには，やはり近似解法を用いなければならない．本稿で考える主な近似解法は，分解法とサブネットワーク倒潰法 (subnetwork collapse) または流量等価システムズ法(flow-equivalent systems)である．これら近似解法を，複雑なコンピュータ・システムの振舞いを分析するために，階層的に構造化するモデル化の手続きと Buzen の操作解析技法とを結合する際に用いる．分析を進めていく上で，分解(decomposition)をしないと，これらのすべての技法が相補完するようには統合できない．

2. モデルの構造

バッチ型でも会話型でもサービスできる多重プログラミングの時分割システムがある．これらシステムに用いられるモデル化技法を考えよう．最新のコンピュータの多くは，この種のシステムのモデルには多くの成功事例があるので，適切な例となるであろう．さらに重要なことは，取り上げた型のシステムのモデルから役立つ洞察が得られた点である．すなわち，現存する多重タスキング・多重プログラミングに基づいたシステム，たとえば Sperry Univac 社の VORTEX システム（日本では UNIVAC V 77 シリーズ SUMMIT BASIC)を効率的な最新のコンピュータ・システムに発展させる方法に役立つことがわかった．

前述の会話型時分割システムの概念図は図1のようになる．点線で囲まれた部分は1つのサブシステムであり，分解の対象となる，中央サブシステムの多重プログラミング・モデルである．システム内では N_T 個の端末からセッションが開始(ログオン)されており，それぞれの端末は思考状態と待ち状態とを交互にとるユーザによって占有されている．思

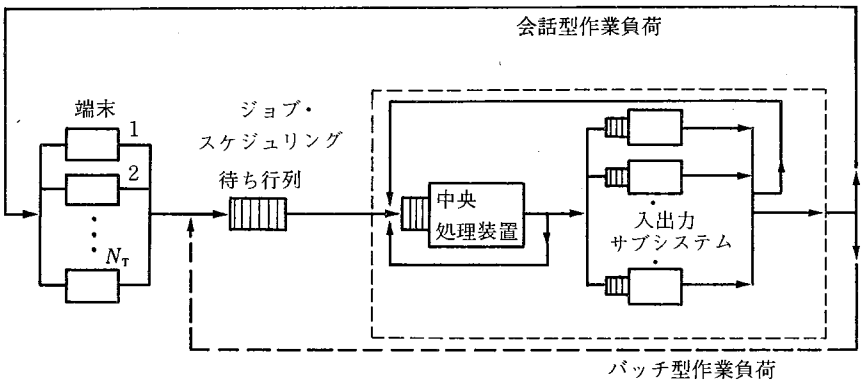


図 1　多重プログラミング時分割システムのモデル
Fig. 1　A multiprogrammed time-sharing system model

考状態の間，ユーザは次にどのようなジョブを投入するかについて考えており，中央サブシステムはそのユーザの仕事を実行していない．ジョブが投入されると，中央サブシステムがそのユーザのジョブを完了するまで，ユーザは待ち状態になる．会話型作業負荷は，N_T 個の会話型端末によるジョブである．バッチ型作業負荷は，カード読取装置のような入力機器や遠隔ジョブ投入 (RJE) 機器から投入されたジョブからなる．

モデル化技法の見地からは，図1のようなコンピュータ・システムは2段階の階層モデルによって表現できる．サブモデルがほぼ完全に分解できるため (Courtois[3] が説いた意味で) には，高位レベルの事象間の時間単位が低位レベルの時間単位よりも十分に大きいことが基本となる．いいかえれば，外部との相互作用の事象間隔中にサブシステム内の状態変化の数が十分に大きいという条件である．この時に，システム全体としての振舞いは，各サブシステムの状態の変化によって生じる影響を抑制できる．「オンラインの振舞いとオフライン振舞いは等しい」という操作概念に基づけば，このようなサブシステムはオフラインで考えることができる．この2段階層化モデルは，図2のように構造化される．低位レベルのモデル (閉じた中央供給器モデル) において，中央処理装置は中央供給器に対応している．入出力サブシステムは，スワッピング，使用者入出力，あるいはページングに用いられるディスクまたはドラム・システムからなる．この中央サブシステムは，記憶装置の制約を持つ多重プログラミング・レベル N と2つのクラスのジョブ (1個のバッチ型ジョブと数多くの会話型ジョブ) を持つ．バックログの下で動くバッチ処理システムは，実行中のジョブの処理を終えたら，直ちに次の新しいジョブが投入されることを保証している．

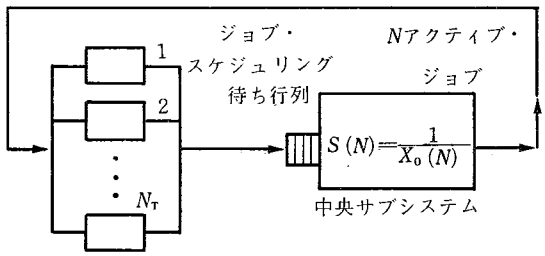
高位レベルのモデル (時分割システムのモデルである) において，中央サブシステムは，以下の負荷依存型サービス関数を持つ等価な装置と置き換えられている．

$$S(N)=\frac{1}{X_0(N)}.$$

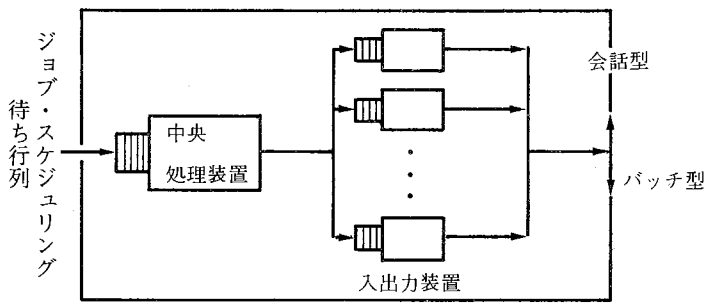
ここで，$X_0(N)$ は，低位レベルのモデルから決定される負荷 N の下での中央サブシステムのスループットを表す．

3.　分析および解法

Courtois によって発展させられた分解技法，Chandy, Herzog, Woo のサブネットワーク倒潰法 (CHW 定理)，および流量等価システムズ法の目標は同じである．ただ，前の2



（a） 高位レベルのモデル（外側のモデル）



（b） 低位レベルの中央サブシステム・モデル（内側のモデル）

図 2 図1の階層化モデル

Fig. 2 A hierarchal model of Fig. 1

つの技法は分析面で一層形式的であり，第3のシステムは非形式的だがより応用しやすい，という違いがある．このことから，分析に用いるモデル化技法の，最も重要な近似解法への方針が導ける．なぜならば，複雑なコンピュータ・システムの現実的な分析を行うために，すべての可能な技法や成果を統合し，互いに補完させることが可能になるからである．流量が等価なシステム(合成の装置からのジョブの流量が，サブネットワークからのジョブの流量と等しくなるようなシステム)という概念に基づいて，図2(b)の中央サブシステムは，サービス関数 $S(N)=1/X_0(N)$ を持つ等価な装置によって置き換えられる．そのサブシステムは，一連の管理された実験のために，オフラインで調べることができる．サブネットワークの分析は，利用可能なソフトウェア・パッケージ(たとえば Univac 開発センターの SNAP プログラム注))を用いて行われる．また操作解析の手段によっても行うことができる．分析中の負荷 N を一定にするため，分析者はジョブ完了後に直ちに別のジョブを追加する．このようにしながら，実際のシステム上で直接に計測する．T 秒の計測期間中に C 個のジョブが完了した場合，条件付き出力率は $X(N)=C/T$ と設定される．高位レベルのモデルは，負荷依存の供給器とともに，会話型ユーザを含んでいる．負荷依存の供給器は，一時的に発生しうる，それぞれの負荷に対する会話型スループットを表現している．パラメタの指定とモデルの解は，このレベルでは，きわめて簡単である．この分析によって，バッチ型および会話型のジョブの平均スループット，思考中の会話型ユーザの平均の数，およびそれぞれの一時的な負荷レベルでシステムが消費する時間の比率，等が求められる．

確率過程論に基づいた，待ち行列網の解に対する伝統的なアプローチ，およびこれに関連する仮定は，Markov 待ち行列網解析と呼ばれる．Buzen[4]によって提案された操作的なアプローチは，基本的な数学的枠組として操作的解析を用いている．そこでは，以下の

3つの操作原理から，基本的な等式および結果を導き出す．

1) すべての量は厳密に計測可能であるべきであり，すべての仮定は直接に検証が行われる．

2) システムは，流量の均衡が保たれていなければならない．

3) 装置(供給器)は一様 (homogeneous) でなければならない．すなわち，ジョブの経路は局所的な待ち行列長から独立していなければならず，平均サービス時間は他の装置の待ち行列長から独立している．

操作原理から伝統的な Markov 過程と同一の等式が導かれる．異なるのは，操作上の仮定が検証可能であり，成立しないことはまれであるという点である．この操作的な待ち行列網解析はモデルの妥当性を詳細に検査できる．また，ある性能量を他の性能量から計算して求めることによって，データ収集が単純化できる．操作結果は理解しやすく，簡単に性能計算や性能予測に応用できる．操作法則と操作式は計測される量に直接に関係しており，計測されたデータに矛盾がないかどうかが検証できる．さらに，操作的解析は，確率解析によって閉じた形で解を求めることができるようなネットワークのクラスに応用できる．したがって，われわれは，待ち行列網モデルの利用に関し確信をもって当たることができる．それぞれの問題に対して，操作解析は，次の2つの基本要素が存在する．

1) システム(現実のものでも，想定されたものでもよい)

2) 期間(観察期間)

以下の項目は，観察期間(T)中に直接計測される基本量である．

A：解察期間Tにおける到着数

C：観察期間Tにおける完了数

B：システムが稼動中であった間の総時間

上記の量から以下の量が導き出される．

$\lambda = A/T$　　到着率

$X = C/T$　　スループット(出力率)　(ジョブ/秒)

$U = B/T$　　利用効率

$S = B/C$　　完了ジョブ当たり平均サービス時間

ジョブ当たりの装置 i への入出力要求の平均数，すなわちジョブ当たりの装置 i への訪問の平均数を訪問率といい，V_i で示すものとする．

$$V_i = X_i/X_0$$

ここで，X_0 はシステムのスループットである．

以下は操作式(操作則)である．

利用効率の法則：$U_i = X_i S_i$

Little の法則：$n_i = X_i R_i$

ジョブの流量の強制法則：$X_i = V_i X_0$

出力の流量の法則：$X_0 = \sum_{i=1}^{k} X_i q_{i0}$

ジョブの流量均衡の仮定(操作原理3)を導入すると，以下の操作式が成立する．

会話型応答時間の関係式：$R = N_T/X_0 - Z$

すべての性能量は，訪問率(V_i)と平均サービス時間(S_i)をパラメタとして与えることにより，計算して求めることができる．実際に，分析では作業負荷に関するデータから，直接に訪問率を抽出することができるであろう．網内の任意の場所の流量をジョブの流量

の強制法則を介して決定するため，分析では，訪問率を用いることができる．このとき，システム内でジョブの流量が均衡しているという仮定を導入することになる．さらに，会話型システムの応答時間もまた見積もることができる．

以上のことを考慮した上で，閉じた待ち行列網システムのスループットと応答時間に見られる漸近線的な振舞いを見る．システム内のジョブの数(N)が増大するにしたがい(訪問率と平均サービス時間は，Nの変化によっても変化しないことを仮定して)，利用効率がほぼ100パーセントになったならば，この装置は“飽和”する．“飽和”している装置はその能力をすべて用いて稼動している．Nの増大によって飽和に至る装置は隘路(bottle neck)と呼ばれる．装置bが隘路であるならば，次の式が成立する．

$$V_bS_b=\max(V_1S_1,\cdots,V_kS_k)$$

隘路は，装置および作業負荷といったパラメタによって見つけることができる．Nの増大により$V_b=1$となり，システムのスループットの最大値は，

$$X_0=1/V_bS_b$$

となる．

待ち行列上での遅れを無視するならば，平均応答時間の最小値は，

$$R_0=\sum_{i=1}^{k}V_iS_i$$

となる．

図2(a)のような会話型システムを考えてみよう．N_T個の端末のそれぞれは思考時間Zを持っており，稼動ジョブの数をNとすると，思考中の端末数はN_T-N個である．中央サブシステムには，一様なサービス時間と訪問率(ともにNの値に依存しない)を持ったK個の装置がある．稼動ジョブによる負荷$P(N)$の分布は，以下の状態空間均衡式を満足する．

$$P(N)X_0(N)=P(N-1)\frac{N_T-N+1}{Z}$$

Buzenの論文[4]に記述されているアルゴリズムから得られる値$P(O)$を用いて上の均衡式を繰り返し計算することにより，$P(N)$の値を計算することができる．

図3は，このアプローチから得られる，中央サブシステムのスループット関数$X_0(N)$を示している．直線で示されるものは思考中の端末からのジョブ投入率(job submission rate)であり，$(N_T-N)/Z$で表される．両者の交点(N^*)は，ジョブの完了率(システムのスループット)と投入率が均衡するような負荷の値を示し，Nの最も望ましい値を表す．

システムのモデル化技法について異なったアプローチを採用することにより，図4に示

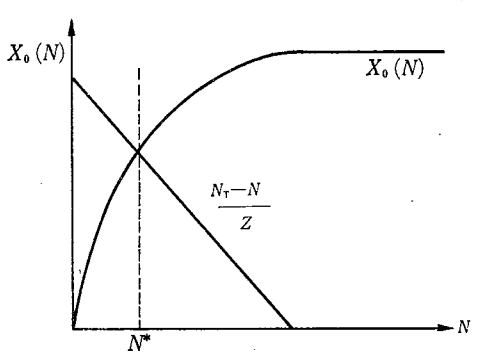


図 3 スループット関数および負荷分布
Fig. 3 Throughput function vs load distribution

されるような混合システムを考えることができる．会話型作業負荷は，N_T 個の端末から投入されるジョブを意味し，バッチ型作業負荷はその他の手段によって投入されるジョブを意味する．バッチ型作業負荷に関してシステムは開かれているので，バッチ型ジョブの数は変化しうる．

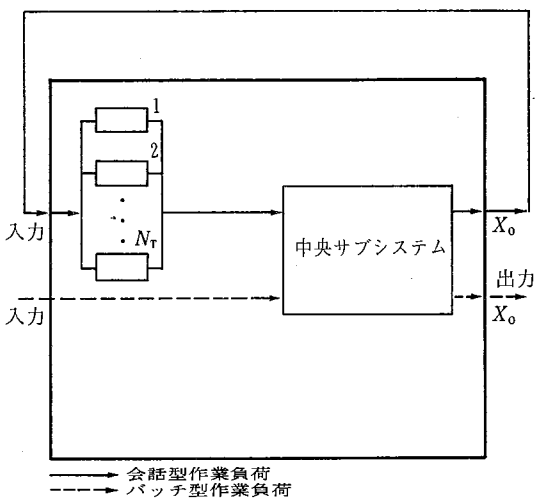


図 4　混合システム
Fig. 4　A mixed system

この場合，バッチ型スループット X_0 は既知であるが，会話型スループット X_0' は X や待ち行列網の他のパラメタに依存している．

いま，以下のデータが得られたとする．

・システムには 40 個の端末がある ($N_T=40$)
・思考時間は 15 秒である ($Z=15$)
・ディスクの平均サービス時間は 40 ミリ秒である ($S_i=40$)
・各会話型ジョブはディスクへの入出力要求を 10 回行う ($V'_i=10$)
・各バッチ型ジョブはディスクへの入出力要求を 5 回行う ($V_i=5$)
・ディスクの利用効率は 90 パーセント ($U_i=0.9$)
・バッチ型ジョブの投入率は 0.5 ジョブ/秒 ($X_0=0.5$)

以上のデータから会話型スループット X'_0 と応答時間 R' を計算したい．

ジョブの流量の強制法則から，バッチ型ジョブに関する装置の完了率を求めることができる．

$$X_i=X_0V_i=(0.5)\cdot(5)=2.5 \quad \text{要求/秒}$$

利用効率の法則から

$$U_i/S_i=X_i+X'_i=(0.9)/(0.04)=22.5 \quad \text{要求/秒}$$

したがって，会話型ジョブに関する装置の完了率は，

$$X'_i=22.5-X_i=20 \quad \text{要求/秒}$$

ふたたびジョブの流量の強制法則から，会話型スループット X'_0 は，

$$X'_0=X'_i/V'_i=20/10=2 \quad \text{ジョブ/秒}$$

会話型応答時間の関係式から，

$$R'=N_T/X'_0-Z=40/2-15=5 \text{ 秒}$$

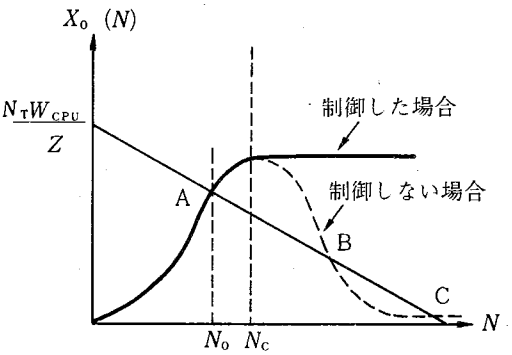
となる．

現在の多くのコンピュータ・システムは，多重プログラミング，仮想記憶システムである．これはユーザに実際に割り当てられる主記憶装置の大きさよりも大きな仮想アドレス空間を持つ．このことはページングと呼ばれる動的アドレス変換機構を手段として行われる．このようなシステムでは，多重プログラミングされるジョブの数 N は時間の経過によって変化する．負荷 N は通常，ジョブ・スケジューラによって制御されている．普通，ユーザのアドレス空間のうち必要な部分は，要求ページングという方式によって，主記憶装置上に移される．スケジューリングや資源割当ては，2段階に分けて行われる．第1段階では，端末から投入されたジョブはジョブ・スケジューリング待ち行列に置かれ，多重プログラミングされるまで待たされる．このローディング前処理がなされた後，プログラムはレディ状態となり，中央処理装置(CPU)のスケジューリング待ち行列に加わる．そしてCPUの使用を他のレディ状態にあるジョブと争うことになる．いったんCPUがジョブに割り当てられると，ジョブは仕事を終えるか入出力操作が必要となるまでCPUを使うことができる．通常は，過大な待ち行列上での遅れを防止するためにタイム・スライスの技法が採用される．本質的にそのようなシステムでは，すべての事象はページングによる入出力要求である．ページングとページングの間のCPU実行時間は，入出力要求に要する実行時間に匹敵するか，それより少ない．したがって，ページの読み込みが行われている間，CPUは他のレディ状態にあるジョブに渡される．ジョブは，そのアドレス空間が仮想記憶装置に移されたとき，内側のモデル(中央サブシステム)の対象となる．ジョブは，(a)完了または(b)割り当てられたタイム・スライスの消費のどちらかの理由により内側のモデルから離れる．後者のとき，そのアドレス空間は仮想記憶装置からなくなり，ジョブはジョブ・スケジューリング待ち行列に戻る．

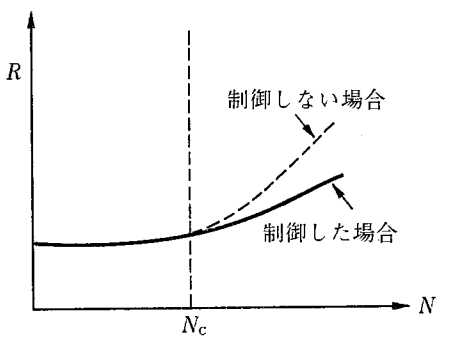
図5は，負荷 N に対するスループット $X_0(N)$ の典型的な曲線を示す．N(多重プログラミング・レベル)が増大するにつれて，初めのうちはCPUの生産性も上昇する．これは，CPUがより多く使われており，CPUと入出力とのオーバーラップの利点が発揮されているからである．しかし，Nがさらに増大すると，個々のプログラムに割り当て可能な記憶装置の空間が少なくなり，ページ不在率は鋭く上昇する．それは，CPUのほとんどがページングの処理に使われ，有用な仕事の処理がほとんど行われなくなる状況に至るまで続く．このような現象をスラッシングと呼ぶ．したがって，ジョブ・スケジューラが多重プログラミングの適切なレベルを維持し，システムが集中しないようにすることが重要になる．最適なあるいは最適に近い多重プログラミング・レベルを見つけるための，簡単なグラフ手法を図5に示す．いま，1つのトランザクションを処理するのに必要な命令の合計の平均値を W_{CPU} とする．システムへの作業負荷の流入率 (inflow rate) は，次のようになる．

$$\frac{(N_T-N)\cdot W_{CPU}}{Z}$$

そして，システムからのスループットは $X_0(N)$ である．図5にある「制御した場合」の $X_0(N)$ (実線部分)とは，ジョブ・スケジューラが N の値を臨界点 N_C 以下に定めているときのスループットである．制御を行わないシステムの場合(破線部分)，3つの可能な操作点，A，B，Cがある．B点は不安定な位置にあるが，AおよびC点は安定な位置にある．またC点は，スループットが非常に低い値を示すことから，望ましくない位置である．たとえ，初めのうち，システムが望ましい位置A点を維持できたにしろ，ジョブが



(a)



(b)

図 5　時分割仮想記憶システムの(a)スループット (b)平均応答時間の近似解析

Fig. 5　An approximate analysis of (a) throughput (b) average response time of time sharing virtual system

固まりとなって大量に到着したり，ある時期にいくつかのジョブが極端に長い時間サービスを受けたりすると，システムはC点に移行することになる．制御の目的は，システムがいつも望ましくかつ安定した位置A点の近辺で実行されるのを保証することである．このような制御を行うことにより，たとえ，N_T がさらに増大したとしてもスループットが極端に低下することはない．

平均応答時間のおよその見積りは，以下の式から得られる．

$$T_r=\frac{N}{X_0(N)}-Z$$

制御を行った場合(実線)と行わなかった場合(破線)について，T_r の変化を図5(b)に示す．

より精確にモデル分析を行うため，平均的なジョブの総処理時間に一致させて時分割する．これにより，証明済みでかつ現実的な結果が適用できる．これらの結果を用いれば，CPU の処理装置共用モデルはラウンド・ロビン・アルゴリズム(round-robin algorithm)に満足のいく近似を与える．

図2(a)に示した，並列に置いた端末の集合は，数学的には，無限の数の供給器を持った1個の装置と等価である[5]．このような条件のもとで，マクロモデル(高位レベルのモデル)は，2個のサービス・センタを持つ1つの閉じた網と考えることができる．これは図

6に示される閉じた待ち行列網である．サービス・センターの一方は供給器 IS (infinite server, 無限の供給器)であり，N_T 個の端末を代表する．他方のサービス・センターは供給器 PS (processor-sharing) であり，タイム・スライスによる多重プログラミング・システムの数学理論上のものである．Buzen のアルゴリズム[5]は以下のような性能量を得るために用いることができる．

行列 g を計算するため，以下の式を用いる．

$$g(n, k)=g(n, k-1)+Y_k g(n-1, k)$$

ここで，$Y_k=V_k S_k$ であるので，

$$\text{利用効率}: U_i=Y_i\frac{g(N-1, 2)}{g(N, 2)}$$

$$\text{システムのスループット}: X_0(N_T)=\frac{g(N_T-1, 2)}{g(N_T, 2)}$$

このアルゴリズムの実装は，待ち行列網モデルを用いた多くのソフトウェア・パッケージの中に見いだすことができる．

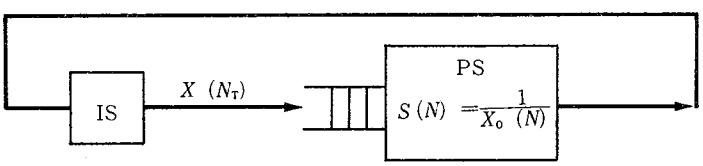


図 6 等価な端末起動多重プログラミング・システム
Fig. 6 Equivalent terminal driven multiprogramming systems

多重プログラミング・レベルNの制御(あるいは調節)は，仮想記憶コンピュータ・システムの最適な性能を引き出し，スラッシングを防止するのに重要な意味を持っている．この制御のアルゴリズムは様々な形をとっている．実際的な負荷制御のアルゴリズムを発見し，プログラムの振舞いとコンピュータ・システムの性能間の関連性を示す場合，多重プログラミング・システムのモデルと同じように，プログラムのモデルや作業負荷のモデル，さらに記憶装置の割当て方法のモデルを必要とする．

現在では，状態遷移モデル[6]がプログラムの振舞いを最も正当に記述していると考えられている．この種のプログラム・モデル化技法で用いられているアプローチは，状態と遷移の振舞いとを切り離すものである．その中では，状態に遷移の回数や関連する局所的な状態集合を記述するために巨視的なモデルを使う．そして微視的なモデルは，巨視的なモデルによって決定されるそれぞれの状態中の参照パターンを記述するのに使う．

巨視的なモデルの例には，状態の局所的な状態集合と遷移の回数を記述する準 Markov 過程モデルがある．微視的なモデルの例には，IRM (the independent reference model) や LRUSM (the least recently used stack model) といったプログラム・モデルがある．記憶装置割当ての管理にワーキング・セット原理を用いる方法は，スループットが最大限に最適化されるように多重プログラミングの程度を決定できることがわかっている．また，ライフタイム関数(与えられた実記憶装置の大きさに対するページングとページングの間の平均時間を指定する関数)が，実際的な最適負荷制御ルーチンの設計の基礎情報(プログラムの振舞いに関する情報)を十分に含んでいることも示されている．

4. おわりに

待ち行列網モデルの分野では，より現実的なモデルが構築でき，また効率的な計算アル

ゴリズムが提供できるようになっている．これらの進歩により，従来よりも経済的に複雑なシステムをモデル化できるようになった．本稿では，これら様々なモデル化技法を統一された方法で結び付けること，および性能分析を行う上でこれらの方法を互いに補完させること，を意図している．分解法の主要な応用としては，モジュール化を通して問題を単純化することがあげられる．操作的解析は，待ち行列網の応用をわかりやすく実際的なものとしていることから，Markov 待ち行列網を補うものである．

複雑なコンピュータ・システムの振舞いを理解するのに必要な洞察力を得るため，いくつかのアプローチを述べた．さらに他の方法をとった場合には，理解が困難であるようなある種の質的影響を説明した．

本稿の目的は，性能分析のために適切に構造化されたアプローチの普及である。分析は，同格の立場に基づいて行われるべきである．そこではすべての手に入る技法やツールが，互いに補完し有効性を高めるように，併用される．この方針を説明するために，異なった方法論間に存在する結合点を本稿は明らかにしている．

（汎用システム事業部　小田 澄男 訳）

参考文献

[1] F. Baskett, K. M. Chandy, R. R. Muntz and J. Palacios, "Open Closed and Mixed Networks with Different Classes of Customers", *Journal of the ACM*, Vol. 22, No. 2, April 1975, pp. 248-260.

[2] K. M. Chandy and C. H. Sauer, "Approximate Methods for Analyzing Queueing Network Models of Computer Systems", *Comput. Surv.*, Vol. 10, No. 3, Sept. 1978, pp. 281-317.

[3] P. J. Courtois, "Decomposability, Instabilities, and Saturation in Multiprogramming Systems", *Comm. ACM*, Vol. 18, No. 7, July 1975, pp. 371-377.

[4] J. P. Buzen, and P. J. Denning, "The Operational Analysis of Queueing Network Models", *Comput. Surv.*, Vol. 10, No. 3, Sept. 1978, pp. 225-261.

[5] H. Kobayashi, "Modeling and Analysis", Addison-Wesley Publishing Company Inc., Reading. Mass. 1978.

[6] P. J. Denning, "Optimal Multiprogrammed Memory Management", *Current Trends in Programming Methodology*, Vol. III, Software Modeling, K. M. Chandy and R. T. Yeh (eds), Prentice-Hall 1978, pp. 298-322.

執筆者紹介　Benjamin S. H. Wang

1960年台湾国立大学卒業．1964年 Missouri 大学大学院修士課程修了．M. S.(電子技術)．同年より4年間，Westinghouse 社調査センターにて従事．1972年 Illinois 大学より Ph. D(コンピュータ・サイエンス) 取得．同年 B. F. Grrodrich 社に入社．1978年 Sperry Univac 社へ移籍，現在，モデル化技法，シミュレーション，コンピュータ・システムの性能分析・評価等を担当．

編集者注　SNAP とは，System Net Activity Program の略称であり，VS/9 システムにおける CPU，チャネル，各タスクのアクティビティの測定に用いるプログラムである．

報告 移植性——高水準言語の作成事例

Portability——High Level Language Implementation

F. P. Mehrlich, S. M. Tanner

要 約 本稿は，UNIVAC シリーズ1100，シリーズ90，およびシリーズ80といった処理系に依存しない単一の原始コードの ANSI COBOL コンパイラの作成における Salt Lake のソフトウェア開発センターでの経験を報告したものである．なお，このコンパイラは，シリーズ1100，シリーズ90およびシリーズ80の上で原始プログラムを翻訳し，UTS 400 および UTS 4000 端末機群で実行可能な目的コードを作る．このプロジェクトによって，PLUS を使用した単一の原始コードの概念は今後もさらに利用でき，効率的なプロダクトを生成することが実証された．この処理系に依存しないコンパイラを生成するという方法は，種々のコンピュータ用のコンパイラを独立して開発・保守する場合よりも，はるかに費用が少なくてすんだ．

本稿では，移植の概念，移植性のあるプロダクトを開発する際に遭遇する特有の問題や考慮点，および UTS 400 COBOL のプロジェクトにおいて効果的に採用された技法について説明し，得られた結果をまとめる．

Abstract This paper reviews the experience of Salt Lake Software Development Center in creating a single source ANSI COBOL compiler, to run on 1100 series, series 90 or series 80 systems while producing interpretable code for the microprocessor based UTS 400 and UTS 4000 families. This project has demonstrated that the single source concept, using PLUS, is viable and has produced an efficient produdt. The incremental cost in attaining single source is far less than would have been the case with separate development and maintenance efforts.

This paper presents the concepts of portability, the unique problems encountered and special considerations in developing a portable product, and the techniques which were effectively employed on the UTS 400 COBOL project, and then summarizes the results which were obtained.

1. はじめに

今日，多くのアーキテクチャおよびオペレーティング・システムがわれわれを取り囲んでいる．一方，コンピュータ産業は絶えず変化しつづけている．大きなシステムはより大きく，小さなシステムは(少なくともある部分)ではより小さくなっている．これらのシステムを開発し作成する過程では，費用・人的資源・時間，および他の有限な資源の投資額が流動的なことから，多くの負担がかかってくる．この開発に動機付けを与える主な要因としては標準化活動，ソフトウェア工学の進展，競合力保持の必要性等といった新たな環境がある．本稿では，この多大な投資およびその各々に関連する諸事情を解析することにより，ソフトウェア開発の分野にはまだまだ改良の努力をしなければならないことを示す．これまで，低価格のハードウェアを設計・生産するために多くの研究がなされた．これと同様にソフトウェア開発に対しても，そのプロダクトの生産および保守の効率を高めるために努力しなければならない．ここで考慮すべきことは，いかにして経済的に目標とするシステムや環境に対応しうるかという点である．

Reprinted from ACM SIGPLAN NOTICES, (January 1980), Vol. 15, No. 1, pp. 139-145.

2. 移植性の概念

これに挑戦する1つの手段として，移植性 (portabilily) として知られている概念の採用があげられる．“移植性”とは，異なるソフトウェア環境を持つアーキテクチャの下で操作可能な単一のソフトウェアを開発するときに用いられる概念と技法とを指している．移植性を適用することによって，そのシステムは，標準の制定，保守性の向上，および柔軟性が達成できる．

現在，新しいコンピュータ・システムが使用可能になると，そのシステムに適した新しいソフトウェアを作るための多くの努力が始まる．すなわち，その強化された機能や最新技法による設計を有効に活用し，利用者に適したソフトウェアが容易に選びだせるように，新システムを支援するソフトウェア作りが行われる．多くの場合，言語プロセッサといった主要ソフトウェアは，再設計および再コーディングする．その結果，肥大した開発および保守という困難な工程が再び始まる．

このことは，新システムに限られたわけではない．標準の新設や改訂，新技術の発表，または環境の変化等のために，すでに確立しているシステム上で，この工程は再び発生する．この場合の不利益な点として，たとえば，同種の仕事のために複数のソフトウェア・プロダクトを作成すること等があげられる．ソフトウェアの再開発・改良には多大な努力が必要であることは論を待たない．また，たとえどのように注意深く標準によったとしても，不明確な領域の解釈の違いや異なるシステム上での異なる開発による違いにより均一でないプロダクトになるという不利益もある．

移植性はコンピュータの分野では新しい概念ではない．この技法は，ある期間 FORTRAN や COBOL のような標準化された言語を使って作られた応用プログラムで成功していた．言語プロセッサの設計に移植性の概念を適用する際，とりわけマイクロプロセッサの環境においては，2つの異なった局面が見えてくる．これは，開発作業そのものと，目標となる機械の目的コードの形式とである．

3. UTS 400 COBOL に適用された概念

Sperry Univac 社では，標準化された高水準言語を選び，新たに UTS 400 インテリジェント端末シリーズ用の目的コードを生成することを検討した．さらに，これは，UNIVAC シリーズ 1100 では OS 1100 を使い，90/30 シリーズ(およびシリーズ 80)では OS/3 を使い，90/70 シリーズでは VS/9 を使って動くクロス・コンパイラである必要があった．この基準に最も合致する言語は COBOL であり，これを開発するための言語として PLUS (Programming Language for Univac Systems[1]) が適当であると決定した．

PLUS 言語は，Sperry Univac 社のシステム記述言語である．PLUS 言語は，構造化プログラミング技法と強力な命令群を支援するアルゴリズム言語であり，複雑なデータ構造や種々のデータ型が使用できる．さらに PLUS は，いま考えられているすべてのホスト・コンピュータ上での開発ですでに成果をあげており，よい最適コードを生成するという利点がある．

前述の要求を満たすために，COBOL コンパイラの設計は，ある程度の非能率性は妥協しつつ，できあがったプロダクトが移植性を持つように方向づけられた．またコンパイラの目標としては，端末個有の命令群を生成するより，むしろインタプリンタで解釈実行可能な目的コードを生成することとした．

4. 移植性を持つ言語の設計と開発

4.1 移植性を持つ言語の設計に対する最初の考察

“移植性”について考慮すべき事柄と基準を見とおすために，コンパイラが複数のホスト上にあり，複数のシステムに対する目的コードを生成するという一般的な場合を調べてみる．図1に，コンパイラの内部処理を，機能とフェーズに分類して示した．はじめに原始コードが構文解析部 (syntactic analyzer) で解析されて，中間言語 (intermediate language : 略して IL) および記号テーブル (symbol table) が生成される．つぎに IL は，コード生成部 (code generator) および最適化部 (optimizer) によって，ある1種類の機械の目的コードに変換される．多くの異なったシステムの目的コードを得るために，コンパイラの後のフェーズ(たぶんテーブル駆動形であろうが)は，必要な命令群用のコードを生成するように修正される．これで1種類のコンピュータ上でいくつかのコンピュータ用のコードを生成することができて，クロス・コンパイラを完成したことになる．ここで，すべてのシス

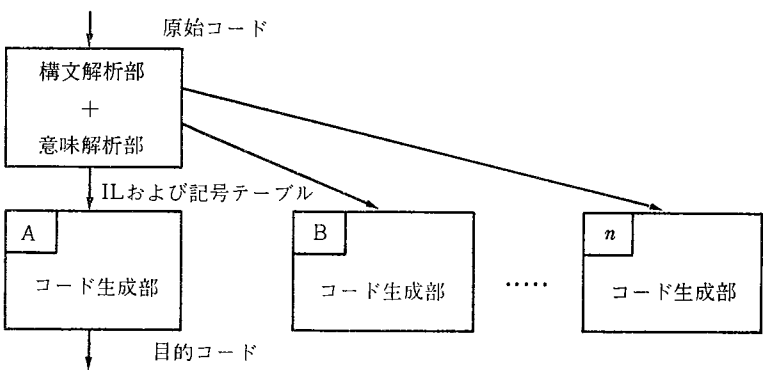


図 1 移植性を持つコンパイラの設計
Fig. 1 A portable compiler design

テムに共通で，かなり強制力のある標準を持つ開発用言語を使用したとすれば，このコンパイラは他のシステムの上でもコンパイルが可能である．こうして各々が多くの異なる目的コードを生成することのできる一群のコンパイラが完成する．

4.2 移植性に対する要求

“移植性”を採用するに当たって，“一般的に効率が悪くなる”という指摘がある．これには，いくつかの納得できる理由がある．しかし十分な検証により，この問題は避けられることが明らかとなる．上記の問題を緩和するためには，すべてのシステム上の開発用言語において構造的データ (structural data) および手続きコード (procedural code) に対する最適化を義務づけることである．ある処理系に依存しないプログラムを作成するための重要な段階は，“移植性を持つ”機械(portable machine : 略して PM) を定義することである．このことは，各々のアーキテクチャに含まれない構成概念を模倣したり補足したりする面倒を，すべてのシステムから取り除く助けとなる．この機械はコンパイラ開発者にはホスト・システムのイメージとなる．UTS 400 COBOL の場合には，いくつかの基本単位を次のように定義した．

	1100	90/70	80 および 90/30	PM
1語	36 ビット	32 ビット	32 ビット	32 ビット
1語のバイト数	4または6バイト	4バイト	4バイト	4バイト
1バイトのビット数	9または8ビット	8ビット	8ビット	8ビット
ガードの長さ	132 文字	80 文字	72 文字	72 文字
ポインタの長さ	2バイト	3バイト	3バイト	3バイト

それから，種々のシステムの特性の適当なサブセットを作り出す．これによって，プログラム作成時に，別のホスト・システムへ移行する場合に問題となる点をほとんど考慮しなくてもよいようなコーディング技法が，簡単に利用できるようになる．

4.3 1つのプログラムとしてのコンパイラ

ここで，コンパイラを単純なプログラムや単純なアルゴリズムの集りであると考えよう(図2参照)．ただし，そのプログラムやアルゴリズムは言語プロセッサの中で階層的に編成され，うまく定義された機能を実行するものである．

これは，“移植性”の導入によって，コンパイラ設計のどの領域が影響を受けるかをさらに細かく考察するためである．各々のプログラムは，手続き領域とそれに関連するデータから構成される．2つの異なる領域はひとまとめにされ，より大きなシステムの論理的な構成要素となる．典型的な構成要素のデータ領域は，作業変数，構造的データ（すなわち制御テーブル），インタフェース（連絡）領域等を含んでいる．また手続き領域は，中間演算，ポインタの参照，サブプログラムの呼出し等から構成される．これら各領域は，“移植性”の導入によって何らかの影響を受け，技法が複雑になる可能性がある．しかし，実際に必要なことは，一定のプログラミング形式を決定することであり，その形式を修得させる訓練は非常に単純である．このことから，“移植性を持つ”コードのまったく自動的な生成が可能となる．

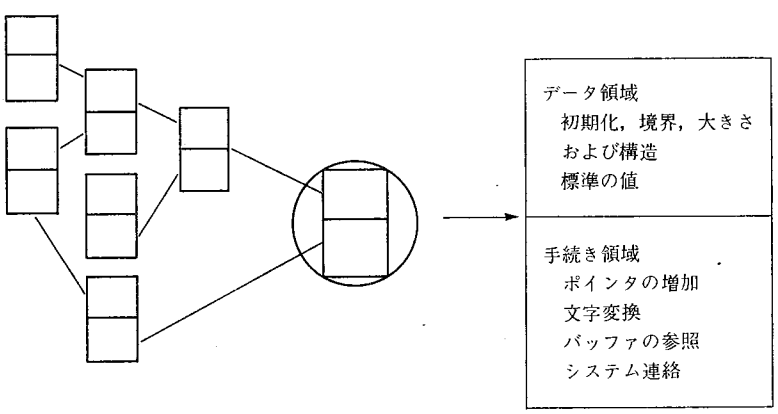


図 2 典型的コンパイラの構成要素
Fig. 2 A typical compiler subcomponent

4.3.1 宣言領域の考察

まずデータ領域の設計に関して，いくつかの考慮すべき点を調べる．すべてを完全に調べることはできないが，代表的な分野がもれることはない．この際の1つの重要な観点は，初期化についてである．このことは，しばしば見落されたり，当然なことと思われたりしがちである．それぞれのホスト・コンピュータにはそれぞれの仮定が存在する．たと

えば，文字表現はどうか，作業領域が実行に先立って自動的にクリアされるかどうか，また実際の記憶領域の中に存在する自然な境界がどうなっているか，等々．通常は，これらの機能に依存することによって，より良い効率が確保される．しかし“移植性を持つ”プログラムにおいては，これらの機能は，ホスト・システムが変更されると，重大な悪影響を引き起こすおそれがある．この問題を解決するには，宣言で初期値を設定していたり，代入文で値をとくに設定している場合を除いて，すべてのデータは初期化されていないものとして扱うべきである．また，初期化された文字列は，印書情報，内部比較，または原始入力に使用されるときはいつでも，システム標準であると仮定すべきである．このことにより，変換の必要性が減少し，開発言語は効率の良いコードを生成することができる．

主記憶領域の自然な境界の利用は非常に重要であるが，一連の異なる機械の上で，いくつかの不一致が存在するということも明らかである．たとえばシリーズ 1100 の 1 語は 36 ビットからなり，1 語は 6 バイト(この場合には 1 バイト=6 ビット)または 4 バイト(この場合には 1 バイト=9 ビット)に分割できる．シリーズ 90 およびシリーズ 80 の 1 語は 32 ビットからなり，1 語は 8 ビットの 4 バイトに分割できる．それゆえ，ホスト・コンピュータにとって最も能率的にデータを参照するために，各々のデータの属性を宣言するときに抽象的な方法を採用する．すなわち，ビット数によって割り当てるより，むしろ WORD とか BYTE のような属性を使用する．こうしておくと PLUS 言語は，与えられたホスト・コンピュータで自動的に最適の表現を作り出すことができる．

これらの簡単な基準に従うことによって生じるデータ領域は，このコンパイラが必要とする複雑なアルゴリズムの演算を満足させるのに十分なデータを含み，さらにホスト・システムの標準の特性を利用することによって，データの値や構造的表現およびデータのアクセス等は，目に見えるほど大きな損失をこうむることはない．結果的に与えられたホスト・システムでの独特な要求に対して，可能な最上の方法でこたえられる．

4.3.2 手続き領域の考察

つぎに，手続き領域に関して考察する．手続き領域では，すでに大部分のコードが移植性を持っている．したがって，ここでは移植性を持たない部分を分離してとりあげることにする．これらの領域は定義がきちんとされていることが多く，作業は大幅に単純化できる．コードに対するハードウェア負荷の顕著な例は，ポインタに用いる増分の負担である．シリーズ 1100 では語単位の番地付けを行い，シリーズ 90 およびシリーズ 80 はバイト単位の番地付けを行う．したがって，ポインタに 1 を加えた結果は処理系ごとに異なっている．この問題は，次の 2 つの方法のうちのいずれかで回避できる．可能なら，ポインタに基づく (BASED 属性を持つ) 構造は指標付けした構造に置き換える．これにより，両方のシステムがそれぞれの方法で指標を決めることができる．しかし，これが実用的でなくて，ポインタを使用する必要があるならば，増減値に一定の係数 (シリーズ 90 およびシリーズ 80 に対しては 4，シリーズ 1100 に対しては 1 と定義されている)を乗じることによって増減を行う．シリーズ 1100 の表現においては 1 による乗算が生成されるが，PLUS コンパイラの最適化のフェーズではこの段階を省略する．シリーズ 90 およびシリーズ 80 のコンパイラ上では，4 による乗算が生成され，2 ビット左へ桁送りするようにコンパイルされる．このようにして，ポインタを用いる増分に関する問題が非常に効率的に解決できる．

もう 1 つのハードウェア負荷上考慮すべき問題は，文字の数値データをそれと等しい 2 進数のデータに変換することである．このために内部符号として ASCII を使用している

シリーズ1100では，各々のバイトの情報を調べ，その値から8進数の60を差し引いて(必要ならば10倍して)変換された結果を加えるというアルゴリズムを，採用する．内部符号としてEBCDICを使用しているシリーズ90およびシリーズ80上でも，16進数のF0に等しい差し引き係数を単純に再定義して，シリーズ1100と同一のアルゴリズムが適用できる．このようにして，この問題に関しても，単純な解決法を用いる．手続き領域で最後の考慮すべき問題は，オペレーティング・システムとのインタフェースであり，これについては次節で述べる．

4.4 システムの共同者としてのコンパイラ

さて，コンパイラは，オペレーティング・システムと多数のインタフェースを持つ複雑なプログラムである(図3参照)．通常，コンパイラの機能の多くは，コード内に直接統合されている．これはインタフェースの連結を簡単にするためである．一方，“移植性を持つ”コンパイラでは，すべてのインタフェースのためのコードを，広範囲のホスト・コンピュータやシステムに適用できる，一般的な方法で定義しなければならない．そのためには，一般的な呼出し系列を持った一連のアセンブリ支援ルーチン群が必要となる．UTS COBOLコンパイラでは，各々のフェーズおよびコンパイラ全体にわたる個々の要求を処理する支援ルーチンを定義し，準備している．各ホスト・コンピュータ上には，同一の方法で整理された同一の名前と機能を持つアセンブリ・ルーチンがある．

これらは，論理的な互換性を保証し，よって，アセンブラ・レベルの保守および拡張を簡素化する助けとなる．これらの問題に関連する領域の例に，入出力の誤り処理，プログラムの流れ，パラメタ処理，機器の割当て，およびユーティリティの支援等がある．

4.5 特別な問題の実例

“移植性を持つ”コードを設計する上で考察すべきいくつかの問題が，コンパイラの開発過程で明らかになった．

シリーズ1100のPLUSコンパイラとシリーズ90(あるいはシリーズ80)のPLUSコンパイラではデータの構造上の表現が時によって異なっていた．ある条件のもとでのシリーズ90およびシリーズ80のPLUSコンパイラは，配列のそれぞれの写しを語の境界から始めるようにしてある．このことを知っていないとむだなテーブルを生成してしまうことになる．

つぎに，モジュール名は，一般にはシステム制御カードから取り出されるが，VS/9上

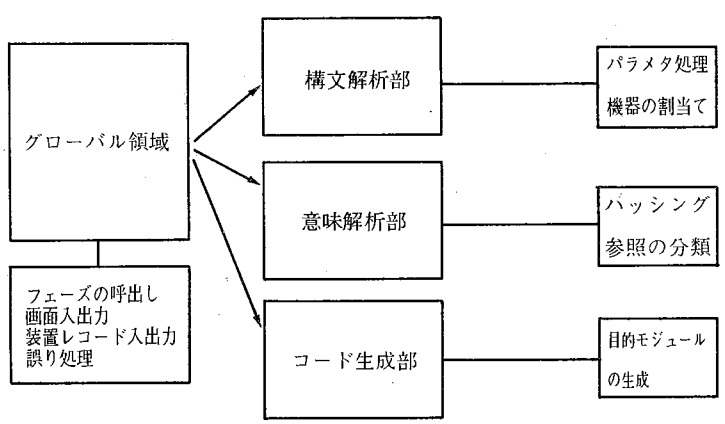


図3 システム・インタフェース・モジュール
Fig. 3 Sysem interface modules

では出力ファイルの名前を取り出すことができない．コンパイラは，使用者がモジュールカード上のオプションとして名前を与えることを仮定している．しかし，シリーズ1100上では，このオプションは文法上の誤りとなる．このため，シリーズ90のテープ作成時に，シリーズ1100用のプロセッサに MODULE 文を追加するように変更しなければならなかった．

また，VS/9 および OS/3 上で発見されたとくに捕えにくい誤りは，EXPORTED 属性を持つ名前の切り詰めに起因していた．シリーズ1100上では，外部名は FD コード表現をしているので，渡されるデータおよび手続きの名前は最初の12文字(2語)に切り詰められる．しかしシリーズ90およびシリーズ80上では，8文字(2語)に切り詰められる．このため，同一の外部名がなん回か定義されることになる．しかし，VS/9の連結編集プログラムでは，この状態を誤りとして検出しない(ただしOS/3では検出される)ラベルの個数が多いので，この誤りを作業者は見のがしてしまう．こうすると番地の参照は，最初に見つけられたラベルに結びつけられ，とんでもない結果を生じる．診断メッセージが表示されていないのでコード上は正しく見えるが，誤った動作を行うことは明らかである．

5. 移植性を持つ目的コードの設計

5.1 移植性を持つ設計の視野の拡張

これまでに，移植性を持つ言語処理系を開発するのに必要な考察および基準を述べてきた．ここでは目的コードの“移植性”を保証するために，調べなければならないその他の分析と考察を行う．

高水準言語には，2つの主な目標がある．第1は与えられたアプリケンションの構文上および意味上の支援であり，第2は与えられたアーキテクチャの設計との互換性である．ところが，アプリケーションが広がり，アーキテクチャが分散するとともに，直接システム設計とは互換性のない言語を開発することがしばしば必要となる．

その立場の顕著な例として，技術計算向きに設計されたシステムの上で，事務計算用の言語が使えるようにする作業がある．その結果として，ぎこちなく，複雑で，効率の悪い目的コードが生成されることになる．さらに別の考慮すべき点が，マイクロプロセッサを基礎としたシステムの出現とともに追加される．マイクロプロセッサは一般に，より小さな語長，非常に限定された命令群，最小限のレジスタ，原始的な入出力機能しか提供しない．そのため，浮動小数点演算やパック十進数演算のようないくつかの演算には，特別なルーチンを必要とする．それは存在しないハードウェア機能をエミュレートして，混み入ったデータ変換を行うルーチンである．

第4章の図1で，各々の新しいアーキテクチャおよび命令群を支援するためのコード生成部の開発を提案した．コード生成部の複雑さと，これが引き起こすオーバヘッドの面から，言語の設計に対して問題がある．すなわち，膨大な費用がかかるだけでなく，できあがったプロダクトの信頼性も疑わしくなる．コード生成部を解析した結果，ほとんどのコードがシステム間でまったく交換できないことがわかっている．そのため，各々の新しいシステム用にコード生成部の大部分を書き直さなければならない．書き直しの時間は，言語の可用性を損い，おそらく競合力を失わせる一因となろう．これらから，“目的コードの移植性”の概念は，“移植性を持つ”言語開発の設計概念と一致する．これによって，システムの進歩や多くの顧客にすばやく対応できる真に柔軟性のあるプロダクトが生まれるのである．

“移植性を持つ”設計を図解するために使用した前述の図1をあらためることによって，“移植性”の意味に対して新たな洞察を与えることができる．いくつものコード生成部を作成するよりむしろ，たった1つのコード生成部を使用する．目的コードの最終目標は，与えられた命令群を対象とするのではなく，言語の要求に適合し，柔軟な構造を処理する解釈実行可能なインタプリティブなコードにすることである(図4参照)．

5.2 インタプリティブな目的コードとその機能

第4世代のアーキテクチャの設計目標は，高水準言語の要求を高水準な演算や操作で処理する抽象的な仮想機械によって定義することであった．この設計によって，コンパイラは，より自由な環境，より進んだ機能，より大きな柔軟性が与えられる．

高水準の命令群は，使用者のアプリケーションに適合する機能，語長，およびデータの型を含んでいる．実際には，操作コードの各々のオペランドは，マイクロ・コード機械への手続き呼出しになり，その手続き呼出しが複数の機械レベルの命令を生成する．

この仮想機械の機械語を利用したコンパイラでは，コード生成が簡単に行え，さらに，この機械語を支援するシステム間で統一がとれていることからくる利点も利用できる．プログラムを実行するために必要な機能は異なる機械の間で変更されず，マイクロ・コード化されたプロセッサだけが機械に応じて変更される．

これまで，コンピュータ業界は積極的に新しい技術を追い求めたけれども，共通の機械語を定義してこなかった．たとえ定義されたとしても，既存の機械にそのような新しい機械語を使用することはできないであろう．したがって，この進んだ技法を既存の機械と将来の機械に適用するための実際的な方法は，“移植性を持つ”コンパイラによって，生成され洗練された言語向きのコードをソフトウェアで解釈することである．したがって，第4世代の技術は，既存のシステムに合致する，翻訳可能な方法，すなわち，システムの高水準な目的コードと機械コードの間でインタプリタとして働く中間的なソフトウェア・デバイスを使う，幅広い，柔軟な，真に“移植性を持つ”プロダクトをうみだす．

インタプリタの技法の利点は，主として次の点である．

1) インタプリタにより解釈実行されるコードを極端に簡素化できる．したがって，相当に大きく複雑なプログラムでも，同じプログラムを直接コード化するよりもさらに

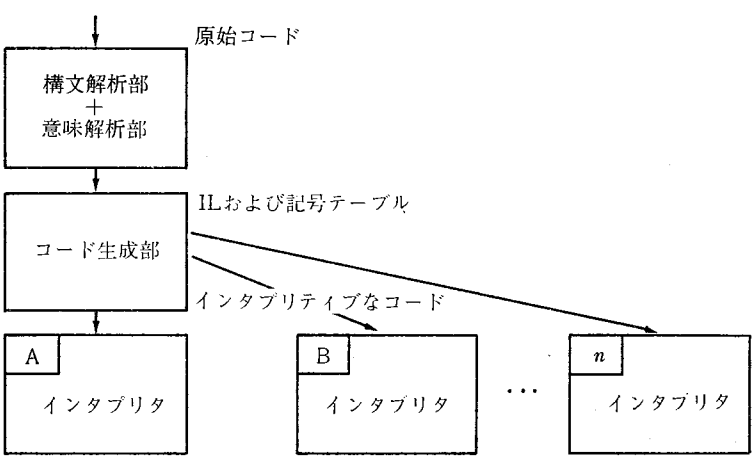


図 4 移植性を持つ言語の設計
Fig. 4 Portable language design

小さい記憶領域ですむ．これについては5.3節で詳しく論じる．

2) インタプリタによる解釈は，ハードウェアからの独立とソフトウェアの“移植性”に向けての第1段階である．ハードウェアを改良するかまたは変更する場合，すべてのソフトウェアを捨てて作り直す必要はなく，目的のハードウェアにインタプリタを書き直すだけでよい．ハードウェア・プロセッサやコードの簡潔さあるいはコードの実行速度に新旧のインタプリタ間では違いがあるだけである．

3) この技法はシステム・ソフトウェアの開発に対する構造的なアプローチである．プログラム作成可能な主記憶領域をわずか数キロバイトしか含んでいなくても，プロダクト開発を始めることができ，能力の拡張に伴い発展させることができる．

UTS 400 インテリジェント端末のための COBOL コンパイラ開発に，このインタプリティブな手法をうまく採用できた．他のインテリジェント端末のプロダクトの支援も，マイクロプロセッサをそれぞれ用いて，プロダクト用のインタプリタを追加開発するだけで敏速に成し遂げられよう．

5.3 インタプリティブな目的コードとその効率

生成されたコードを評価するとき，次の2つの主な要因が重要となる．これは実行速度と生成されたコードの大きさである．主記憶領域の大きさが制限されており，多くのマイクロ・コード命令から構成されるマイクロプロセッサの場合には，機能の豊富さと実行速度の釣合いが非常に重要である．

5.3.1 実行時間

インタプリタで大きなプログラムを走らせると実行速度が犠牲になると一般に考えられるので，コードの効率的な実行が問題である．インライン・コードの手法が最も速いと考えられるので，比較のための基準として用いる．100パーセントという表示は，システムがうみだすことのできる最高の速さをインライン・コードによって成し遂げるということを示している．サブルーチンの呼出しは，望んでいるサブルーチンに番地付けし利用するために必要とされる最少量のオーバヘッドのみであり，ほとんど100パーセントに等しい速さである．インタプリティブなコードの実行のオーバヘッドは，サブルーチンの呼出しの方法よりも速さがわずかに遅くなる．しかしながら，これはインタプリティブなコードの密度と，その解釈実行を成し遂げるために使用された実際の技法に大きく依存している．実用的な意味では，この差は図5に示すようにわずかである．

インタプリティブなコードの実行による負荷の低さは，マクロプログラム化された機械上での大きなオーバヘッドと対照的であり，マイクロプログラム化されたプロセッサの特性である．

インタプリティブな実行に対する理論的なオーバヘッドは，言語プロセッサにより生成

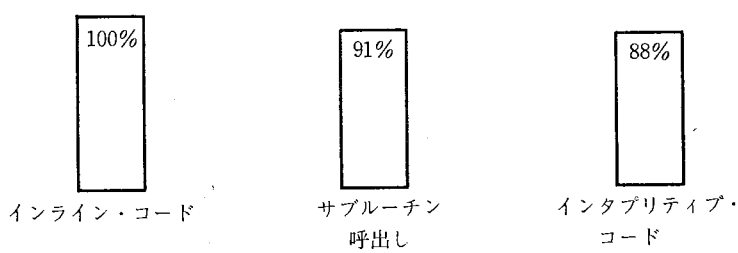


図 5 実行速度の比較
Fig. 5 Execution speed comparison

されたインタプリティブなコードと，そのコードがどのようににしてインタプリティブに処理されるかということによって決まる．いいかえると，理論的なオーバヘッドは，原始プログラムの命令に関係し，原始プログラムのデータとその属性の宣言に関係する．このオーバヘッドを減少させるのには，次の2つの要因を緩和させるとよい．

第1要因は，従来の技法を使った低水準言語によるコーディングの複雑さで，平均的なプログラマに効率的でないコードを作らせてしまう．これとは対照的に，インタプリタおよびインタプリティブなコード自身は，プログラムの大きさに関係なく十分に最適化できる．すなわち，とくに小さなプログラムに対しては余り有効ではないが，インタプリティブな実行は，使用者プログラムが大きくなればなるほど高水準言語によるコーディングの利点が増大する．ただし，実行時間は，同じプログラムが従来からのサブルーチン呼出し技法を利用して書かれた場合よりも長い．

第2の要因は，低水準マクロ言語により一般的に作成されたコードは最適でないということである．したがって，理論的に最適化されたコードとの比較は，実際の環境には適用できない．

5.3.2 主記憶領域の利用

主記憶領域内に常駐する小さなプログラムに対しては，目的コードの大きさはたいして意味を持たない．しかし，プログラムが大きくなるにつれてプログラムで使用可能な記憶領域を越えることがある．したがって，主記憶領域の要求とプログラムの大きさを考慮しなければならない．従来のコンパイラ技法に基づいた言語プロセッサは，インライン・コード，サブルーチンの呼出し，あるいはその両方の組合せを作る．これはまた，システムに基づく異なる各マイクロプロセッサに共通のフロント・エンド (front-end)，および単一のコード生成部を持つコンパイラ群によって採用される技法をも代表している．最小限の機能を持つ極端に小さなプログラム以外のすべてのプログラムに対して，インタプリティブなコードの技法は，他の技法を使用する場合よりも小さな主記憶領域しか必要としない．いいかえれば大きさの固定した主記憶領域のもとでは，インタプリティブなコード技法の方が，より大きなプログラムを主記憶領域内に存在させることができる．インタプリタは，プログラムの大きさとは無関係に常に主記憶領域中に存在しているので，プログラムが大きいほどインタプリティブな技法の利点が発起される．

主記憶領域の節約の特徴は，MOVE 命令を例として考えれば十分である．ここで，MOVE 命令は，送出し側と受取り側との2つのオペランドの番地と，転記する大きさとしてパラメタ ($P_1 \cdots P_6$) 化された6バイトを伝えなければならないものとする．

1) サブルーチン：

C	A_1A_2	番地 A_1A_2 へのサブルーチン呼出し
$P_1P_2P_3P_4P_5P_6$		パラメタ・リスト

2) インタプリティブな目的コード：

F	関数演算子
$P_1P_2P_3P_4P_5P_6$	パラメタ・リスト

'C' (サブルーチン呼出し)に連結するには2バイトの番地を必要とし，インタプリティブな'F' (関数演算子)の場合にはこの2バイトは必要ではないことに注意されたい．したがって，サブルーチン呼出しの場合には全部で9バイト必要であり，インタプリティブな手法では7バイトしか必要でない．このことは，各々の結合に対して主記憶領域が1/4節約できることを示している．ある規模の大きさのプログラムでは，この性質はかなり重要で

ある．図6の必要な主記憶領域の比較に，この利点が要約されている．インタプリティブなコード技法に必要な主記憶領域にはインタプリタ自身も含まれる．一方インタプリティブでないサブルーチン呼出しのコード技法では，必要なサブルーチンの記憶領域および最小の実行ルーチンのみを含むインライン・コードの技法は，主記憶領域を非常にたくさん必要とするので，有効な代案とはみなされず，一般的にはほとんど採用されない．

これまでみてきたとおり，サブルーチン結合もまたプログラムの記憶領域の観点から不経済である．ある特別な理由によって，プログラムの大きさが重要な問題と考えられない場合でさえも，機械からの独立性，および“移植性”に対する必要性からインタプリティブな技法が必然的に採用される．

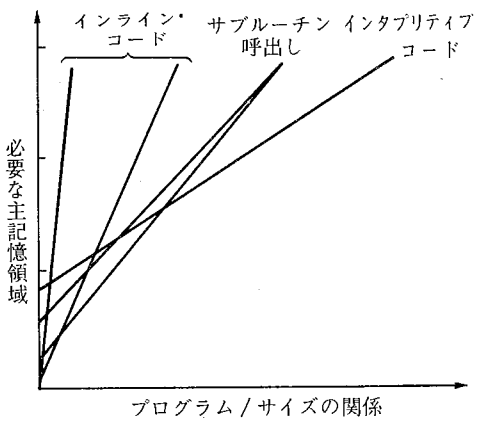


図 6 必要な主記憶領域の所要量
Fig. 6 Memory requirement comparison

6. おわりに

UTS 400 インテリジェント端末用の COBOL コンパイラ・プロジェクトの経験により，インタプリティブな技法の可能性ばかりでなく，単一の原始プログラムによる言語プロセッサの作成技法の概念の可能性も示した．結果として，各々のシステム上で効率的に働くプロダクトが生産された．

この経験によって“移植性を持つ”言語プロセッサの開発にかかる費用は従来の開発による費用と同程度であるけれども，後続のホスト・システムや目的のシステムの各々に対するソフトウェアは，通常と比べ非常に少ない費用で開発でき，維持できるということが示された．“移植性”は，言語の標準化と保守しやすさに関係するということも示された．“移植性を持つ”言語は，使用者プログラムの大きさと速さ，あるいは翻訳と実行に不利となる影響を与えないということが示され，人的資源・時間および費用を大幅に節約する可能性を開いた．

（プロダクト・サポート統括二部 池原 憲男 訳）

参考文献 [1] F. W. Stodola, “The PLUS Programming Language”, *ACM SIGPLAN NOTICES*, Vol. 15, No. 1, Jan. 1980, pp. 146-155. “システム記述言語 PLUS”, 技報, 日本ユニバック(株), 1981年2月, No. 0, pp. 11-19.

執筆者紹介 Ferdinand P. Mehrlich

1972年 Brigham Young 大学コンピュータ・サイエンス学科卒業．同年 Sperry Univac 社に入社．システムズ・プログラマに従事．UTS 4000 COBOL の移植作業プロジェクトのリーダを歴任し，現在，UTS 4000 テキスト処理ユーティリティ・グループのグループ・リーダ．

Steven M. Tanner

1967年より1年半，Hill 空軍基地においてコンピュータ・システム・アナリストとして従事．1968年より1972年まで，Burroughs の B 3500, B 4700 等の言語開発グループの一員として，COBOL の開発および保守に従事し，その間，1969年 Weber 州立大学管理工学科を卒業し B. S. を取得．1972年 Sperry Univac 社に入社後，UTS 400 COBOL および UTS COBOL プロジェクトのリーダとしてコンパイラやインタプリタの開発，UTS COBOL システムの設計に従事．

編集者注 UTS 400 COBOL は，日本ユニバック(株)から UTS COBOL として，シリーズ 1100 (OS 1100) とシリーズ 80 (OS/3) に対して提供される．

論説 プログラム・テストにおける神託の仮説について

The Oracle Assumption of Program Testing

E. J. Weyuker

要 約 プログラム・テストにおいてしばしば取りあげられる仮説に，“テストには神託が存在する”というものがある．すなわち，テスターまたは外部的機構は，プログラムが作り出した出力結果が正しいか否かをいつでも正確に決定できるという仮説である．本稿では，この仮説の妥当性を検討し，多くの場合，それは現実に即した仮説ではない，という結論に到達することを示す．さらに，この仮説を承認したときの影響を検討し，それに代わる考え方を考察する．

Abstract A frequently invoked assumption in program testing is that there is an oracle (i. e. the tester or an external mechanism can readily and accurately decide whether or not the output produced by the program is correct). The reasonableness of this assumption is examined and the conclusion is reached that in many cases this is not a realistic assumption. The consequences of operating under this assumption are examined and alternatives are investigated.

1. はじめに

プログラム・テストの方法論やテスト理論を研究するとき，標準的仮説として，テスターはテスト・データに基づいて作り出された出力結果が正しいか否かを正確に決定できると仮定する．もし，この仮説が真ならば，テストの神託(oracle)が存在するということにする(たとえば，参考文献[5],[12]を参照)．

本稿で考える主要な疑問は，この仮説が現実的なものであるか否かということである．もし，実際にこの仮説が現実的なものでないとすれば，神託に依存する結果どのような影響が生じるのか，またそれに代わるどのような代替案があるのかを決定しなければならない．

ところで，テストの神託の仮説を承認する基本的な理由が2つあるように思える．第1に，テストについての文献でとりあげられている多くのプログラムがあまりにも単純なものであることが，この仮説を現実的なものにしている．第2の理由は，それが簡単に人に認めさせられるものだからである．プログラム・テストの分野での最大の研究目標は，テストについての健全な理論的基礎を発展させること[3],[4],[11]と，プログラム中の誤りを検出したり，プログラムの正当性に関する確信を高める実践的な方法論を開発することである．もし，テスト結果の正否が決定できないとすれば，テストは合理的な工程であるとはいえないだろう．実際，これが本稿の結論ではない．しかしながら，神託の存在仮説が多くの問題の潜在的な根源であると気付くことが重要である．

2. 神託の存在仮説に対する反証

まず，最初に観察によるとテスターがテスト結果の正当性を正確に決定できるのは極めてまれである．テスターは結果がもっともらしいか否かを決めることができるように思えるが，もっともらしさ(plausibility)は，正当性(correctness)とは完全に同じではない．正

This article is reprinted, with permission, from the procceedings of 13 th Hawaii International Conference on System Sciences, (January 1980), pp. 44-49.

しい答がわからないことはよくあることである．もし，そうでなければ，プログラムを書く必要がなかったであろう．正当性ではなく，もっともらしさを受け入れるとするならば，テスターが共通の誤りのクラスを見逃すというような問題が生じる．たとえば，典型的なものには思い違い(off-by-one)による誤りがある．たとえば，Gellerのカレンダー・プログラム[2]のようなプログラムを考えてみよう．そのプログラムは，同じ年の特定の2日の間にある日数を求めるものである．いま，1943年の3月27日と5月5日の2日が与えられたとすると，プログラムは40を答えとして返す．確かに，この答はもっともらしい答えである．もし，プログラマが同時にテスターであり神託を与える者であれば，プログラムを書いたときに用いた計算式を適用してテスト結果を検証することであろう．その結果，彼は，自分が計算を間違いなく行う方法を知っていると考えてしまうのである．同じ意味で，もし，彼がうるう年の定義を間違えていれば，その誤解によって生じる誤りを見つけることはありえないであろう．

まさに，このような問題例がDeMillo, Lipton および Sayward の文献[1]に見られる．三角形の辺の長さを表す3つの正整数を分類するプログラムに対して，1組の数(14, 6, 4)はそのテスト・ケースの1つである．

この1組の数は，三角形を構成しない(3辺の長さは，三角不等式を満足しない)のに，そのプログラムは鈍角三角形に分類してしまう．プログラマやテスターがこの型の誤解をすると，それに基づく誤りはプログラミングの誤りを反映していないので，テストによってプログラムの問題点をあばくことが期待できなくなる．

テスターが，正しい答をはっきりと知らなくても，もっともらしくない結果を見つけられることが多い．たとえば，100ドルは大企業の総資産としてもっともらしい結果ではない．また，素人にとっては100万ドルはもっともらしい数字と考えられるが，企業の会計士のような専門家なら，100万ドルは誤りで，110万ドルがもっともらしいと決められる十分な情報を持っていることであろう．では，会計士は1,134,906.43ドルで正しいが，1,135,627.85ドルは誤りであると決定できるだろうか．多分，できないであろう．このように熟練者ですら，もっともらしいが誤った答を，正しい答と思い込んでしまうことが多い．経験や熟練は，もっともらしさの範囲を限定するときにのみ役に立つことが多いのである．

テスターが見つけることができるのは，もっともらしさであって必ずしも正当性ではないということは，出力が誤っているということは確信を持っていえるが，出力が正しいということは必ずしもいうことができないということと同じである．これは，部分的に“可解”であるが帰納的には“可解でない”決定問題の概念に対応している．帰納的関数論の停止問題を含む周知の多くの問題はこのカテゴリーに属する(上記の用語の定義や概念についての詳細は参考文献[7],[8]を参照)．さらに，Weyukerの文献[9],[10]には，プログラムの1つの文は意味的に到達可能(semantically reachable)か，分岐を通るデータが存在するか，またある特定の経路は通過できるか等のプログラムの多くの性質は，すべて部分的に可解であるが，帰納的に可解でない決定問題であることが示されている．

したがって，これらや多くの類似した結果によれば，少なくとも直観的なレベルでは一般に，ある結果が正しいか否かを決定できるということは驚くべきことである．つまり，神託がないということが，理論的な直観である．

さて，プログラムが多くの出力を作成するためにそれらのすべてを検討することが実際にはできないとき，異なる問題が発生する．そのようなテストの結果を検証するにはどのようにするのであろうか．典型的な方法では，テスターは出力をちょっと眺めて，それが

合理的に見えるかを調べたり，とくに誤りが起こりやすいことがわかっている部分を検証するであろう．ある意味で，これもまだ実際に正当性というよりはもっともらしさに依存している．細部をチェックしないで正しいと思えるということにより，テスターはもっともらしさによってテストを行っているのである．たとえ出力の一部分だけしかチェックしなかったとしても，誤りが発生しやすい個所が正しければ，残りの部分もまた正しいようである(すなわち，もっともらしい)と実際に認める．また，用いたデータの量が必ずしも出力を注意深く検査したか否かを決める際の決定要因ではないことに留意しよう．各行に10桁の数字を印字した1ページの出力のような相対的に少量のデータでも，完全にチェックするのは非常に難しく，まためんどうであるかも知れない．

神託の存在仮説の合理性に対する別の反例は，もっともらしさをも決められないプログラムのクラスに見られる．いま，πの1,000桁目の数字を求めるプログラムが欲しいものと仮定しよう．もし，そのプログラムが4と答えたらそれは正しい答であろうか．この問題に対しては，どんなタイプの直観が必要とされるのだろうか．もっともらしさは，具体的でないが，直観的には理解できるレベルで誤りを許容するような“正しさ”と考えることができる．しかし，上の例では，許容できる誤りの受認範囲はない．答えは，正しいか誤っているかのいずれかである．

3. 神託なしでプログラムをテストすること

では，神託の存在を仮定しない場合にはプログラムをどのようにテストすればよいのだろうか．それには，いくつかの可能なアプローチがある．

まず第1の，そして多分これが最良の方法であると思うアプローチは，別のアルゴリズムを用いてプログラムを書き直すことである．そして，それらのプログラムを同じデータで動作させて，それぞれの結果を比較するという方法である．この技法は，“もっともらしさ”を確かめたいときに適用できる．ただし，非常に多くの出力を作成するような課題のときには，それぞれの結果を比較するために3番目のプログラムを書かなければ，この技法が利用できないことは確かである．この方法には，いくつかの短所がある．その1つは，2つの意味で費用が高くつく点である．すなわち,ただ1つの仕事を行うために少なくとも2個の完全なプログラムの設計と実現とそしてデバイスの準備を行わなければならないし，さらに，コンピュータ利用面からもコスト高になる．それは，1組の結果を得るために2個のプログラムを動作させなければならないからである．

コスト高に加えて，さらに，1つの仕事を行うのに別の異なるアルゴリズムを実現する必要がある．この異なるアルゴリズムはどこから生まれてくるのか，通常，何かを行うためのアルゴリズムが1つだけでも見つかることは幸運なことである．それにもかかわらず，2つ目のアルゴリズムを見つけ，そのアルゴリズムが最初のものと異なっていることを示すのは容易な仕事ではない．

さて，神託の存在を仮定しないときのプログラム・テストに有効な2番目の方法は，その結果の正当性が正確に決定できるように“単純化した”データでプログラムを動作させてみることである．そのために，プログラムの一部を修正することが必要なこともあろう．

神託の存在仮説が現実的ではないことを示すプログラム例として，極めて詳細な出力や非常に大量の出力を作成するプログラムをあげることにしよう．そのような事例では，すべての出力をチェックすることは，たとえば不可能なことではなくても，困難なことであ

る．そのため，通常は出力の一部分の正当性かもっともらしさのいずれかを決めることになる．したがって，普通，これらのプログラムは多少の修正を行い単純なデータでより少ない出力を作成するようにできる．そして簡易化した出力の正当性からもとの出力の正当性を推定するのである．この技法は，入力のないプログラムに対してはとくに効果がある．なぜならば，そのような例では，入力するテスト・データの選択問題は無意味になるからである．たとえば10個のもののすべての順列を計算しリストするプログラムを作るものとしよう．そのような順列は3,628,800通りも存在するので，確かに手作業でチェックしたり数え直すには多すぎる量である．そこでプログラムをちょっと修正し，交換される要素の数をパラメタとして，4のような小さな数を入力してプログラムを動作させてみる．4個の要素の場合には，24通りの順列しか存在しないので，これらの結果の正しさを手作業で容易にチェックできるし，その結論からプログラムの正当性を類推することができる．

つぎに，整数 n と $n+1$ との間の0.0001きざみの10,000個の数を対象として，底を2とする対数表を作るプログラムを考えてみよう．これは，もっともらしい結果が容易に見つけられる例である．その結果が正しいか否かを判断するには，人間の能力にかかわる2つの課題がある．その1つは出力の量である．この課題は，n と $n+1$ との間のきざみ幅を0.1とする数の対数を作成するようにテスト・プログラムを修正することにより解決できる．その結果，チェックする結果は10,000個ではなく10個だけとなる．出力を簡易化する別の方法は，その結果の一部のみをプリントすることである．2つ目の課題は，3.0179は2を底とする8.1の対数の値として正しいか否かを判断することが容易ではないことから生じる問題である．$\log_2 8=3$ や $\log_2 16=4$ であることは既知であるから，$\log_2 8.1$ が3.0179であることは確かにもっともらしい．この場合のもっともらしさは，あるいくつかの固定点での厳密解の知識によって導きだされていることに注意しよう．さて，修正したプログラムに対しては10個だけの数値をチェックすれば良いので，それらの値を可能なかぎり正確に計画するように努めることは決して無理なことではない．このために，異なるアルゴリズムを用いて別のプログラムを書くこともありえよう．単純化したデータでプログラムを動作させる方策は実践面からの考察によって導かれる1つの解であるが，それは，しばしば，完全に満足できるものではないことに注意したい．プログラムが単純なデータでは正しく実行するが，大量のデータや境界値データに対しては誤った結果をもたらすことは珍しくはない．オーバフローが見落とされる誤りは，まさにこの種の誤りである．

神託が存在しないときのプログラム・テストに関する第3番目の考え方は，素直にもっともらしさを受け入れることである(これは一般に，すでに実行していることであろう)．そこで目標は，可能なかぎりもっともらしさの範囲を限定していくことである．

もっともらしさの範囲が非常に大きくなれば，事実上，どんな結果ももっともらしいものとなり，したがって，得られた結果の妥当性に関してはほとんど何も知ることにならない．たとえば，コンピュータ化された在庫システムにおいて50,000個までのねじは保有できることがわかっているならば，−4という在庫数は誤りであるということがわかる(すなわち，もっともらしくはない)し，さらに，上限が正しい数であれば，50,000を超える在庫数も誤りである．しかし，0と50,000との間の在庫数はもっともらしい数である．(もちろん，いくつかの値や部分範囲が他の値や部分範囲より，一層頻繁に出現することはある)．他方，許容値の範囲が0から10までの整数であれば，もっともらしさの知識が大いに役立つものとなるに違いない．

では，もっともらしさの範囲を限定するにどのようにすればよいのだろうか．数値計算の問題に関しては，その1つの方法は，いくつかの特定の入力に対する値を知るかあるいは計算することである．さらに，関数の振舞いについての知識があれば，もっともらしさの範囲を効果的に限定することが可能になる．たとえば，角度の正弦を計算するプログラムを仮定しよう．正弦のとりうる値の範囲は -1 と $+1$ の間であることはわかっていることである．さらに，0°，30°，45°，90°，135°，150°，180° 等の正弦値と正弦関数の曲線が図1のようになることが既知であるとする．

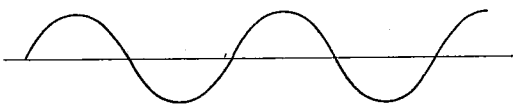


図1 正 弦 曲 線
Fig. 1 Sine curve

いま，42° の正弦を計算したいのであれば，その結果は -1 と $+1$ の間にあること以上の多くのことがわかる．まず，その値は，$\sin 30° = 0.5000$ と $\sin 45° = 0.7071$ の間の値であることがわかる．さらに，30° と 45° の間の曲線は凸型であるから直線近似によって 42° の正弦は 0.6657 より小さくなることはないということもわかる．

その上，30° と 45° の間では正弦曲線は相対的に“平坦”であるから直線近似は良くあてはまる．それゆえ，$\sin 42°$ の実際の値は 0.6657 に極めて近い値であると考えられるであろう．したがって，0.7050 という値はもっともらしい範囲に入るけれども，それは疑わしい値であると確信を持っていえるであろう．さらに，正弦曲線の振舞いやさらに正確な計算の結果によって，もっともらしさの範囲をさらに一層限定することができるようになる．

4. 神託の存在仮説の影響

神託の存在仮説がしばしば現実的なものではないという結論を導く証拠を議論し，テストに関する課題のいくつかの代替策とそれらの効果を調べてきたが，ここでは，神託の存在仮説を受け入れたときの影響を考えてみよう．

とくに言及し考察を加える価値がある2つの異なる状況がある．その1つの場合は，出力結果は実際に正しいのに，テスター/神託がそれは誤っていると決定するときである．これは，2つの状況のうちでは起こる度合が少ないほうである．この正しくない“神託”の決定からいくつかの影響が起こりうる．とにかくも，まず存在しない誤りを探しだすために時間を浪費することである．また，その神託により，無意味なデバッグを続けプログラムの引渡しに遅れることがあるならば，その時間分の損失を被ることになる．もちろん，デバッガーがプログラムの“誤りを直す”ために正しいプログラムを修正し，その結果誤りを作ってしまうならば，事はさらに重大である．

もう1つのより一般的なケースは，実際の結果は誤りであるがテスターや神託がその結果を正しいと思いこむ場合である．よく知られているように，テストや検証が行われて現場に引き渡されたほとんどのプログラムあるいは多くのプログラムは，誤りを依然として含んでいる．しかしながら，これは本質的には異なる状況である．一般に，不完全テスト (nonexhausitive testing) をしたときには，潜在的に誤りが存在していると考えられる．しかし，テストが行われ正しいものとして認められたプログラムは，実際に正しいことを期待されている．少なくとも，プログラムを動作させた特定のデータに対しては正しい結果

を出すと理解されるからである．しかし，そうでないならば，たとえば完全テストを行ったとしてもプログラム中に誤りが含まれていないと保証することはできない．

さらに，１つ強調すべき点がある．テストの専従者(すなわち，独立したテスティング・グループで働く人々)との議論によれば，多くのテストの文献で神託の存在を仮定しているにもかかわらず，テストの専業者は正しい結果とはどんなものかをせいぜい知っているだけで，正しい結果が何であるべきかについてはほとんど何も知らないように思われる．システム・ソフトウェアに関しては，普通，結果はそれ以前の改訂版から得られた結果と比較される．その場合，前の結果の正当性は一般に仮定される．しかし，テスターはそれらの大本の結果がどのように検証されたかについては，めったに考えてこなかった．また一般にテスティング・グループは，テスト結果が正しいこととか受け入れられることとかをどのようにまたはなぜ決定したかを，報告文書に記入しないように思われる．したがって，ソフトウェアの利用者は，一般に，ソフトウェアに関する有効な情報すべては入手していないので，偽りの安心観を与えられているのかも知れないのである．

一般に，ソフトウェア工学，とくにプログラム・テストの分野が進歩するにつれて，テストの妥当性についての計測法の開発に力点がおかれよう．

プログラムは，指示された仕事を満たすために書くだけではなく，プログラムが要求どおりに働くことを確証できるものでなければならないだろう．このことは，ハードウェアの製作者には，日常，要求されていることなのである．

さて，そのようなテストの計測が開発できるとすれば，プログラムについて示してきたことを厳密に述べることができなければならない．現在，その指標として用いられているものの１つは，プログラムの各分岐を通ることを要求するものである[6]．文献[3],[1],[11]を含む多くの人々は，なぜ，そのような測度がプログラム・テストの十分性の指標として不満足であるかを長々と議論している．しかしながら，その長所は明白であるということに議論の余地がある．証明してきたことを正確に述べることはできる．これまで何を示してきたかを，たとえば，“プログラムの３個所の分岐を除くすべての分岐を通した”とかあるいは“分岐の96パーセントを通した”というように述べることができる．しかし，これらの記述もわかっていることの完全に正確な記述ではない．このような記述では分岐を通過し，かつ“正しい結果を出した”という仮定が暗黙のうちに認められる．しかし，これまで議論してきたように，このことは一般に決定できることではない．それゆえに，妥当性に関するこの計測やその他の計測もまた，いままで議論してきたような本質的な欠陥を持っている．．したがって，テストの研究が進展するにつれて，また，理想テストの方法(一般に理想テストではない完全なテストは存在しないことは周知のことである)に向う．そこで，直面しなければならない２つの本質的な障害があることがわかる．その１つは，上で述べた非可解な決定問題の結果である．しかし，これらの結果は，大部分は理論的なものである．しかしながら，２つ目の障害は，ある意味でテストをプログラムに対して行う都度，直面する現実に即した実際的な問題である．われわれは，得られた結果が正しいか否かを問い，そして決定できなければならない．このことは，テスターが直面しなければならない本質的な制約であると信じる．

(教育部　樋川 和伸 訳)

参考文献 [１] R. A. DeMillo, R. J. Lipton and F. G. Sayward, “Hints on Test Data Selection : Help for the practicing Programmer”, *Computer*, Vol. 11, No. 4, Apr. 1978, pp. 34-41.

[2] M. Geller, "Test Data as an Aid in Proving Program Correctness", *Comm. ACM*, Vol. 21, No. 5, May 1978, pp. 368–375.
[3] J. B. Goodenough and S. L. Gerhart, "Toward a Theory of Testing: Data Selection Criteria", in *Current Trends in Programming Methodology*, Vol. 2, ed. R. T. Yeh, Prentice-Hall, 1977, pp. 44–79.
[4] W. E. Howden, "An Evaluation of the Effectiveness of Symbolic Testing", *Software-Practice and Experience*, Vol. 8, 1978, pp. 381–397.
[5] W. E. Howden and P. Eichhorst, "Proving Properties of Programs from Program Traces", in *Tutorial: Software Testing Validation Techniques*, eds. E. Miller and W. E. Howden, IEEE Computer Society, 1978, pp. 46–56.
[6] J. C. Huang, "An Approach to Program Testing", *Computing Surveys*, Vol. 7, 1975, pp. 113–128.
[7] Z. Manna, *Mathematical Theory of Computation*, McGraw-Hill, 1974.
[8] H. Rogers Jr. *Theory of Recursive Functions and Effective Computability*, McGraw-Hill, 1967.
[9] E. J. Weyuker, "Program Schemas with Semantic Restrictions", Ph. D. Thesis, *Dept. Comp. Sci. Tech. Report DCS-TR-60*, Rutgers University, New Brunswick, N. J., June 1977.
[10] E. J. Weyuker, "The Applicability of Program Schema Results to Programs", *Int. J. Comput. Inf. Sci.*, Vol. 8, No. 5, 1979.
[11] E. J. Weyuker and T. J. Ostrand, "Theories of Program Testing and the Application of Revealing Subdomains", *Dept. Comp. Sci. Tech. Report 008*, Courant Institute of Mathematical Sciences, New York University, New York, Feb. 1979.
[12] L. J. White, E. I. Cohen and B. Chandrasekaran, "A Domain Strategy for Computer Program Testing", *Comp. and Info. Sci. Res. Center Tech. Report*, The Ohio State University, Columbus, Ohio, Aug. 1978.

執筆者紹介 Elaine J. Weyuker

New York 州立大学 Binghamton 校 (Harpur College) より数学で A. B. を，Pennsylvania 大学より計算機科学で M. S. E. を，Rutgers 大学より計算機科学で Ph. D. をそれぞれ取得．1968 年から 1969 年まで IBM でシステム・エンジニアとして勤務．1969 年から 1975 年まで New York 州立大学 Richmond 校で計算機科学の講師を務める．1977 年より New York 大学の Courant Institute of Mathematical Sciences の助教授．研究分野はプログラム・テスト検証，プログラム図式，アルゴリズムの設計と解析等．

論文

プログラム・テストの理論と等質な部分領域の応用

Theories of Program Testing and the Application of Revealing Subdomains

E. J. Weyuker, T. J. Ostrand

要 約 Goodenough と Gerhart によるテスト・データ選択についての理論を吟味する．この理論を拡大し洗練するために，等質なテスト基準と部分領域の考えを提案する．この概念はプログラム・テスト構築の基礎を与えるのに使うことができる．

プログラムの入力データ領域のある部分領域において，“その領域で正しく処理されない入力が1つでも存在するならば，その領域のすべての入力は正しく処理されない”場合に，この部分領域を“等質である (revealing)”という．この考えの意図するところは，入力領域を部分領域に分割して，その同値クラスのデータがどれも正しく処理されるか，またはどれも正しく処理されないかの，どちらかになるようにすることにある．そこで，実際のテストでは，各クラスの中から勝手に1つのデータを選ぶことによって構成する．こうすれば，この処理によって完璧にプログラムがテストできる．実用的なテスト方策としては，起こりやすいと考えられる誤りを暴露するような等質な部分領域に分割することとする．

ここでは，いくつかの文献で取りあげられた3つのプログラムを使って本稿で開発した考え方でテストしてみる．

Abstract The theory of test data selection proposed by Goodenough and Gerhart is examined. In order to extend and refine this theory, the concepts of a revealing test criterion and a revealing subdomain are proposed. These notions are then used to provide a basis for constructing program tests.

A subset of a program's input domain is revealing if the existence of one incorrectly processed input implies that all of the subset's elements are processed incorrectly. The intent of this notion is to partition the program's domain in such a way that all elements of an equivalence class are either processed correctly or incorrectly. A test set is then formed by choosing one element from each class. This process represents perfect program testing. For a practical testing strategy, the domain is partitioned into subdomains which are revealing for errors considered likely to occur.

Three programs which have previously appeared in the literature are discussed and tested using the notions developed in this paper.

1. はじめに

プログラムのテストに関する理論の主要な目標は，実際のテスト方法に対する基礎を与えることと，それぞれのテストでの誤りの検出の効力を示す尺度の決定方法を確立することである．

Goodenough と Gerhart[5] はテスト理論のための基本的な定義を与え，プログラムの入力データ領域からテスト・データを選び出す基準について議論している．その中で“プログラムの構造のみをもとにして選ばれたテスト・データでは，十分なテストにはならない”ことがわかりやすく説かれており，“理想テスト”をプログラムの入力仕様と出力

仕様に関係させて定義している．プログラム F の**理想テスト**とは次の条件を満たすテスト・データの集合 $T=\{t_i\}$ である．

『T の中に F によって正しく処理されないデータ t_i があるとき，かつ，そのときにかぎり，F の入力データ領域の中に，正しく処理されないデータ d がある．』

理想テストの結果がすべて正しいということは，そのプログラムの正しさの証明をしたことに等しい．

本稿ではまず，2章と3章で彼らの理論が，現実のテストを構成する目的にどれくらい有用であるかを調べてみる．また，いくつかの問題を指摘して，それらの救済策を提示する．

4章では，1つの仕様に合うように作成された任意のプログラムについても，彼らの理論を変形してそのテストの基礎を与える．しかし，このアプローチは，実用的な成果があるとは立証できない．

つぎに5章では，それまでに認識された問題に対処するために，等質なテスト選択基準と等質な部分領域の概念を導入する．6章では，この考え方に基づいて，3つのプログラム例について，そのテストを構成してみる．等質な部分領域の考え方は，テスト全般についての見通しを与え，有意義なテストの選択を行ううえで有用である．

2. Goodenough と Gerhart の定義

記法

F：プログラム．

D：入力領域．

R：出力領域．

$F(d)$：入力 $d\in D$ に対して，F は(停止するならば)出力 $F(d)\in R$ を生成する．

$\mathrm{OUT}(x, y)$：F の出力仕様．$x\in D$，$y\in R$．

$\mathrm{OK}(d)$：$F(d)$ が存在し $\mathrm{OUT}(d, F(d))$ であれば，F は入力 d に対して正しい．これを $\mathrm{OK}(d)$ で表す．

T：テスト・データの集合．$T\subseteq D$．

C：テスト選択基準．T を規定する条件である．

訳注：論文[5]では，テスト T が基準Cに合致している条件を，$\mathrm{COMPLETE}(T, C)$ で表している．

定義

(1) **SUCCESSFUL**

テスト T が**成功である** (SUCCESSFUL).

$\Longleftrightarrow$ $(\forall t\in T)(\mathrm{OK}(t))$．すなわち，

$\mathrm{SUCCESSFUL}(T)=(\forall t\in T)(\mathrm{OK}(t))$

(2) **RELIABLE**

選択基準 C が**信頼できる** (RELIABLE).

$\Longleftrightarrow$ C によって選ばれたすべてのテストは成功であるか，あるいは，すべて成功ではない．すなわち，

$\mathrm{RELIABLE}(C)=(\forall T_1, T_2\subseteq D)((\mathrm{COMPLETE}(T_1, C)\wedge\mathrm{COMPLETE}(T_2, C))$
$\supset(\mathrm{SUCCESSFUL}(T_1)\equiv\mathrm{SUCCESSFUL}(T_2)))$

(3) **VALID**

選択基準 C が**妥当である** (VALID).

$\Longleftrightarrow$ プログラム F が正しくない場合には，C によって選ばれたテストの中には，成功でないテスト T が少なくとも1つ存在する．すなわち，

$$\mathrm{VALID}(C)=(\forall d\in D)(\neg\mathrm{OK}(d)\supset(\exists T\subseteq D)(\mathrm{COMPLETE}(T,C)\wedge\neg\mathrm{SUCCESSFUL}(T))$$

以上の定義から，次の基本定理が得られる．

定理

基準 C が信頼でき，かつ妥当であるならば，C によって選ばれるテストは，理想テストである．すなわち，

$$(\exists T\subseteq D)(\exists C)(\mathrm{COMPLETE}(T,C)\wedge\mathrm{RELIABLE}(C)\wedge\mathrm{VALID}(C)\wedge\mathrm{SUCCESSFUL}(T))\supset(\forall d\in D)(\mathrm{OK}(d))$$

さて，この理論を応用する場合には，いくつかの問題がある．第1にプログラム中の誤りをあらかじめ知ることはできないので，選択基準の信頼性と妥当性が保証されるのはその基準が，テストとしては入力領域全体を選択するものであるときで，かつそのときにかぎられる．このような，全データによる徹底したテストは現実には不可能である．そこで，5章で潜んでいると思われる誤りに応じて領域を分割する方法を論じる．

第2に，上の定義はすべて，ある1つのプログラムに対して定められていることに起因する問題である．一般に，テストの内容は，そのプログラムの振舞いに依存することになるし，プログラムの修正によって基準の妥当性も信頼性も，デバッグの過程では保存されない．

さらに別の問題として，妥当性と信頼性が独立な性質ではないことがあげられる．この問題については3章で議論する．

この理論を使うとき，テスト設計者は理想テストのための基準を見つけようとするだろう．しかし，プログラムの中の誤りを知らないで，あるいは，少なくとも誤りがある特定の種類のものであることを知らないでは，基準Cが理想的かどうかを判断することはできない．5章と6章では，この問題を取りあげて，基準の性質を特定の誤りに関連させた考え方を導入する．これによると，テストの成功とは，プログラムの正しいことを結論づけるものではなく，特定の誤りが存在しないことを意味するものになる．

3. 実際のテストの考察

ここでは，実際のテストとデバッグにおける理想テストの有用性を調べてみる．たいてい，1つのプログラムは次第に洗練・改良・変形され，多くの版（version）がある．次の版が正しいという確信を深めるためには，各版の正しさを検証するのが望ましい．しかし，1つの版の理想テスト基準は，別の版にとって必ずしも理想基準ではない．

たとえば，整数 d を入力とし，

$$F(d)=(d*d)+3$$

$$\mathrm{OK}(d)\longleftrightarrow[(d+d)+6=F(d)]$$

というプログラムの場合，$\mathrm{OK}(3)$ および $\mathrm{OK}(-1)$ であるが，他のすべての d については $\neg\mathrm{OK}(d)$ であるから，

$$C(T)\longleftrightarrow(T=\{t\}\ \wedge\ t\in\{0,1,2\})$$

は妥当で，かつ信頼できる基準となる．いま，このデータによるテストの結果，定数の誤

りに気づいて,

$$F(d)=(d*d)+6$$

と修正すると，OK(0) および OK(2) で，他の d については ¬OK(d) となり，基準 C はもはや信頼できるものではない．

さらに，逆に

$$C'(T)\longleftrightarrow(T=\{t\}\ \wedge\ t\in\{-1,1,3\})$$

は，当初のプログラムに対しては信頼できるものではないが，妥当となる．しかし，一部修正したプログラムに対しては信頼できるものであり，かつ妥当である．

訳注：当初のプログラムでは

$F(d)=(d*d)+3$

OK(d) $\longleftrightarrow$ [$(d+d)+6=F(d)$]

である．

$t=-1$ ⇨ $F(-1)=(-1)*(-1)+3=4$

OK(-1) $\longleftrightarrow$ [$((-1)+(-1))+6=4=F(-1)$] ∴ OK(-1)

$t=1$ ⇨ $F(1)=4$, OK(1) $\longleftrightarrow$ [$8=4$] ∴ ¬OK(1)

$t=3$ ⇨ $F(3)=12$, OK(3) $\longleftrightarrow$ [$12=12$] ∴ OK(3)

であるから

$C'(T)$ $\longleftrightarrow$ [$T=\{t\}\ \wedge\ t\in\{-1,1,3\}$]

で，-1 と 3 は OK であるが，1 は OK でない．

¬RELIABLE(C'). よって，C' は RELIABLE でない．

一方，C' によって選ばれたテストの中に ¬OK(1) となる 1 があり，$T=\{-1,1,3\}$ は SUCCESSFUL でない．

¬SUCCESSFUL(T). よって，C' は VALID でない．

一方，修正したプログラムでは

$F(d)=(d*d)+6$

OK(d) $\longleftrightarrow$ [$(d+d)+6=F(d)$]

$t=-1$ ⇨ $F(-1)=7$, OK(-1) $\longleftrightarrow$ [$4=7$] ∴ ¬OK(-1)

$t=1$ ⇨ $F(1)=7$, OK(1) $\longleftrightarrow$ [$8=7$] ∴ ¬OK(1)

$t=3$ ⇨ $F(3)=15$, OK(3) $\longleftrightarrow$ [$12=15$] ∴ ¬OK(3)

したがって，すべての $t\in\{-1,1,3\}$ について ¬OK(t) である．そこで，C' で選ばれたテストはすべて成功でない．よって C' は RELIABLE である．

また，これは成功でないテストであるから，C' は VALID である．

このように，基準の持つ性質は，デバッグの過程で保存されない．それは常に増加するかあるいは保存されるとか，または常に減少するかあるいは保存されるという意味での“単調性”を持ってはいない．

基準はテスト・データの特性を規定するが，入力領域のどのデータも，基準によって選ばれる可能性を持っているべきである．何ら入力データを排除しない基準は妥当である[5]．ただし，領域中のあるデータを，テストの対象から排除したとしても，まだ妥当性を保つことはできる[5]．

基準が妥当であっても，その信頼性やプログラムの正しさについてはなにもいえない．

われわれの最初の定理は，信頼性と妥当性とが独立ではないことを指摘するものである．

定理 1: $(\forall C)(\mathrm{VALID}(C)\vee\mathrm{RELIABLE}(C))$

なぜなら，妥当でないテスト基準によって誤りは検出されないから，妥当でないテスト基準によって選択されるすべてのテストは成功である．よって，この定理は成り立つ．

実際，基準が妥当でなかったと仮定すると，次のことが成り立つ．

1) プログラムには誤りが存在する．

かつ，

2-1) その基準を満たすテストはどれも成功する．

または

2-2) その基準は信頼できる．

訳注：VALID(C) の定義を使えば，

$$\neg\mathrm{VALID}(C)$$
$$\equiv\neg(\neg(\exists d\in D)[\neg\mathrm{OK}(d)\wedge\neg(\exists T\subseteq D)(\mathrm{COMPLETE}(T,C)\wedge\neg\mathrm{SUCCESSFUL}(T))]$$
$$\equiv(\exists d\in D)[\neg\mathrm{OK}(d)\wedge(\forall T\subseteq D)(\neg\mathrm{COMPLETE}(T,C)\vee\mathrm{SUCCESSFUL}(T))]$$

である．そこで

1) $\neg\mathrm{OK}(d)$ となる $d\in D$ が存在する．つまり誤りが存在する．

かつ

2) すべての T について $\mathrm{SUCCESSFUL}(T)$ であるか，

または，

$\neg\mathrm{COMPLETE}(T,C)$ である．

ところで，$\mathrm{RELIABLE}(C)$ の定義からは，$\neg\mathrm{COMPLETE}(T,C)$ ならば，常に $\mathrm{RELIABLE}(C)$ である．

2番目の指摘は，少なくともテストの観点からは，妥当でない基準によるテストは，やっても何も得られないことを意味している．

4. 一様に妥当な基準と一様に信頼できる基準

Goodenough と Gerhart の理想テストの性質が，プログラム構造テストと同様，テストされるプログラムに依存することは，すでにみた．これに対して，基準の妥当性と信頼性が出力仕様のみに依存するようにできれば，この基準は，与えられた仕様を満たすように作られたどんなプログラムに対しても普遍なものになる．これはブラック・ボックス・テストとしてよく知られている．次に示す**一様に妥当**と**一様に信頼できる**ことの定義では，プログラムFを明示変数として含んでいる．

定義

・基準Cが**一様に妥当である** (uniformly valid)

$$\Longleftrightarrow\ (\forall F)[(\exists d\in D)(\neg\mathrm{OK}(F,d))$$
$$\supset(\exists T\subseteq D)(C(T)\wedge\neg\mathrm{SUCCESSFUL}(F,T))]$$

・基準Cが**一様に信頼できる** (uniformly reliable)

$$\Longleftrightarrow\ (\forall F)(\forall T_1,\forall T_2\subseteq D)$$
$$[(C(T_1)\wedge C(T_2))\supset(\mathrm{SUCCESSFUL}(F,T_1)\equiv\mathrm{SUCCESSFUL}(F,T_2))]$$

ある出力仕様に対して**一様な理想基準**とは，一様に妥当でかつ一様に信頼できるものである．この定義は，論文[5]の定義に固有のプログラムへの依存性を解消するであろうが，この概念は重大な弱点を持っている．つまり，任意のプログラムに対して，ある意味で自明でない一様な理想基準は存在しない．

いま，基準 C で選んだすべてのテストの和集合が入力領域Dに等しいとき，Cを**自明に妥当である**ということにする．

自明に妥当な基準は明らかに妥当である．しかし，自明に妥当でない基準Cは任意に与えられた出力仕様に対して一様に妥当とはなりえない．なぜならば，Cのテストのどれにも含まれないような要素dに対して，dについては正しくなく，$D-\{d\}$ のすべての要素については正しいようなプログラムを書くことができるからである．したがって，次の定理を得る．

定理 2: 基準 C は一様に妥当である.
$\Longleftrightarrow$ 基準 C は自明に妥当である.

信頼性についても同様な弱点がある. すなわち,

定理 3: 基準 C は一様に信頼できる.
$\Longleftrightarrow$ 基準 C はただ1つのテストを選ぶ.

証明: Cがただ1つのテストを選べば明らかにどのプログラムにとっても信頼できる. Cが違うテスト T_1 と T_2 を選び, t は T_1 のデータで, T_2 には含まれないとする. T_2 のすべて入力に対しては正しく, t に対しては正しくないプログラムを書くことができる. そこで, この2つのテストはこのプログラムに対して違う結果を出すから, Cは信頼できない. □

定理2と3とから, 次の系を得る.

系: 基準 C は一様に妥当でかつ一様に信頼できる.
$\Longleftrightarrow$ 基準 C はただ1つのテスト $T=D$ を選ぶ.

こうして一様妥当性と一様信頼性は, 非実用的な徹底テストのみによって検証できることになる.

次の結論は, 系を変形したもので, よく引用される『テストは誤りの存在を示すことができるだけで, 誤りがないことを示すことはできない』[3]ことを表している.

定理 4: $(\forall C)(\forall T)$
$[(C(T) \wedge T \fallingdotseq D) \supset (\exists F)(\mathrm{SUCCESSFUL}(F,T) \wedge \neg \mathrm{SUCCESSFUL}(F,D))]$

証明: $t \notin T$ なるデータ t があるので, プログラム F を T に対して正しく, t に対してのみ正しくないものにすればよい. □

他にも否定的な結果が示されている. Weyuker はプログラム中の特定の文, 分岐あるいは経路が実行されるかどうかも, またはすべての文, 分岐あるいは経路が実行されるかどうかも, 決定できるアルゴリズムは存在しえないことを証明している[12][13].

Howden[7] は, 任意のプログラム F と出力仕様に対して, F が T について正しいならば, D についても正しいといえる, 空でない有限集合 $T \subseteq D$ を作る手続きは存在しないことを示した. この決定問題の非可解性に対して, 彼はテスト生成の手続きが成功するようなプログラムのクラスを見つけることで対応している.

DeMillo らの論文[2]で述べられているように, プログラマは "できるだけ正しいものに近いようプログラムを作る…(彼ら)はどういう種類の誤りをおかしやすいかについて, おおむねよく知っている. (また)プログラムを詳細に調べる能力も機会も持っている". テスト理論は, よく発生することが知られている誤りのクラスを有効に利用すべきであり, それら誤りの存在をあばく, あるいは存在しないことを保証することのできるようなテスト選択基準の本質を捕えるべきである.

5. 等質なテスト選択基準

ここでは, テストについての見方を正しさから, "準正しさ" とでもいうようなところへ下げる. つまり, ある特定の誤りの存在をあばいたり, その誤りは起こらないことを示

す方法をとる．そして具体的方法論というよりは，テストへの接近法を概論したい．

テスト基準 C が，入力領域の部分集合 S について等質である，ということを次のように定義する．

定義：REVEALING (C, S)

$$\Longleftrightarrow \quad [(\exists d \in S)(\neg \mathrm{OK}(d)) \supset (\forall T \subseteq S)(C(T) \supset \neg \mathrm{SUCCESSFUL}(T))]$$

部分領域 S について等質な基準によって選ばれるテストは，S について理想テストである．

等質な基準には2つの特殊なケースがある．1つは，S が D に等しいケースである．

定理 5：REVEALING $(C, D) \Longleftrightarrow$ VALID $(C) \wedge$ RELIABLE (C)

証明：C が等質であるならば，D 中に誤りになるデータがあるときは，C によるテストはすべて成功ではなくなる．したがって，C は信頼でき，かつ空ではないから，妥当となる．逆に C が妥当であり，かつ信頼できるものとする．そのとき，もし誤りが存在すれば，C によるテストはすべて成功でない． □

2つ目の特殊なケースは，前の定義で，基準 C を取り除いた場合である．この場合には，S 中のあるデータに対して誤りが起こることが，S の任意のデータに対して誤りが起こることを意味するのであれば，S は**等質な部分領域**であるという．この概念が，われわれのテストへの接近法の基礎である．直観的には，等質な部分領域というのは，プログラム上で同じように処理されており，かつ仕様上でも同じように処理されるようになっているデータの集まりである．入力領域をこのようなデータ集合に分割する方法は，経路テスト[7]，機能テスト[9]それと特異値テスト[6]とを結合したものである．

ところで，S が等質であることを示すことは（S が十分小さくて，その中のすべてのデータについて実行してみることができるなら別であるが），S においてのプログラムの正当性を証明することと同等である．そこで通常可能なことは，S が，特定の誤りに対しては等質であることを示すことにとどまる．

S がある**誤り E に対して等質である**とは，誤り E がプログラム中に存在し，S 中のあるデータ i に対して発生する $(\neg \mathrm{OK}(i))$ ならば，S のどのデータ d に対しても $\neg \mathrm{OK}(d)$ になる場合である．このような領域 S のデータ t 対して $\mathrm{OK}(t)$ であっても，それは考えていた特定の誤りが存在しないことを示すにすぎない．逆に $\neg \mathrm{OK}(t)$ は，その特定の誤りが存在することを必ずしも保証しているわけではない．また，個々の誤りに対して等質であっても，2つ以上の誤りが同時に発生することに対しては，その領域がそのまま等質であるとはかぎらない．

われわれの目標は，入力領域をプログラムに独立な属性とプログラムの構造上の性質の両方に基づいて分割し，同様によく発生すると認識されている誤りの分類を行うことである．後の点は，ある意味で突然変異テスト[2]の考え方に通じるものである．そこでの最も起こりやすい誤りについての認識が正しいものか，あるいはまた，そのような分類が個々の課題から離れて抽象的に定義できるものなのかについては議論の余地があるが，起こりそうな誤りが存在しないことを示そうという目標は，適切である．

Howden[8] は記号テスト法とその誤り検出効果についての研究の中でいっている．“実データによるテストで誤りを発見できるのは，プログラマが運良くそのようなデータを選んだときである”．

だから，等質な部分領域の考えは，この問題に対する部分解である．実際，もしプログラムの入力領域が等質な部分領域に分割できたとすれば，このように盲目的なテスト・データの選択を行う必要はない．領域中のデータは，どれをとっても誤りになるか，どれをとっても正しく処理されるからである．

6. 等質な部分領域の構築

入力領域をまず経路分割する．経路分割した各領域のデータは，プログラム上の同じ1つの経路，あるいは同じ経路集合の中の1経路に従って処理されるものである．2番目の分割は問題分割である．これは仕様，アルゴリズムおよびデータ構造等のプログラムとは独立のソースに含まれる性質の共通性に基づく分割である．経路分割と問題分割の交わりは，等質な部分領域の性質を持っている．

例 1：すでにいくつか研究例がある[2],[11] 三角形分類の問題を取りあげる．問題の仕様は以下のとおりである．

入力：3つの正の整数 A, B, C, $(A \geqq B \geqq C)$.

出力：A, B, C が満足する記述は，次のどれであるかを示す．

1) それらは三角形の辺をなさない．
2) それらは正三角形の辺である．
3) それらは正三角形ではないが二等辺三角形の辺である．
4) それらは直角不等辺三角形の辺である．
5) それらは鈍角不等辺三角形の辺である．
6) それらは鋭角不等辺三角形の辺である．

図1は参考文献[2]に記されているプログラムの流れ図である．この問題は，目的が入力領域の分割であり，またプログラム中に繰返しもないので，等質な部分領域の応用にはもってこいである．

図2は，このプログラムの経路分割による6つの領域を表す条件を記している．

仕様に基づく分割，つまり問題分割によっては，8つの領域が認められる．

S_1: $(A<B) \vee (B<C)$ (不正なデータ)
S_2: $(A \geqq B+C) \wedge (A \geqq B \geqq C)$ (三角不等式を満足しない)
S_3: $A=B=C$ (正三角形)
S_4: $A=B>C$ (二等辺三角形)
S_5: $(A>B=C) \wedge (A<B+C)$ (二等辺三角形)
S_6: $(A>B>C) \wedge (A^2=B^2+C^2)$ (直角不等辺三角形)
S_7: $(A>B>C) \wedge (A^2<B^2+C^2)$ (鋭角不等辺三角形)
S_8: $(A>B>C) \wedge (A^2>B^2+C^2) \wedge (A<B+C)$ (鈍角不等辺三角形)

図3は経路分割と問題分割の交わりから得られる9つの空でない部分領域を示している．テスト・データは，各領域から任意の要素を選べばよく，図4はその例である．そこには，データに対する正しい出力と，実際のプログラムの出力とを併せて記してある．領域 D_2 と D_3 については結果が正しくない．不正な入力であるのに，特定の三角形の辺であるように出力している．D_2 と D_3 は，この誤りに対して等質なのである．

違うプログラムに対しては，違う領域分割になる．たとえば，図1で1と2の判定を各々 "$A \neq C$" および "$A<C$" と書き間違がえたとする．これでは，$A>B>C$ なる入力

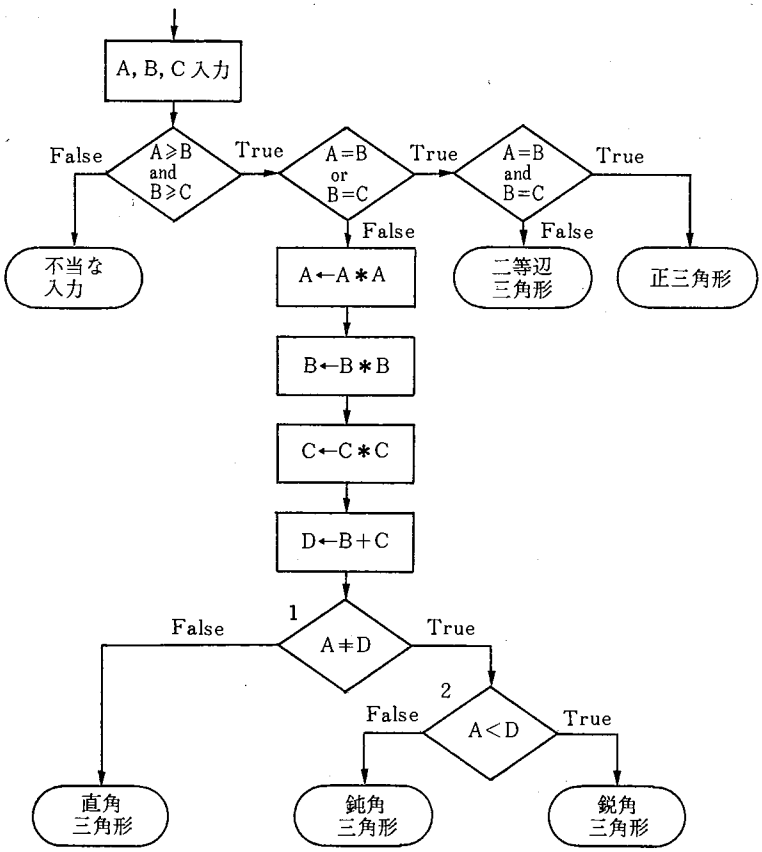


図 1　三角形分類問題の流れ図

Fig. 1　Flowchart for triangle classification problem

P_1: $\neg[(A \geqq B) \wedge (B \geqq C)]$, すなわち, $(A<B) \vee (B<C)$

P_2: $(A>B>C) \wedge (A^2=B^2+C^2)$

P_3: $(A>B>C) \wedge (A^2>B^2+C^2)$

P_4: $(A>B>C) \wedge (A^2<B^2+C^2)$

P_5: $(A \geqq B \geqq C) \wedge [(A=B) \vee (B=C)] \wedge \neg[(A=B) \wedge (B=C)]$,
すなわち, $(A=B>C) \vee (A>B=C)$

P_6: $(A=B=C)$

図 2　三角形分類プログラムの経路分割領域の条件

Fig. 2　Path domain conditions for triangle classification program

$D_1=S_1 \cap P_1$: $(A<B) \vee (B<C)$

$D_2=S_2 \cap P_3$: $(A \geqq B+C) \wedge (A>B>C)$

$D_3=S_2 \cap P_5$: $(A \geqq B+C) \wedge (B=C)$

$D_4=S_3 \cap P_6$: $(A=B=C)$

$D_5=S_4 \cap P_5$: $(A=B>C)$

$D_6=S_5 \cap P_5$: $(A>B=C) \wedge (A<B+C)$

$D_7=S_6 \cap P_2$: $(A>B>C) \wedge (A^2=B^2+C^2)$

$D_8=S_7 \cap P_4$: $(A>B>C) \wedge (A^2<B^2+C^2)$

$D_9=S_8 \cap P_3$: $(A>B>C) \wedge (A^2>B^2+C^2) \wedge (A<B+C)$

図 3　問題分割と経路分割の交わりによって作られる部分領域

Fig. 3　Subdomains formed by intersection of problem partition and path domain partition

領域	テスト・データ	正しい出力	実際の出力
D_1	(1, 2, 3)	不正な順	不正な順
D_2	(14, 6, 4)	三角形でない	鈍角三角形
D_3	(2, 1, 1)	三角形でない	二等辺三角形
D_4	(1, 1, 1)	正三角形	正三角形
D_5	(2, 2, 1)	二等辺三角形	二等辺三角形
D_6	(3, 2, 2)	二等辺三角形	二等辺三角形
D_7	(5, 4, 3)	直角三角形	直角三角形
D_8	(6, 5, 4)	鋭角三角形	鋭角三角形
D_9	(4, 3, 2)	鈍角三角形	鈍角三角形

図 4 三角形分類プログラムのテスト・データと出力
Fig. 4 Test data and output for triangle classification program

P_1': $(A<B)\vee(B<C)$
P_2': ϕ
P_3': $(A>B>C)$
P_4': ϕ
P_5': $(A=B>C)\vee(A>B=C)$
P_6': $(A=B=C)$

図 5 正しくない三角形分類プログラムの経路分割領域の条件
Fig. 5 Path domain conditions for incorrect triangle classification program

は，すべて鈍角不等辺三角形として出力されることになる．このプログラムの経路分割は図5のようになり，問題分割との交わりによる領域の中には，

$$P_3'\cap S_6=S_6,\quad P_3'\cap S_7=S_7,\quad P_3'\cap S_8=S_8$$

が含まれる．この3つの部分領域は，いま考えている誤りに対して等質である．

ところで，図1のプログラムは，入力(−3, −4, −5)に対しては鋭角不等辺三角形として出力する．領域分割をする上で，入力はすべて正整数であると仮定したが，そう仮定する理由は，$A\geqq B\geqq C$ の仮定をする理由と同様に，確かではない．不当な入力には，2つの違う状況があることを認識すべきである．第1は，入力の形式が正しくない状況であり，第2は正しい形式ではあるが，必要な性質を持っていない状況である．第2のタイプの不当な入力は，正当な入力と同様に取り扱われるべきである．すなわち，その不当性を決めるためには，入力操作以上の何らかの処理が必要なのである．

例 2: これは正しさの証明ができるくらい単純であり，多くの人が，証明している[1],[10]．問題は，x が整数，y が非負整数であるときに x^y を計算するものである．

図6は，計算のアルゴリズムを示す．

入力(x, y)に対してとられる経路は，y の値でのみ決まることに注意する．図7は経路と，関連する y の値を示している．この経路分割領域の合併は，y の全領域である．

つぎに，問題の仕様とアルゴリズムを分析して，x と y の領域を分割する方法をさがす．x については，明かに，正，負およびゼロの3つの場合に分けられる．さらに，$x=1$ と $x=-1$ は，出力の正しさの確認が容易であるから取りあげてみる．そして，x の領域は，5つのクラスに分割される．

y については，4つの特別な値がある．$y=0$ は，結果が常に1になるべきである．$y=$

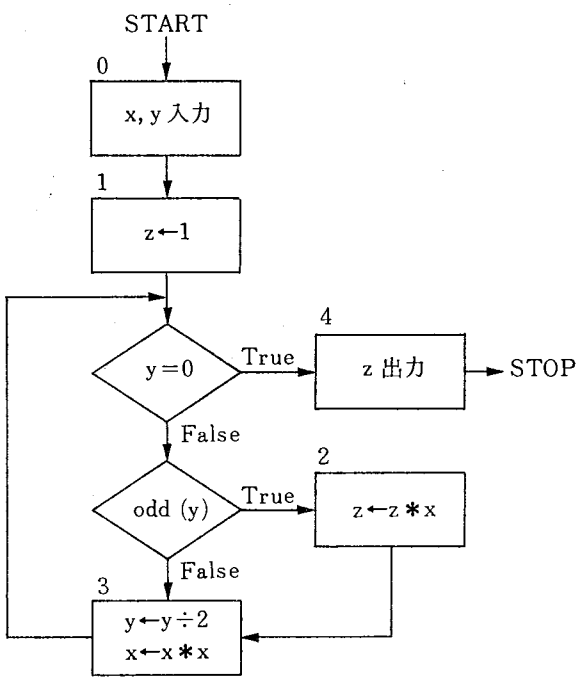


図 6　累乗計算プログラムの流れ図
Fig. 6　Flowchart for fast exponentiation program

経路分割	経路表現	経路をとる y の値	y の 2 進表現
P_1	014	0	0
P_2	013^n234, $n\geqq 1$	2^n	10^n
P_3	$01(23)^n4$, $n\geqq 1$	2^n-1	1^n
P_4	$0123(3+23)^a3(3+23)^b234$	2^n-1を除く奇数	$1(0+1)^c0(0+1)^d1$
P_5	$013(3+23)^a23(3+23)^b234$	$0, 2^n$を除く偶数	$1(0+1)^c1(0+1)^d0$

図 7　累乗計算プログラムの経路分割領域
(経路表現は，図 6 の流れ図の箱につけた番号を使った正則表現)
Fig. 7　Path domains chosen for exponentiation program

X 入力クラス	Y 入力クラス
X_1: $x<-1$	Y_1: $y=0$
X_2: $x=-1$	Y_2: $y=2^n$, $n\geqq 1$
X_3: $x=0$	Y_3: $y=2^n-1$, $n>1$
X_4: $x=1$	Y_4: $y=1$
X_5: $x>1$	Y_5: y は 2^n-1を除く奇数
	Y_6: y は 2^n, 0 を除く偶数
部分領域: $X_i\times Y_j$, $i=1,\cdots,5$; $j=1,\cdots,6$	

図 8　累乗計算プログラムの部分領域
Fig. 8　Subdomains for exponentiation program

1 は，プログラムが誤っていても，x^y の値は計算できる．$y=2^n$ の場合，z が使われるのは最後の繰返しだけである．$y=2^n-1$, $n>1$ も一般的でない．すべて z にためこまれる．この 4 つのケースと，他の正整数の領域を考慮に入れて，y の問題分割ができる．

図 8 は，x と y についての経路分割と問題分割の交わりを示している．x の入力値は経路に関係しないので，入力領域の分割は，x の問題分割と同じである．

つきに，考えられるいくつかの誤りについて，部分領域の等質性を調べてみる．

誤り 1：z の初期値が 0.

誤り 2：判定 $y=0$ を $y\neq 0$ と書く．

誤り 3：判定 odd(y) を even(y) と書く．

誤り 4：判定 odd(y) がなく，常に $z\leftarrow z*x$ を行う．

誤り 5：$z\leftarrow z*x$ を $z\leftarrow z+x$ と書く．

誤り 6：$y\leftarrow y\div 2$ を $y\leftarrow y-2$ と書く．

これら誤りのうち，誤り 4 を除いては，それぞれの誤りに対して，どの部分領域も等質であることがわかる．誤り 4 の場合は，$y=0$, $x=0$, $x=1$, $y=2^n-1$ および $x=-1$ かつ y が奇数の 5 つのケースを除いては結果が正しくなくなる．この誤りの存在で経路分割が変わり，部分領域は，図 8 の Y_1, Y_2, Y_3, Y_4 とそれら以外の Y_7 とに分けられる．そうすると，$X_2\times Y_7$ だけは等質でないことがわかる．なぜならば，y が奇数のときは結果が正しく，偶数のときは結果が正しくないからである．ただし，この領域が等質でなくとも，他の部分領域によってこの誤りは暴露されるであろう．

> **例 3**：図 9 のプログラムは Geller[4] から採ったもので，同じ暦年中の 2 つの日付の間の日数を求める問題である．入力は，(month 1, day 1) と (month 2, day 2) の 2 つの組と 1 つの year である．1 番目の日付が 2 番目の日付よりも前の日であることと，入力はすべての点で妥当であることが仮定されている．

Geller の論文[4]中のプログラムには，文法の誤りと論理の誤りが存在しているが，図 9 では文法の誤りは修正されている．2 番目の if 文の述語の誤りはそのままにしてある．

入力データの構造，期待される出力，それに入力に対するプログラムでの扱い方を調べることによって，入力領域の部分領域は，以下のようになる．

S_1：month 1=month 2

うるう年計算が含まれるか否かを考えることによって，さらに分割が進む．

S_2：month 1>2 かつ month 2>month 1+1

S_3：month 1≠2 かつ month 2=month 1+1

S_4：month 1=1 かつ month 2>2

S_5：month 1=2 かつ month 2>2

うるう年の計算の違いによって，S_4 と S_5 はまた分割される．

Y_1：year rem 4=0 かつ year rem 100≠0

Y_2：year rem 100=0 かつ year rem 400≠0

Y_3：year rem 400=0

Y_4：year rem 4≠0

このプログラムの場合，典型的な誤りとして，1 つずれる誤り，つまり出力が，正しい値 ±1 であるとか，1 月分の日数だけ違うようなものを考えてみる．この種の誤りに対しては，部分領域はどれも等質である．また，2 番目の if 文の誤りに対しては，S_4, S_5 と

```
procedure calendar (integer value day 1, month 1, day 2, month 2, year);
  begin
    integer days;
    if month 2=month 1 then days:=day 2−day 1
      comment if the dates are in the same month, we can compute
              the number of days between them immediately;
    else
      begin
        integer array daysin (1 : 12);
        daysin (1):=31; daysin (3):=31; daysin (4):=30;
        daysin (5):=31; daysin (6):=30; daysin (7):=31;
        daysin (8):=31; daysin (9):=30; daysin (10):=31;
        daysin (11):=30; daysin (12):=31;
        if ((year rem 4)=0) or
          ((year rem 100)=0 and (year rem 400)=0)
          then daysin (2):=28
          else daysin (2):=29;
        comment set daysin (2) according to whether or not
                year is a leap year;
        days:=day 2+(daysin(month 1)−day 1);
        comment this gives (the correct number of days-
                days in complete intervening months);
        for 1:=month 1+1 until month 2−1 do
          days:=daysin (1)+days;
        comment add in the days in complete intervening months;
      end;
    write (days)
  end;
```

図 9　カレンダー計算プログラム
Fig. 9　Calendar computation program

もに等質である．すなわち $S_4 \cap Y_i$ あるいは $S_5 \cap Y_i$，$i=1,3,4$ の入力は，どれも，2月の日数を正しく作らないので，出力も正しくない．$S_4 \cap Y_2$ と $S_5 \cap Y_2$ の入力はすべて正しい結果を与える．

以上のように，等質な部分領域の考えがタイプの異なるいろいろなプログラムに適用できるものであることを示してみた．このテスト法は，プログラムの目的が入力領域の分割や，それに基づく計算にある場合に最も良く適用しやすいようにみえる．三角形の問題とカレンダの問題がそうであった．しかし，この方法は，全入力領域について一様に処理すべき問題のプログラムにも効果的なのである．2番目の累乗計算プログラムがその例で，そこでもデータの構造と，そのプログラムでの取り扱い方に基づいて分割した領域が，誤り検出に効果があることを調べた．

7. おわりに

本稿では Gooenough と Gerhart の理論を吟味し，この理論を容易には応用できないものにしている問題点を認識した．そこで実用上の困難を除くために，この理論に対して2つの変形を試みた．まず第1は，結局彼らの理論のブラック・ボックス・テスト版に帰するもので，この考えも実用にならないことを示した．

第2の変形は，等質な領域の概念を導入したことによって，妥当性と信頼性の間の依存性を除いたことである．これは，考える領域をより小さなより扱いやすい部分に分けることを可能にした．

さらには，等質性の意味を，誤りに関連させることによって，テストの目標をプログラムの正しさを示すことから，特定の誤りがないことを示すことに修正した．

等質な基準および等質な部分領域の考えを広範な問題とプログラムに適用していくためには，問題分割をより体系的あるいは形式的に行う方法を見つけることが必要である．これは，プログラム自体を作ることと同じ仕事といえるから，形式化も容易なことではない．

もう1つ考えなければならない点は，大きなプログラムへの適用の可能性である．そのためには，経路分割のための自動化ツールがより洗練され，大規模プログラムにも使えるようになることが求められる．それによって，問題分割と経路分割の交わりによるより良い領域分割を得ることができて，大規模プログラムに対する等質な部分領域による接近がより適用可能になるだろう．

理論的な面では，どんな条件の下でなら，起こりうるどんな誤りに対しても等質でありうるかを問いたい．このことは，どんな部分領域についても可能であろうか．例外なく等質な領域を得ることができないにしても，どれだけ近づくことができるだろうか．問題は，現実に到達することのできる良いテストの持つ特質に応じて，どの程度の正当な信頼を，そのテスト結果におくことができるかを見つけることである．等質な部分領域の概念は，この目標に少し近づけてくれる．

最後に，S. Gerhart, R. Hamlet, K.-C. Tai, P. Abrahams, と多くの助言をくれたレフリーに感謝する．

（システム開発統括部 プログラミング技術部 長谷川 邦夫 訳）

参考文献

[1] R. M. Burstall, "Program proving as hand simulation with a little induction," in *Proc. IFIP Congr. 74.* Stockholm, Sweden: North-Holland, 1974, pp. 308–312.

[2] R. A. DeMillo, R. J. Lipton, and F. G. Sayward, "Hints on test data selection: Help for the practicing programmer," *Computer*, Vol. 11, Apr. 1978, pp. 34–41.

[3] E. W. Dijkstra, "Notes on structured programming," in *Structured Programming*, O.-J. Dahl, E. W. Dijkstra, C. A. R. Hoare, Ed. New York: Academic, 1972, pp. 1–81.

[4] M. Geller, "Test data as an aid in proving program correctness," *Comm. ACM*, Vol. 21, No. 5, May 1978, pp. 368–375.

[5] J. B. Goodenough and S. L. Gerhart, "Toward a theory of testing: Data selection criteria," in *Current Trends in Programming Methodology*, Vol. 2, R. T. Yeh, Ed. Englewood Cliffs, NJ: Prentice-Hall, 1977, pp. 44–79.

[6] W. E. Howden, "An evaluation of the effectiveness of symbolic testing," *Software-Practice and Experience*, Vol. 8, 1978, pp. 381–397.

[7] W. E. Howden, "Reliability of the path analysis testing strategy," *IEEE Trans. Software Eng.*, Vol. SE-2, Sept. 1976, pp. 208–215.

[8] W. E. Howden, "Symbolic testing and the DISSECT symbolic evaluation system," *IEEE Trans. Software Eng.*, Vol. SE-3, July 1977, pp. 266–278.

[9] R. E. Keirstead and D. B. Parker, "On the feasability of formal certification," in *Program Test Methods*, W. Hetzel, Ed. Englewood Cliffs, NJ: Prentice-Hall, 1973, pp. 291–301.

[10] Z. Manna, *Mathematical Theory of Computation.* New York: McGraw-Hill, 1974.

[11] C. V. Ramamoorthy, S. F. Ho, and W. T. Chen, "On the automated generation of program test data," *IEEE Trans. Software Eng.*, Vol. SE-2, Dec. 1976, pp. 293–300.

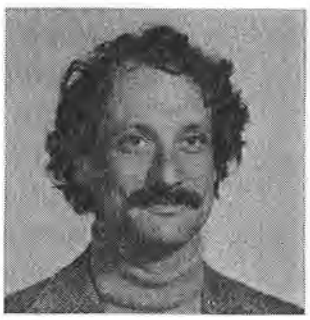
[12] E. J. Weyuker, *Program Schemas with Semantic Restrictions*, Ph. D. dissertation, Dep. Comput. Sci., Rutgers Univ., New Brunswick, NJ, Tech. Rep. DCS-TR-60, June 1977.

[13] E. J. Weyuker, "The applicability of program schema results to programs," *Int. J. Comput. Inform. Sci.* Vol. 8, Oct. 1979, pp. 387–403.

執筆者紹介 Elaine J. Weyuker

New York 州立大学 Binghamton 校 (Harpur College) より数学で A. B. を，Pennsylvania 大学より計算機科学で M. S. E. を，Rutgers 大学より計算機科学で Ph. D. をそれぞれ取得．1968 年から 1969 年まで IBM でシステム・エンジニアとして勤務．1969 年から 1975 年まで New York 州立大学 Richmond 校で計算機科学の講師を務める．1977 年より New York 大学の Courant Institute of Mathematical Sciences の助教授．研究分野はプログラム・テストと検証，プログラム図式，アルゴリズムの設計と解析等．

Thomas J. Ostrand



MIT より数学で S. B. を，Pennsylvania 大学から計算機科学で M. S. E. と Ph. D を取得．1970 年から 1971 年に RCA 社でプログラミングに従事，1971 年から 1978 年に Rutgers 大学の計算機学部で教鞭をとる．1978 年 9 月より，Sperry Univac 社のソフトウェア研究グループの Senior Computer Scierntist.

研究分野は，セル・オートマトン，プログラムの検証，プログラムの同値変換，有限状態オートマトンを用いた構文解析等．現在，Sperry Univac 社のソフトウェア工学研究グループの Research Manager で，ソフトウェアの検証とテスト，ソフトウェアの設計法，アルゴリズムの解析等に関心を持っている．

コンピュータ利用の教育

細井　正

1. コンピュータ利用の教育：CBE

CBE (Computer Based Education) という用語は，1961年 Illinois 大学の Dr. Bitzer らによって PLATO システムの研究開発プロジェクトが発足する時点から提唱されてきた用語である．より具体的には，学習者がコンピュータと対話形式で個別学習をする CAI (Computer Assisted Instruction) と，教師がコンピュータを利用して学習指導上の必要なデータを作成する CMI (Computer Managed Instruction) とに分けることができる．

LSI 技術の発展に裏付けされたマイクロコンピュータの出現は CAI に2度目のチャンスを与えていると R. Sugarman は述べているが，これまでの研究開発の中から技術的側面に注目してその概要を述べる．

2. CAI でのコースウェアの開発技術

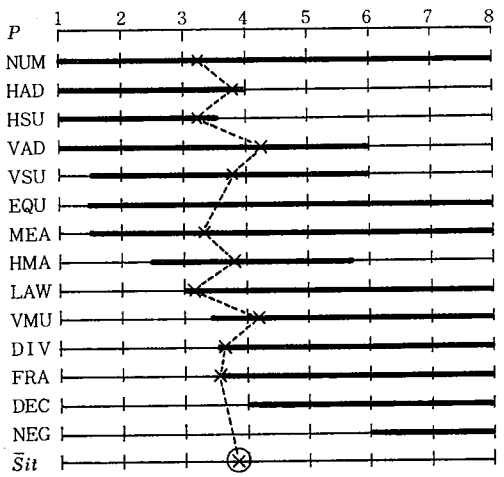
学習者がコンピュータと対話しながら個別学習をするということは，学習者の知的能力，学習履歴，学習特性等の幅広い多様性を考慮に入れた適切な教授，学習活動を展開できるようにすることである．このために一般の情報処理システムと違って，CAI システムにはハードウェア，ソフトウェアの他にコースウェアの開発技術が重要な要素となる．

コースウェアは当初，プログラム学習の理論を適用して，学習者に提示される教授内容とその系列を学習目標の分析結果に基づいて構造化する手法をとり，反応・反応系列の評価—診断—処方，さらには学習者が与えられた教授項目に対する応答方法や時間制御が，コンピュータ・プログラムによって制御されるプログラム制御様式のものが多く開発された．その後，これに対して，学習者は自らの自主的・主体的な選択によって学習内容を自由に選び，自分の考えに従って応答し，コンピュータによって評価されるのでなく，自己評価を行って学習がすすめられるようになった．これは，V. C. Bunderson らによって提唱された学習者制御様式 (Learner Control Mode) で，成果を上げていった．

いずれの場合も，コースウェアの作成者が考えられる範囲内であらかじめ設定した状況に応じた展開が行われるという意味で，学習コースが固定的な構造を持ったコースウェアであるといえる．

一方，コンピュータを利用して学習者に提示する問題を自動的に生成しようとする研究が，Stanford 大学の P. Supper らによって行われ，学習者の能力に応じた難易度の学習問題を生成・提示する，ストランド構造 CAI ドリル・コースウェアが開発された．現在 CCC 社 (Computer Curriculum Corp.) から一般に提供され，欧米で広く使用されている算数や言語教育の教材がある．

算数ドリルの場合，ストランド (Strand) とは同一演算操作の型に属する問題系列を指し，一連の同じタイプの問題群から構成されるものである．具体的には図1に示すように数概念，横書きたし算，縦書き引き算等でこの場合は14のストランドから構成されている．図1の横軸は，学年段階尺度 (Grade Placement Scale) で，学習者の進度・達成水準を



p：学年段階尺度
NUM：数概念　HMA：横書き掛け算
NAD：横書きたし算　LAW：四則の演算法則
HSU：横書き引き算　VMU：縦書き掛け算
VAD：縦書きたし算　DIV：割り算
VSU：縦書き引き算　FRA：分数
EQU：式(等式・不等式)　DEC：小数
MEA：測定　NEG：負の数
$\bar{S}it$：平均学年段階値

太い実線（ドリル項目の分布しているところ）
×印（生徒の学習進歩における到達水準 $S_{it}(j)$）
⊗印（生徒の到達水準を示す平均学年段階値 $\bar{S}_{it}$）

図1　ストランド構造の算数カリキュラムの全体の構造

評価するための成績評価と，ドリルの問題項目を構造化して系列的に配置するための共通の尺度として用いる．コンピュータで自動生成して学習者に提示される問題の決定は

① 各ストランドにおける達成水準 ($S_{it}(j)$, $\bar{S}_{it}$) と成績

② 異なった難易度のドリル項目を混ぜて，反復的に訓練する混合・復習方略の採用

③ 学習者自身のペースによる学習

という3つの教授方略 (Instructional Strategy) にもとづき，項目抽出関数によって操作的に実行される．

生成的なコースウェア開発技術のもう1つの例は，人工知能的アプローチによる情報構造志向型 (Information Structure Oriented—ISO) CAI である．J. R. Carbonel や J. S. Brown らは，事実に関する知識を符号化するための意味ネットワークと効率的な推論機構を導入して，学習者に提示するテキスト，質問，学習者からの応答入力に対するフィードバック等を自動的に生成する CAI を研究し，実験的に南米の地理を教える SCHOLAR や，電源装置 (Power Supply) の故障修理技能を教える SOPHIE を開発した．また，医学教育の分野で最近注目を集めている MYCIN もこのようなアプローチの成果である．

コンピュータを使ったゲームやシミュレーション技術を適用したコースウェア開発も進展している．大学教育の中で物理学や化学実験の CAI コースが開発され，利用されているし，医学教育ではシミュレーション・プログラムをマイクロ化して内蔵した教育機材が市販されている．

3. CMI における学習評価

CMI を効果的に実施するには，教授・学習過程での評価に関して，どのような手法があり，それに必要な情報はどの時点でどのような形式で提供されなければならないか，という観点から教育活動を見ていなければならない．

学習評価について，B. S. Bloom は教育目標の分類学 (Taxonomy) に立脚し，教授・学習活動の開始時点で，選抜や位置づけのためになされる評価を事前的評価，教授・学習活動の過程で，適切な処方を得るためになされる評価を形成的評価，1つの単元の終了時・学期学年・学習コースの終了時点において全体的成果をみるものを総括的評価と，それぞれの評価活動を区別し，CMI の枠組を与えた．

これら3つの学習評価の技術的側面での理論に関し，従来の相対評価は実行手順を裏付ける理論的研究が十分にあったうえでの学習評価への利用であった．しかし，新しく注目を集めている到達度評価の理論，すなわち目標準拠測定(Criterion Referenced Measurement) は実践的技術としての手順が確立されたとは，まだいえない状況である．

評価に当たってのテスト項目の作り方やデータ分析法も CMI の重要な課題である．アナライザーという学習機器の普及とともにデータ分析法が活発に研究され，理論的には整理された成果としてS-P表による分析法がある．S-P 表とは，「N 人の学習者の n 個の問題に対する反応とその正・誤により1あるいは0と採点し，問題と学習者をそれぞれ正答率の高い順に並べかえた得点一覧表」である．この表から図2に示すように学習者の得点累積分布曲線を示すS曲線と，問題の正答者数累積分布を示すP曲線の2つの独立した曲線が得られ，同時にP曲線とS曲線とのずれを示す面積，差異量Dが得られる，S曲線とP曲線が一致しているとき，すなわち差異量Dが小さいとき問題が等質であり，それが大きいときは，非等質であると考えられる．

差異量Dを正規化した数値を差異係数とし，差異係数の大きさによって問題の異質性を調べることができる．またS-P表の周辺度を基準変量とし，それと入れ替わりのない完全な反応パターンとの共分散の比を1から減じた値を注意係数といって，ある特定の学習者が異質であるかを調べることができる．このようなことから，学習指導の一貫性や等質性を計量的に分析することができる（図2）．

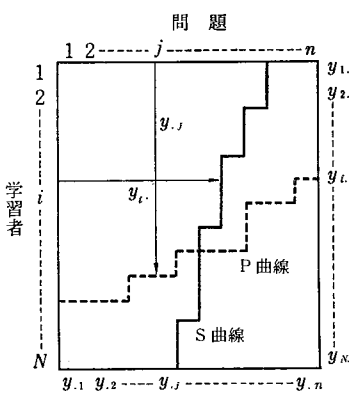


図 2 S-P 表の図表示

4. 今後の展望

もともと非常に技術化しがたい教授・学習過程のいろいろな問題解決に，コンピュータを利用しようというのであるから，CBE ほど情報処理技術分野に過大な要求をしている例は少ないといえる．CBE をより発展させるうえで，強く要求されている主な

関連技術をあげると，

① ヒューマン・インタフェース：音声や文字等のパターン認識の技術に基づいた，よりスムーズな交互作用の構築.

② 人工知能・知識工学：自然言語理解や高度な推論機能による問題解決への自然な対話

③ プログラム言語：SMALLTALK, LOGO等に見られる目的志向的言語による感応的なシステムの構築.

④ ローカル・ネットワーク：Distance Educationという言葉で示されるように分散処理志向が今後重要な課題となってくる.

⑤ データベース：多目的なデータ・ファイルは低価格で質的・量的にも十分なものが利用できることが大きなポイントとなっている.

⑥ 日本語入出力：英数字のキー・ボードからの脱却が強く要求されている.

等があり，CBE の発展は情報処理技術の発達とともに期待されていくであろう．このことは第5世代コンピュータに関する中間報告の中にも随所に見られる事柄である.

参考文献

[1] J. R. Carbonell, "計算機利用による教育への人工知能的アプローチ", 淵 一博監訳「人工知能の基礎」, 近代科学社, 1978, pp. 383-416.

[2] J. S. Brown and R. R. Burton, "教授システムにおける知識の多表現について", 淵 一博監訳「人工知能の基礎」, 近代科学社, 1978, pp. 287-322.

[3] 東 洋他,「教育のプログラム」情報科学講座 E. 17. 2, 共立出版, 1977.

[4] 佐藤隆博編著,「CMI システム—教育におけるコンピュータ利用」, (社)電子通信学会, 1976.

[5] E. A. Feigenbaum and W. J. Clancey, "知識工学—その方向と目標", 数理科学, 1981, 4, pp. 11-20.

[6] J. A. Chambers and J. W. Sprecher, "Computer Assisted Instruction: Current Trends and Critical Issues" in *Comm. of the ACM*, June 1980, Vol. 23, No. 6, pp. 332-342.

[7] R. Sugarman, "A Second Chance for Computer Assisted Instruction", *IEEE SPECTRUM*, Aug. 1978, pp. 29-39.

(教育技術開発室)

プログラムレス・ツール

山岸史明

1. はじめに

コンピュータが社会に受け入れられ，普及するにつれ，世の中のシステム化の要求はますます増大している．この要求の増大は，データ処理部門には

① システムの改善・保守に費用がかさむ.

② ソフトウェアが大きく複雑なものになり，新規開発に着手できない.

③ 適用業務が複雑・高度化し，高度な技術が求められる.

等の悩みをもたらした．また一方，利用部門には

① プログラムの開発・引き渡しに時間がかかる.

② 要求どおりにできあがらず，変更もむずかしい.

③ 非定型な業務が多く，機械化がむずかしい.

④ 出力帳表が見にくく，すぐ紙の山になる.

等，不満の遠因となっている.

これらの悩みや不満の解決策の1つとして，利用者部門が自ら直接コンピュータを利用し，業務を機械化しうる環境と機能を提供する方法がある．現在，このようなプログラムレスの利用者指向ツールが強く求められている.

2. あるプログラムレス・ツールの導入例

A社はプリント基板等の電子部品を製造している会社であるが，その工場のスタッフ部門の人々は，ほとんどの作業を紙と鉛筆そして膨大なコンピュータ・リストによって毎日の仕事を処理してきた.

彼らは，オンラインによる生産管理のアプリケーションの開発をデータ処理部門に依頼したが，開発費用・期間ともに満足のいく答が得られなかった．そこで，彼らはある利用者指向のプログラムレス・ツールを検討し，その導入を決めた．このソフトウェアのスタディ期間中からデータ処理部門の専門家を加えずに，スタッフ部門のみでアプリケーションを開発した.

最初の利用者は4名であったが，その後数百人まで拡大し，他のアプリケーションにもこのソフトウェアの採用が広がっていった.

各種レポートは，このアプリケーションを介し，現場担当者によって短時間に作成されるようになり，その結果，それまでの作業手順の無駄も見直され，多くの進捗上の問題点にも改善が加えられた.

従来，利用部門の人々は，強大なコンピュータ・システムを自ら命令して使うことなど，想像もしなかった．それが1週間程度の教育で現実のものとなったため，彼らはさらに適用分野を広げ，製造現場以外でも積極的に採用していった．

じつは，これは米国の GTE 社における Sperry Univac 社の MAPPER 1100 の導入事例である．

表 1 MAPPER 1100 適用事例

Monitoring the use of test equipment
Materials management
Shipping
International marketing
International contract reports
Diagnostic reports
Workload analysis
Inventory reports
Breakdown reports
Analysis of breakdowns by type and cause
Engineering changes
Inventory costs
Production information
Customer delivery reports
Problem investigation analysis
Shop floor analysis for expeditors
Subassembly tracking
Reports on department capacity of production
Production scheduling
Machine monitoring reports
Expediting systems
Shop floor despatching and control
Component shortage reports
Vendor analysis
Purchase order reports
Work-in-progress tracking
Performance analysis
Open order reports
Customer complaint reports and analysis
Where-used reports
Lead-time analysis
Identification of slow-moving stock
Tool control
Personnel resource control
Test equipment control
Machine load log
Quality reports and analysis
Contract sales orders
Changes in orders
Master parts stock status
Labor reports
Customer information
Budgets
Period cost counts
Product line planning
Prototype development support
Capital expenditure tracking
Order entry
Order picking, packing and shipping
Invoice preparation
Pricing information
Reports to aid forecasting
Salesmen commission payments
Overtime analysis
Customers' returned goods analysis
Transport cost reports
Engineering change order control
Printed wiring card assembly despatching.

表1は GTE 社の利用者部門が，その後にデータ処理部門の助けを借りず独力で開発していったアプリケーションの一覧表である．

また，図1は MAPPER 1100 を採用してシステム開発をした場合と従来のシステム開発の場合の工

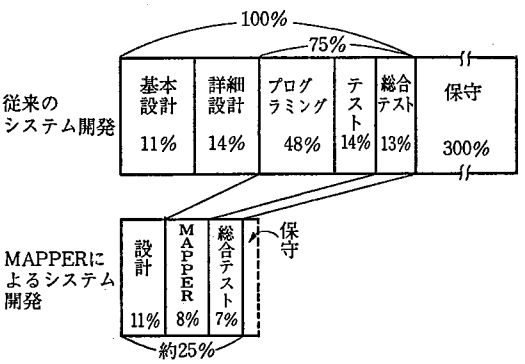


図 1 MAPPER の効果

表 2 MAPPER 1100 の主な機能

1) レポートの作成，削除	
・レポートの形式定義	GEN
・レポートの登録	AR
・レポートの削除	DR
・レポートの複写	XR
・レポートの保存	REP
2) レポートの表示，印書，その他の出力	
・レポートの画面表示	RID とタイプ
・外部媒体への出力	AUX
・印書装置への出力	PR
・穿孔装置への出力	PUNCH
・メッセージ交換	SS(メッセージ)
	SEND(レポート)
3) レポートの更新	
・新ラインの挿入	▷]n＋
・ラインの複写	▷]n×
・ラインの削除	▷]n－
・ラインの内容変更	▷～」
・サーチ結果のライン内容	UPD
変更およびライン削除	CHG
4) レポートの検索	
・ラインの検索－1	FIND
・ラインの検索－2	S, SU(SEARCH)
・ラインの検索－3	LOC(LOCATE)
5) レポートの分類，照合	
・レポートの分類	SORT
・レポートの照合	MA/MAU(MATCH)
6) レポートの計算	TOT (Totalizer)
7) コントロール・ファンクション	
・サイン・オン	ステーションを MAPPER に結合する (READY状態)
・サイン・オフ	X
・モードの選択	M
・レポート・タイプの表示	T
・パッチ・ランのスタート	START
・レポート・ジェネレーション	RUN
8) その他のファンクション	
・日数計算	DATE
・一般計算	A (FORTRAN タイプ)
9) 画面操作 (LINE 0)	
レポートの任意の場所を選んで表示する機能	
LINE, FMT, RL, SHFT, HLD CHRS, HLDIN	
報告書作成機能（ラン・ファンクション）	
以上のファンクションに加え，判断 (IF)，制御の移動 (GO TO) 等のコマンドを使用してパラメータ形式により処理手順を登録し，実行させる機能．	

程期間の比較である．開発段階だけでも，約1/3に工程が縮小でき，保守面においても開発工程同様に大幅に削減されている．

3. プログラムレス・ツール「MAPPER 1100」の機能と特徴

MAPPER 1100は，UNIVACシリーズ1100のオペレーティング・システムのもとで動くソフトウェアであり，端末を使いながら会話形式で処理を行う．その機能には，汎用のファイル管理と定型・非定型のレポート作成がある．実際にMAPPER 1100を使用する場合には，プログラムを組む必要がなく，単にデータ・フォーマットの定義を行い，データを入力することによって各種機能（レポートやファイルの検索・更新・編集や，データの加工・出力・表示・計算・分類・照合等，図2，表2参照）が使える．もちろん，定型的な処理手順は，あらかじめ登録しておいて，RUN としてそれを呼び出すこともできる．さらに，最近とみに重要な機能として注目されている電子メーリング機能やセキュリティ機能も充実している．さらにデータとして日本語も扱え，グラフ機能も計画されている．

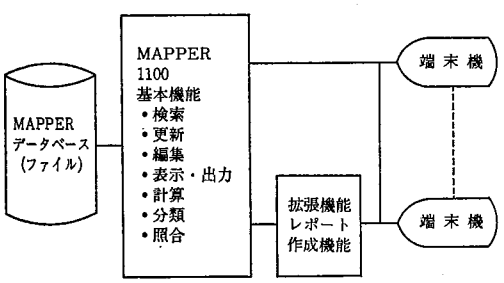


図 2 MAPPER 1100 の概要

なお，全体の構成は図3のとおりである．

最後に，MAPPER 1100 の特徴をまとめると，

① プログラムレスである．
② 汎用データベース管理機能を持つ．
③ 会話型レポート作成処理である．
④ 豊富なコマンド群（約80）を持つ．
⑤ データとして漢字が扱える．
⑥ オフィスのペーパーレスに寄与する．
⑦ メーリング機能を持つ．
⑧ 導入が容易である．
⑨ リアルタイム処理でタイムリに情報が入手できる．
⑩ 外部ファイルの移植性がある．
⑪ デマンド機能を持つ．

となる．

4. おわりに

ますますコンピュータ利用の方向は，利用者指向のシステムへと向っていくものと思われる．これらの動きの背景としては，

① 新規開発業務の増大
② その保守量の増大
③ 即応性の要求
④ オフィス作業の生産性の向上(OA 対応)

等があげられる．この動向に対応するには，利用者部門の人々が，データ処理部門の手を煩わさずに，簡単にコンピュータを使い，自分たちの業務を直接システム化することができるソフトウェアの開発が考えられ，良質のプログラムレス・ツールの提供が，今後さらに効率の良いシステム化の道を開くことになる．

（第一事業統括部　事業企画部）

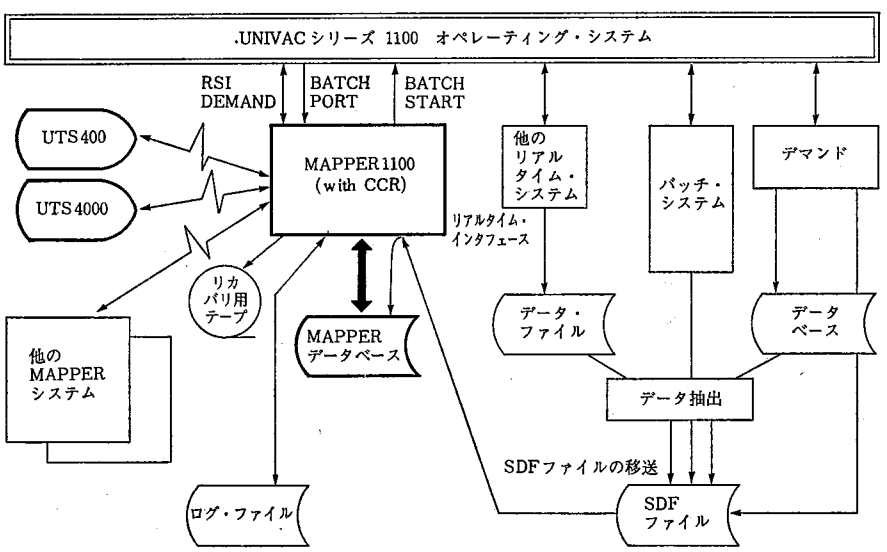


図 3 MAPPER 1100 の全体構成図

菅野卓雄，榊　裕之 監訳

超 LSI システム入門

培風館，B変形判，xiv＋416 pp.，1981年，7,800円

VLSI は，多くの複雑なサブシステムから成り立っており，いまや超大規模集積回路というよりも超大規模集積システム（VLSI システム）と呼ぶほうがふさわしい様相を呈してきた．さらに，予測される高集積化の進展とその広範な利用を考えると，電子回路の小型化として始まった IC や LSI の延長上で，VLSI を考えることが許されなくなった．

今日，VLSI エレクトロニクスは，製造技術の発展にたずさわっている人々だけでなく，コンピュータ・サイエンスやコンピュータ・アーキテクチャの研究に従事している人々の問題にもなっている．すなわち，ディジタル・システムの設計法，ハードウェアとソフトウェアの機能分担，計算アルゴリズムの設計等に影響を与え，コンピュータ・サイエンスの主要な活動分野となりつつある．

ところが，これまでの集積回路の設計は半導体企業の回路・論理設計者の領域で，コンピュータ・アーキテクチャにたずさわる人々は，その回路の仕様決定や設計に滅多に参加していなかった．また，従来のこの分野のテキストは，極く狭い部分について詳細な説明を与えようとするものが多く，一般的な応用への適用に欠けていた．

本書は，コンピュータ・システムの設計をこころざす人を対象に，システム設計からパターン設計まで，VLSI 設計の基本事項と要点を述べたものである．しかも，単に入門書にとどまらず，各所で将来の問題を論じている．

構成は，以下に示すように4つの大きな部分に分かれている．

まず，最初の部分は，

第1章　MOS 素子とその回路

第2章　集積システムの製作法

で，素子，回路，製造技術の基礎が述べられている部分である．すなわち，第1章では，nMOS FET の基本的性質が，また複数の MOS FET から構成した典型的回路の説明・解析が，実用的見地から具体的に記されている．第2章では，パターン形成の順序や基本的な設計ルールが具体的に述べられている．この他，製造技術に関連したり，下地物質の性質に緊密に関与する事項の解説もある．

第3章　システム化した構造におけるデータと制御の流れ

第4章　集積システムの設計の実際：回路図から回路パターンのレイアウト，ウェーハ製造まで

において，システム設計とその具体化手法，実用的な規則を述べている．すなわち，第3章では，将来の VLSI の設計法と現実の基本回路の設計を示し，規則的な回路構造の必要性を説く．第4章では，マスク・レイアウト技術に触れ，マスク・パターン・データ作成用の CAD システムについて検討している．

3番目の部分は

第5章　LSI コンピュータ・システムの概要と OM 2 データ・バス・チップの設計

第6章　システム・コントローラのアーキテクチャと設計，OM 2 コントローラ・チップの設計

において，LSI システム設計の実際例（California 工科大学が開発した LSI コンピュータ・システム OM 2）を紹介している．第5章では，データ・バス・チップについて，個々の論理サブシステムをどのように LSI 回路に組み込めば，実際のコンピュータ・システムのアーキテクチャを満足することができるかを詳しく述べている．第6章においては，OM 2 システムのコントローラ・チップの働きについて述べるとともに，全システムの順序制御についても詳しく解説している．

第7章　システムのタイミング

第8章　高度並列処理システム

第9章　計算システムの物理

の3つの章では，現在および将来の興味ある研究課題，すなわち同期式システムの可能性，技術の物理的限界の問題等を考察する．すなわち，第7章では，論理設計にかかせないタイミングの問題を論じている．一般的な同期式システムの手法と限界，自己同期システムの考え方やその構成要素およびシステム設計が示されている．第8章では，VLSI と並列処理に関する4つの独立したテーマ（従来型コンピュータにおける通信と並列処理，プロセッサ・アレイ向きのアルゴリズム，階層構造コンピュータ，全域的通信を伴う高度並列処理構造）が書れてい

る．第9章では，有限な時間やエネルギーをどの程度まで小さくできるかという，極限の問題を取り扱っている．

なお，原著は C. A. Mead and L. Coway, "Introduction to VLSI Systems", (Addiso-Wesley, Massachusetts, 1980) 396 pp., $25.95 である．

Patrick Henry Winston, Berthold Klaus Paul Horn 著

"LISP"

Addison-Wesley, xii＋430 pp., 1981.

Winston 教授の

"Artificial Intelligence", Addison-Wesley, 1977

は，すぐれた人工知能の入門書として高く評価されている．同書は2部から成り，第1部では，人工知能の主要なテーマを述べ，その基本的な考え方を平明に説明している．第2部では，第1部の基本的な考え方を実際に LISP でプログラムする技法を述べている．

第1部はすでに京都大学の長尾教授と電子技術総合研究所の白井氏によって邦訳されている．

長尾真，白井良明共訳，"人工知能"，培風館，288 pp, 1980.

本書「LISP」は第2部の改訂増補版である．本書は，この成り立ちからわかるように，単に LISP プログラミングのための本ではなく，LISP によって人工知能の応用システムを製作するための本である．同様の本として，英書では

Eugene Charniak, C. Riesback, D. McDermott, "Artificial Intelligent Programming", Lawrence Erlbaum Associates, 1979.

があり，邦書では

佐伯胖監修，"LISP による認知心理学(全3巻)"，東京大学出版会，1981，1982，未刊.

がある．本書のタイトル「LISP」も，「Artificial Intelligence Programming in LISP」という意味である．

さて，本書は2部構成であり，第1部では基本的な LISP プログラミングの練習を行い，第2部で，問題解決，エキスパート・システム，自然言語理解といった応用人工知能システムの製作におけるプログラミング技法を述べている．章立てを示すと；

PART I

1 UNDERSTANDING SYMBOL MANIPULATION
2 BASIC LISP FUNCTIONS
3 DEFINITIONS, PREDICATES, CONDITIONALS, AND SCOPING
4 RECURSION AND ITERATION
5 PROPERTIES, A-LISTS, ARRAYS, AND ACCESS FUNCTIONS
6 USING LAMBDA DEFINITIONS
7 PRINTING, READING, AND ATOM MANIPULATION
8 DEFINING FEXPRS AND MACROS
9 LIST STORAGE, RECLAMATION, AND SURGERY
10 EXAMPLES INVOLVING BINARY IMAGES
11 EXAMPLES INVOLVING SEARCH
12 EXAMPLES FROM MATHEMATICS

PART II

13 THE BLOCKS WORLD
14 RULES FOR GOOD PROGRAMMING AND DEBUGGING
15 ANSWERING QUESTIONS ABOUT GOALS
16 GETTING FUNCTIONS FROM DATA
17 SYMBOLIC PATTERN MATCHING AND SIMPLE THEOREM PLOVING
18 EXPERT PROBLEM SOLVING USING IF-THEN RULES
19 INTERPRETING AUGMENTED TRANSITION NETWORKS
20 COMPILING AUGMENTED TRANSITION NETWORKS
21 PROGRAM WRITING PROGRAMS AND NATURAL LANGUAGE INTERFACES
22 IMPLEMENTING FRAMES
23 LISP IN LISP

なお，本書では一貫して MACLISP を使用しているが，本書を「Artificial Intelligence Programming in LISP」の演習書として利用する場合，これは支障とはならないだろう．また，すべての練習問題には解答が付けられ，自習用にも適している．

最近，にわかに人工知能に対する関心が高まり，symbol cruncher にとっては同慶の至りである．

応用人工知能システムの symbol cruncher にとっては，「Artificial Intelligence Programming in PROLOG」である；

Robert Kowalski, "Logic for Problem Solving", North-Holland 1979.

と，本書は "must" であろう．

本書を紹介する所以である．

●SIGGRAPH 81 開催——Dallas, 1981年8月3日～7日

18のコース，15のテクニカル・セッション，フィルム／ビデオ・テープによるショー，機器展示等が行われた．コースは，入門レベルのチュートリアルと特定テーマによる上級のセミナーから成り，チュートリアルでは，ラスター・グラフィックス，2次元コンピュータ・アニメーション，ユーザ・インタフェースの心理学および設計等が取りあげられた．一方，セミナーでは，低価格グラフィックス，VLSIのCADシステム，画像合成，3次元コンピュータ・アニメーション，管理者用ビジネス・グラフィックス，立体モデリング，彫塑曲面等が講義された．また，今回のパネル・セッションは，高性能グラフィック・プロセッサ用のVLSIチップ，インタラクティブ・システムのユーザ・インタフェース，ANSI/X 3H 3におけるグラフィックスの標準化をテーマとして行われた．このほか，社長討論会(Presidents' Forum)と題して，コンピュータ・グラフィックス業界を代表する Vector General Inc., Evans & Sutherland Computer Corp., Dicomed Corp., Versatec Corp. の社長によるパネル討議が行われた．

●IJCAI '81——Vancouver, 1981年8月24日～28日

同会議は隔年に International Joint Conferences on Artificial Intellingence によって開催されるもので，本年の特徴はチュートリアル講演にあり，「コンピュータによる視覚」(H. G. Barrow, J. M. Tenenbaum, Fairchild Artificial Intellingence Research Laboratory),「エキスパート・システム：知識の活用」(F. Hayes-Roth, Rand Corp.),「人工知能と認知科学の先進的計算環境」(B. Sheil, S. Brown, Xerox Palo Alto Research Center),「ロボット工学——原理と応用」(M. Raibert, Carnegie-Mellon Univ. Robotics Institute) 等が講義された．

●情報処理学会夏季シンポジウム——伊東市，1981年7月16日～17日

これは，「情報処理学会プログラミング・シンポジウム委員会が主催し，「ソフトウェア業と技術移転」のテーマで開催された．また，同名テーマでパネル討議が開かれた．主な報告は，UNIX の使用経験(SRA)，LISP による3次元グラフィックス（マイクロ・コミュニケーションズ)，知的ビデオディスク(JBA)，PASCAL 教育（武市コンサルティング・オフィス)，マイコン・ソフトウェアの流通（システムズ・フォーミュレート)，Ada とモジュラリティ（立教大)，科学技術計算の話題（日大）等であった．

●7 th International Conference on Very Large Data Bases——France, 1981年9月9日～11日

今回のパネル討議のテーマは，画像データベースのユーザ・インタフェース，データベース・マシン，分散データ処理と分散データベース処理，ソフトウェア開発用データベースであった．また，招待講演は，J. Gray (米国)，K. Sevcik (カナダ)，D. J. Dewitt (米国)，H. Gallaire (フランス)，D. Tsichritzis (カナダ)，D. Chamberlin (米国) の各氏によって行われた．

●COMPCOM Fall '81開催——Washington D. C., 1981年9月15日～17日

テーマは「生産性向上 (Productivity—An Urgent Priority)」であり，R. de Lauer (国防省) および小林宏治 (日電) の両氏を基調講演者に迎えて開かれた．また，B. Boehm (TRW/DSSG)，J. Albus (NBS)，C. V. Ramamoorthy (Univ. of California) の各氏によって招待講演が行われた．なお，パネル討議は，ソフトウェア生産性尺度，人間工学，ソフトウェア管理の生産性，オフィス・オートメーション，米空軍の ICAM プロジェクト，データベース標準化，ソフトウェア・プロダクトの品質等であり，一般講演の主なテーマは，ソフトウェア開発の人間工学，ソフトウェア信頼性に対する資源割当，統合ソフトウェア標準 STEPS 等であった．

●昭和56年電気四学会連合大会開催——東京，1981年10月1日～3日

特別講演は，「新エネルギー技術開発の現状と将来性」（新エネルギー総合開発機構・綿森力氏）と「物質と生命——組換え DNA 技術の発展」（慶大・渡辺格氏)．パネル討議のテーマは，電気系工学の教育，超高速集積化電子デバイス，広域地震災害に対する電気技術の諸問題，最近の医用エレクトロニクス等であった．一般講演の主な話題は，地震観測および情報処理システムの現状，高密度磁気バブルメモリ，光ビーム記憶の現状と将来，紫外線リソグラフィ，液晶ディスプレイのカラー化，極限微細 NMOS デバイス，SOS LSI，化合物半導体デバイス，デバイス・シミュレーション，VLSI のテスタビリティ，ディジタル・テレビジョンの標準化等であった．

●第10回情報化週間——10月に情報処理関連の各種の行事・催物が全国的に行われた．ここでは，次の3つのシンポジウムを紹介する．

①データ・ショウ'81国際シンポジウム（電子工業振興協会主催，東京，1981年9月28日～29日）．M. D. Ercegovac 氏 (UCLA) の講演「米国におけるスーパー・コンピュータ開発の現状と展望」のほか，パネル討議が開かれた．また，「超高速機能デバイスの動向」（電総研・片岡照栄氏），「スーパー・コンピュータ開発の必要性」（日本気象協会・広瀬元孝氏），「わが国におけるスーパー・コンピュータ利用の現状と展望」，（三菱総研・川面恵司氏）等も報告された．

②情報化週間シンポジウム「情報革命——80年代の課題」（日本情報処理開発協会主催，東京，1981年10月5日～6日）．基調講演は，小松左京氏による「国民コンピュータ時代」，招待講演は A. Osborn 氏 (Osborn Computer) による「マイコン革命——マイクロエレクトロニクスの社会・産業へのインパクト」と N. P. Negroponte 氏 (MIT) による「オフィス革命——ニューメディアの可能性と創造への挑戦」であり，このほかパネル討議も行われた．

③第5世代コンピュータ (FGCS, Fifth Generation Computer Systems) 国際会議（日本情報処理開発協会主催，東京，1981年10月19日～22日）．この会議は知識情報処理システム (KIPS, Knowledge Information Processing System) を最大の特徴とする第5世代コンピュータの要件等を討議するもので，知識工学の創始者 E. A. Feigenbaum 氏 (Stanford Univ.)を含む6名の招待講演者と日・米・英・仏・西独の技術者約300名が参加した．

●情報処理学会第23回（昭和56年後期）全国大会開催——東京，1981年10月14日～16日

早川武夫氏（専修大）の特別講演「法とコンピュータ」と元岡達氏（東大）の招待講演「第5世代コンピュータの構想」が行われたほか，「Ada とどうつき合うか」（司会 東大・和田英一氏），「コンピュータ・トラブルをめぐって」（司会 東京理科大・菅野文友氏）をテーマとするとするパネル討議が開かれた．

今回の分野と主なテーマは次のとおりであった．

①コンピュータ・アーキテクチャ：ダイナミック・マイクロプログラム制御計算機QA-2のシステム管理，ファームウェア開発支援システム CHEF，計算機—FAX接続方式，COBOL マシンの評価，データフロー・マシン TOPSTAR-II による PROLOG 並列処理システムの試作，LISP マシン ELIS 等．

②データベース：B-tree エンジン，可変構造多重処理データベース・マシン，ソートをベースにしたデータベース・マシン，関係形式データベース問合せ言語としての多層論理，CODASYL データベースに対するリレーショナル・インタフェース，リレーショナル・データベース管理システム RDB/V 1等．

③ソフトウェア工学：プログラマーズ・ワーク・ベンチ・システム PWB 1，ソフトウェアの再利用，ソフトウェア修理情報管理システム，形式的仕様に基づくプログラム変換，ソース・プログラムの世代管理，図形によるプログラム入力システム等．

④ネットワーク：分散システムの構成設計支援ツール，通信プロトコル記述用言語 AFDET，ネットワーク・ジョブ管理等．

⑤人工知能および自然言語処理：PROLOG による問題解決，脳の機能と演繹的認識法，知識ベース・システムのプログラム環境，格構造に基づいた意味モデルによる日本語の解析，日英機械翻訳システム ATLAS/U，計算機処理のための辞書作成，意味論的観点からの言語理解システム等．

⑥その他：文字認識，音声認識，数値計算，マイクロコンピュータ，論理設計シミュレーション，画像処理，数式処理，形状処理，オフィス・システム，日本語端末技術，エディタ等．

●ACM '81開催——Los Angels，1981年11月9日～11日

基調講演は，SF「華氏45度の世界」の著者として有名な小説家 R. Bradbury 氏によって，「1984年のかなた——不可視の革命 (Beyond 1984: The Invisible Revolution) と題して行われた．また，ソフトウェア技術の経済学 (Software Engineering Economics)，グラフィックスの動向，コンピュータと法律，コンピュータの機密保護・監査・統制，ソフトウェア品質保証，ソフトウェアの構成管理 (Software Configuration Management) をテーマとするチュートリアル・セミナーが開かれた．チューリング賞は E. F. Codd 氏 (IBM) で，1970年の論文 "A Relational Model of Data for Large Shared Data Banks" に始まる一連の関係型データベース管理システムに関する研究業績に対して与えられた．ホッパー賞の Bricklin 氏 (Software Arts, Inc.) はパーソナル・コンピュータ用の簡易型言語 VisiCalc (Visual Calculator) の開発による貢献が評価されたもの．今回のパネル討議のテーマは，医療情報システム・ネットワーク，コンピュータ・チェス，Ada の技術的課題，システム資料作成の新技術，自動情報検索，80年代のソフトウェア製品産業，CAI におけるビデオ・ディスクの等であった．

MEMORANDUM

〈UNIVAC 関係の論文講演等の要約〉

●構造的流れ図を作成するプログラム——ここで述べる「構造的流れ図作成プログラム」Contour は，ループや条件文の範囲を破線で囲むことによって，明確にするものであり，複合文中に含まれる複合文は，地形図における等高線に類似した表現となる．Contour で採用した等高線方式のプログラム構造表現は，画面表示に要する水平方向の長さをあまり必要としないことや，While 文や Case 文の範囲が明確に記述できる等の利点を持っている．なお，このプログラムの対象言語は，PASCAL とプログラミング言語Cである．(J. F. Gimpel, "A Method of Preparing Structured Flowcharts", *ACM SIGPLAN Notices*, Vol. 15, No. 10, Oct., 1980)

●小企業における大規模計算の導入の影響——ジョブ制御言語への依存度の軽減，手続型言語から仕様型言語への移行，ソフトウェア教育やドキュメンテーションに対するコンピュータの支援等が進められている．(R. M. Firestone, "Large Scale Computing for the Small Enterprise." Conf. on Computers in Small Business, 1981)

●ハードウェア技術の動向——将来のVLSI 素子としては MOS が多用されようが，大型コンピュータについては当分の間バイポーラ素子が用いられよう．CMOS/SOS は高速性に優れているが，価格の点で問題がある．ジョセフソン素子は，電力消費がゼロに近く極めて高速であるが，その実用は 1980 年代の末となる．また，VLSI チップの製法では，最近電子ビームによる超薄金属マスクの作成と軟X線による露光が有望な技術として登場してきた．そしてVLSI の設計では，大量生産チップ以外は，すべてセミ・カスタム方式によって作られることになろう．(H. L. Apfelbaum, "Technology Trends in Hardware in INFOTECH State of the Art Report, Series 9 No. 1, Fifth Generation")

●有限回の先読みによる正則文法のパージング手法——1つの文法が与えられたときに，それが固定的先読みでパース可能かどうかを判定し，かつ，可能な場合にはその最小先読み回数を求めるアルゴリズムを提示する．そして，計算済みの大きなパージング・テーブルを用い，リアルタイムにパースする方法，約 $3n$ 個の記憶域を用い入力ストリングの長さと比例する時間をかけてパースする方法，の双方を評価する．(T. J. Ostrand, "Parsing Regular Grammars with Finite Lookahead", *Acta Informatica*, Vol. 16 Fasc. 2, 1981)

●自己検証のための実現技術——VLSI におけるテスタビリティ問題を解決する方法として，自己検証 (self—verification) の概念を導入し，つぎに自己検証を実現する技術として，ロジックの分割，障害検出回路および内部刺激発生器 (Internal Stimuli Generator) の配置や設計等を論じる．さらに，自己検証の実現例の詳細を示すとともに，その実現コストに関する定量的考察を行う．(R. M. Sedmak, "Implementation Techniques for Self-verification", Test Conference, 1980)

●抽象データ型の仕様記述と実現のためのカプセル化機構——"モジュール" は，抽象データ型の記述のために，構成的仕様定義（論理的構造の仕様記述と，操作の意味の仕様記述の2つ）を用いている．シンボル・テーブルを "モジュール" の具体的応用例として，その正当性を証明する．(B. G. Claybrook, et al., "Module: An Encapsulation Mechanism for Specifying and Implementing Abstract Data Types", ACM '80)

●分散型データベース・ネットワークの総合設計システム——これは，ファイルの配置，並行性制御，問合せ処理，信頼性，資源感度分析 (Resource Sensitivity Analysis) の問題を解決するために開発されたシステムであり，分散型データベース・ネットワークのモデルとして待ち行列網を採用している．また，分散型データベース・ネットワークをホスト・コンピュータを中心とするホスト・サブネットワークとコミュニケーション・サブネットワークの2つに分けて取り扱っており，入力モジュール，インタフェース・モジュール，モデリングおよび出力モジュール，分析モジュールによって構成される．(G. M. Schneider, et al., "An Integrated Design System for Distributed Database Networks", COMPCON '80 Fall)

●分散型データベース (DDB) の新しい更新法——CMT (Cooperative Multi-Thread) と呼ぶ方法で，複数の DBMS が時分割によって協力しつつ DDB を制御する方法を採っている．特徴は，マルチ・スレッド・モードによる高いスループット，柔軟性，パフォーマンスの予測が可能．(R. J. Greene, "An Alternative Approach to Distributed Data Base

Updating", NCC '81)

●レーダの走査パターンのリアルタイム・シミュレータ——レーダの走査パターンを作成しフロッピー・ディスクに出力するパターン生成システム (PDS, Pattern Development System) と，フロッピー・ディスク上の特定のパターンを DA 変換し各レーダーのシミュレータに送るシミュレーテッド・パターン生成システム (SPGS, Simulated Pattern Generation System) とについて報告している．(J. S. Sidhu, "Microprocessor-Based Radar Pattern Development and Simulation Systems", SCSC '80)

●ネットワーク・アーキテクチャの現状——最近，ローカル・ネットワークの構築用ハードウェアやローカル・ネットワークのプロトコルの開発が急速に進められている．また，複数のローカル・ネットワークを最下層の要素として包含する階層的ネットワークの構築が今後の趨勢と見られており，複数ネットワーク間の接続（インターネッティング）がその鍵とされ，パス結合と回線交換の技術の重要性が増大している．(K. J. Thurber, "An Assessment of the Status of Network Architectures", COMPCON Fall '80)

●オペレーティング・システムのためのセキュリティ——ユーザ・プロファイル，実行プロファイル（プロジェクト用のプロファイルに相当）およびプログラム・プロファイルのそれぞれに蓄積されたアクセス・リストによってセキュリティを制御するところからプロファイル準拠セキュリティ方策 (Profileoriented Security Policy) と呼ばれる．対象のアクセス・リストではなく，主体 (Subject, ユーザやプログラム等) のアクセス・リストを用いている．(C. R. Young, "A Security Policy for a Profile Oriented Operating System, NCC '81)

●分散型コンピュータ・システムの解析的モデル——このモデルは分散型コンピュータ・システムの性能の予測を行うもので，システムの構造や負荷 (workload) の変更に関しシステム設計者が感度分析を行う際の高水準ツールとして作用できる．解析的モデルの取り扱いに不慣れなユーザでも使用できるようにユーザ・インタフェースの使いやすさを配慮しているほか，待ち行列網解析 (Queueing Network Analysis) の最近の成果も取り入れている．(R. C. Heinselman, "An Analytic Model for Distributed Computer Systems", Annual Conference on Modeling and Simulation, 1981)

☆

技　　報
UNIVAC TECHNOLOGY REVIEW
No. 2

発　行　日	昭和 57 年 2 月 28 日
発行人兼編集人	富　田　和　夫
発　行　所	日本ユニバック株式会社 東京都港区赤坂 2-17-51 〒107 TEL (03) 585-4111 (代表)
頒 布 価 格	1,500 円
印　刷　所	三美印刷株式会社