ISSN 0289-6257

技報

UNIVAC TECHNOLOGY REVIEW

1986年8月　第11号

日本ユニバック

原子炉の工学的安全施設の性能評価のためには，事故時の冷却材の沸騰や凝縮の挙動，すなわち2相流挙動の詳細な解析が不可欠である．従来は，これらの解析に気相と液相が平衡状態にあると仮定して解析する手法（HEM モデル）を用いていたが，この手法では気相と液相が分離している状態や非平衡状態の正確な解析ができない．近年，気相と液相とを独立に扱う2流体モデルが考案され，この手法を用いた2相流解析が主流になりつつある．二ノ方らの「**2相流解析コードの開発**」は，高速炉のナトリウム沸騰解析用に開発した2流体モデルによる2相流解析コード SABENA の1次元版について，その計算モデルや実験解析の結果を述べたものである．

複雑な形状を取り扱う型彫りシステムでは，型彫りモデルを複数の曲面で表現する．曲面上の切削領域をトリム，トリムを滑らかに接続して創成される曲面をフィレット曲面と呼ぶ．従来のフィレット曲面創成法は，2曲面を対象とするものであった．橋本の「**複数の曲面を接続するフィレット曲面創成法**」は，多数のトリムを同時に接続する方法について述べている．複数のトリムを取り扱う上では安定性が最も問題となる．このため従来の方法の中で，とくに安定性の高い断面形状接続法をトリムに適用し，解としての可能性のあるものをすべて求め，それを選別する手法を採用している．

Ada* は，米国国防総省の要請によって開発された組込型計算機システムの設計・開発・保守のための言語である．G. Snyder らの「**UNIVAC シリーズ 1100 Ada コンパイラ・システム**」は，UNIVAC シリーズ 1100 用の Ada コンパイラ・システムの概要，使用者インタフェース，内部構造，コンパイラの開発方法などについて述べている．なお，本コンパイラと次に述べるプログラミング支援環境 Byron** 1100 は，Sperry 社の防衛機器部門の依頼によって，Intermetrics 社が開発中のものである．Ada のプログラミング支援環境としては，国防総省の定めた Stoneman と呼ばれる APSE (Ada Programming Support Environment) の要求仕様が標準とされている．

* Ada は米国政府(Ada Joint Program Office)の登録商標である．
** Byron は Intermetrics 社の登録商標である．

M. Gordon らの「**Byron 1100 プログラミング支援環境**」は，先の APSE に基づき開発された Byron 1100 の機能および使用法を紹介している．Byron 1100 は，プログラム開発言語 PDL (Program Development Language) と統合プログラミング支援ツールによって構成される．Byron 1100 は最初の商用の Ada プログラミング支援環境の一つである．

近年，伝統的なシステム開発のライフサイクルの概念は，有害ではないかという議論がなされ，従来と異なる開発方法が注目されている．それらの方法の一つとしてラピッド・プロトタイピングがある．加藤の「**JSD 仕様の実行系の試作**」は，JSD (Jackson System Development) の仕様を実行する道具の開発に関する実験報告である．本報告では，そのために導入した言語 J を紹介するとともに実行方法と実行結果を述べている．

データ・フローに基づくソフトウェアの記述は，従来のプログラミングのそれと異なっているため，ソフトウェアの設計の初期の段階を除いては一般的に使用されていない．データ・フローをストリームを用いて表現すると，実行可能なプログラムをこの方法で記述できる．大田の「**ストリームを用いた論理型言語インタプリタ**」は，ストリームの Lisp による実現，データ・フローに基づく木探索の問題解法を説明し，これらの準備を経た後に論理型プログラミング言語のインタプリタの実現について述べている．

ハードウェア・トラブルの要因となる潜在的不良を検出する技法については，従来から各種のマージナル・テスト（電圧・温度・クロックなど）が行われてきた．小塩らの「**バックパネルの潜在的不良検出手法**」は，従来と異なる新しい方法であり，静電発生器を用いた高電圧での放電現象によってバックパネル内の異常な近接箇所を検出する技法について述べている．この方法の採用によって難解な間欠障害を解析するとともに潜在的な不良箇所を検出することができたと報告している．

報告 複数の曲面を接続するフィレット曲面創成法

A Method for Multi-Fillet Surface Construction

橋 本 可 輝

要 約 NC 型彫りシステムが複合曲面を取り扱うようになり，フィレット曲面創成法も，従来の2曲面だけでなく，複数のトリムを対象とすることが要求されている．

複数のトリムを取り扱う上で安定性が最も問題になる．そこで本稿では，従来の方法の中でとくに安定性の高い断面形状接続法をトリムに適用し，さらに解としての可能性のあるものをすべて求め，それを選別する手法を採用する．

この方法により，複雑な形状に対しても安定してフィレット曲面を創成するプログラムが実現できた．

Abstract The increasing demand for complex free surfaces in stamping die design systems has changed the methods to construct fillet surfaces from handling two surfaces to multiple ones. Since the computational stability is most important in constructing a fillet surface, we adopt the section-curve method in order to obtain all possible solutions, from which we select the best solution.

As the results, the program is able to construct the fillet surfaces stably even for complex shapes.

1. はじめに

複雑な形状を取り扱う NC 型彫りシステムでは，型彫りモデルを複数の曲面で表現する．一般に曲面は実際の物より大きく，切削領域外の部分を持つ．A社において新規開発した NC 型彫りシステムでは，曲面ごとに切削領域を示す閉境界を持ち（以下，この閉境界によって領域が規定される曲面をトリムと呼ぶ），型彫りモデルをこれらの集合として取り扱う．

これらのトリム間を滑らかに接続するフィレット曲面は，デザイン，安全対策，また製品加工上要求され，型彫りモデルには必要な形状である（図1）．

筆者の知る従来のフィレット曲面創成法は，2曲面だけを対象としている．このため，曲面の切削領域外の部分にまでフィレット曲面ができ，フィレット曲面をそのまま型彫りモデルの一部として使うことができない．また，複雑な形状の曲面に対して，期待した形状のフィレット曲面が得られないことが多い．

新規開発にあたり，曲面でなくトリムを対象とし，さらに型彫りモデル作成の工数を削減するため，複数のトリムを同時に接続することが要求された．また，複雑な形状を取り扱うため，高い安定性と精度が要求されたが，それを実現したので以下に述べる．

なお，この手法を，複数のトリムを同時に接続する特徴から複合フィレット曲面創成法と呼んで従来の方法と区別し，創成されるフィレット曲面を複合フィレット曲面と呼ぶ．

2. 従来のフィレット曲面創成法

筆者の知るフィレット曲面創成法は，球接点接続法と断面形状接続法の二つに大別される．複合フィレット曲面創成法は，断面形状接続法が取り扱う2曲面を複数のトリムに拡

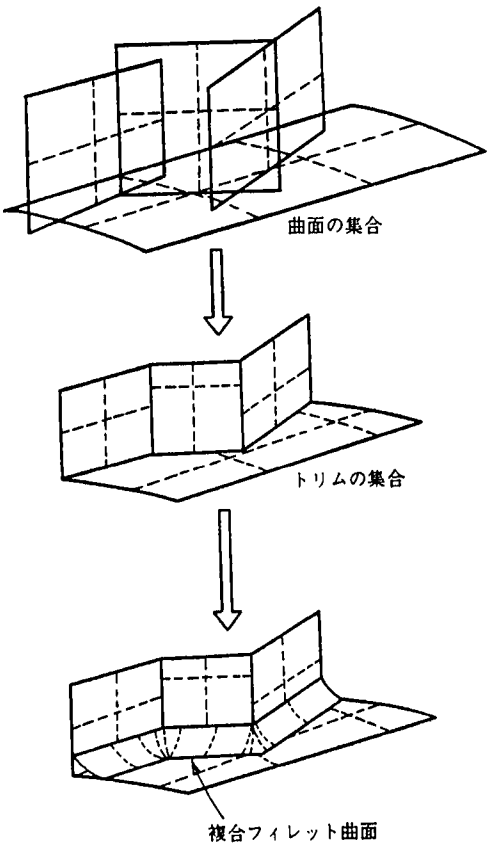


図 1　トリムと複合フィレット曲面の例
Fig. 1　Trim surface and multi-fillet surface

張して適用したものである．以下ではそれぞれの方法の概略と特徴を述べ，断面形状接続法を採用した理由を説明する．

2.1 球接点接続法

指定された半径を持つ球を2曲面にあてたときにできる接点を，順次接続することにより，フィレット曲面を創成する方法である．この方法は日本ユニバック(株)の図形処理パッケージ GEOPACK[1] で採用されている．

処理の流れは次のとおり．

1) 球の中心点を求める手掛りとなる近傍点を与える．
2) 近傍点に最近な曲面上の点を求める（図2(a)）．
3) 曲面上の最近点から新たな近傍点を作る．
   近傍点を作る方法として，①曲面上の最近点を指定，②半径分オフセットし，③オフセット点の中点を近傍点とする，方法などがある（図2(b)）．
4) 直前の近傍点と新たな近傍点が十分接近するまで 2)，3) を繰り返す．十分接近したときの近傍点が球の中心点，曲面上の最近点が球と曲面との接点になる（図2(c)）．
5) 近傍点を2曲面沿いに動かし，順次接点を求める．
6) 接点を接続し，フィレット曲面を作る．

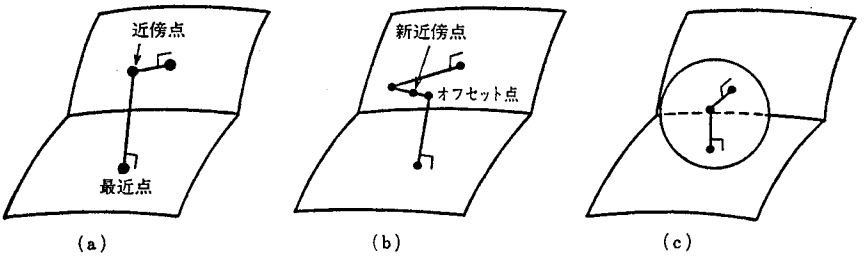


図 2　球接点接続法
Fig. 2　Contact-sphere method

2.2　断面形状接続法

2 曲面間に指定半径を持つ円板ゲージをあてたときにできるゲージの円弧をフィレット曲面の断面形状（以下，フィレット円弧と呼ぶ）と考え，これを順次接続することによって，フィレット曲面を創成する方法である．この方法は，津田ら[2]によって報告されている．

以下に処理の流れを示す．

1)　円板ゲージに相当する平面として，2 曲面の相貫線に垂直な平面を作る．
2)　曲面を平面で切断し，断面曲線を求める（図 3 (a)）．
3)　断面曲線を平面内で指定半径分オフセットする．オフセット曲線の交点が，フィレット円弧の中心点となる．交点に対応する断面曲線上の最近点がフィレット円弧の端点となる（図 3 (b)）．
4)　平面を相貫線沿いに動かし，順次フィレット円弧を求める．
5)　フィレット円弧を接続し，フィレット曲面を作る．

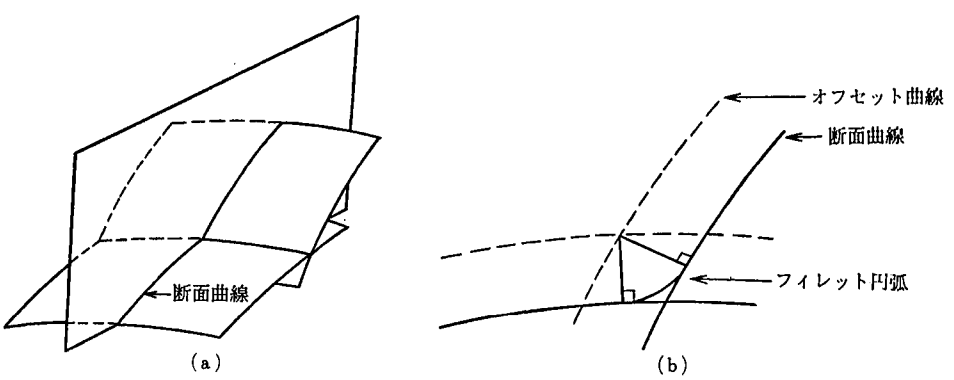


図 3　断面形状接続法
Fig. 3　Section-curve method

2.3　従来法の特徴

球接点接続法と断面形状接続法の特徴を，実現のための前提，処理速度，精度，および安定性の観点から検討する．

1)　断面形状接続法を実現するためには，断面計算，曲線のオフセット計算，および曲線と曲線との交点計算をするプログラムが必要である．これに対して球接点接続法では，曲面上の最近点を求めるプログラムがあれば十分である．このように球接点接続

法のほうが容易に実現できる.

2) 1)で述べた処理内容の相違により，球接点接続法の方が高速であることが期待される．実際，各方法の核となるプログラムの処理速度を GEOPACK のプログラムで測定すると，曲面上の最近点計算では1点あたり2ミリ秒程度，断面計算では1曲線あたり200ミリ秒程度の CPU 時間を要した．測定には UNIVAC 1100/92 を使用した.

3) 球接点接続法は，収束計算の回数を増やすことによって，精度を容易に向上させることができる．これに対し断面形状接続法は，断面計算，オフセット計算，および交点計算の精度がそのままフィレット曲面の精度に反映される.

4) 球接点接続法は，収束法である．収束法は解を一つと仮定し，その解に対して収束計算を行う．このため解が複数個存在する場合，不適当な位置に近傍点を与えると期待に反する解に収束することがある．また，曲面の形状が複雑になると収束しないこともある．これに対し，断面形状接続法では確実に解が求められる．しかも複数の解を得ることができ，解を適切に選別さえすれば安定してフィレット曲面が得られる.

2.4 断面形状接続法を採用した理由

以下の理由で断面形状接続法を選んだ.

1) 断面形状接続法のほうが複雑な形状に対して高い安定性を持つ……複雑な形状のトリム間には，解が複数個存在する．このような形状に対して，球接点接続法では適切な解が得られるという保障はない．つまり，球接点接続法は，解が一意に存在する場合にのみ有効な方法なのである．これに対して，断面形状接続法は3次元の複雑な問題を断面平面内という2次元の問題に置き換えることによって，複数のトリムの取り扱いが容易で，しかも解が必ず求まる．断面形状接続法を採用することで，複合フィレット曲面創成法が目標としている高い安定性が期待できる.

2) 断面形状接続法のほうがフィレット半径の保障されない箇所を滑らかな形状にすることが容易である……トリムとトリムが急角度に接続している箇所や，トリム内で急激に形状が変化している箇所では，フィレット半径が保障されないことがある．開発当初，このような箇所を何らかの方法で滑らかな形状にすることが要求された．この箇所を球接点接続法で処理すると穴が生じ，削り残しや，削り込みの原因となる（図4(a)）．断面形状接続法で処理するとフィレット円弧が交差するため，フィレット円弧の交差を検査すれば容易にこの箇所が発見できる．また，切断面を作りかえ，新たなフィレット円弧を作ることにより，半径は保障されないが，穴のない滑らかな形状にすることができる(図4(b))．詳細は3.3節参照.

3) 高速かつ高精度な断面計算プログラムが使用できる……同時に開発された断面計算プログラム[3]は，数10ミリ秒（CPU 時間）で1断面曲線を求め，しかも精度は1/10000 mm 以下である．これを使用することで，高精度なフィレット曲面を高速に作成することが期待できる.

3. 複合フィレット曲面創成法

複合フィレット曲面を創成するためには，①案内曲線，②案内曲線の左右のトリムの集合，③フィレット半径，および④フィレット円弧を選択するためのベクトルが必要である(図5(a)).

案内曲線は，断面形状接続法における2曲面の相貫線に相当する．この案内曲線沿いに

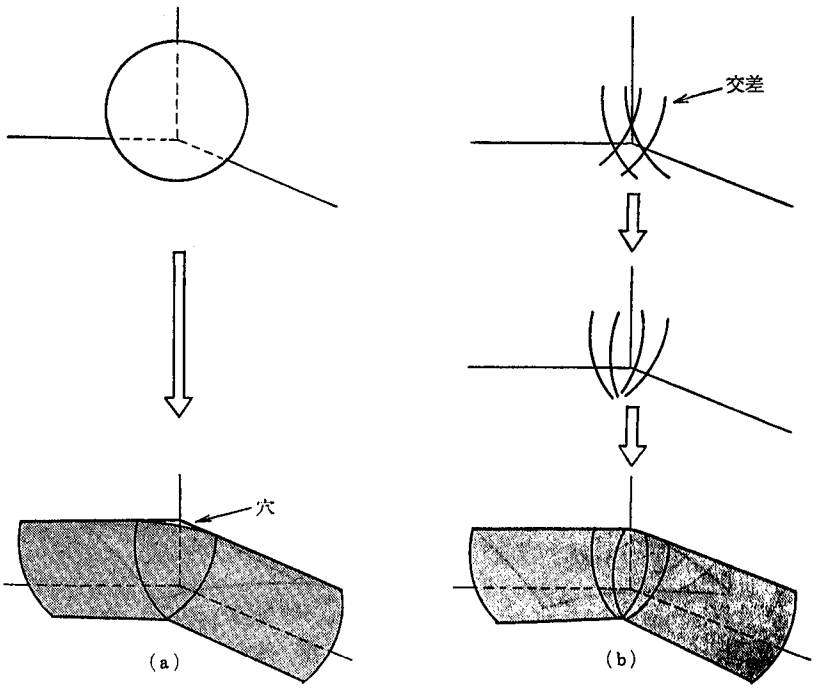


図 4　フィレット半径が保障されない箇所

Fig. 4　A place where the fillet radius can not keep constant

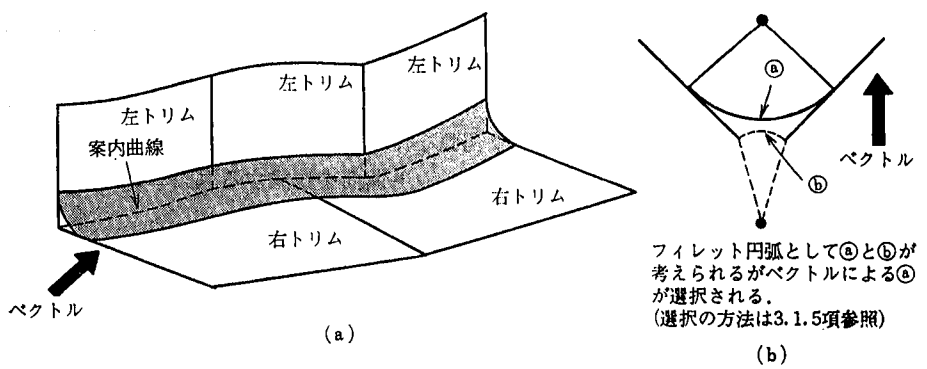


図 5　入 力 情 報

Fig. 5　Input data

切断面が作られ，順次フィレット円弧が作成される．

トリムを左右の集合に分けるのは，従来法の 2 曲面，すなわち案内曲線の左右の曲面をトリムの集合に置き換えたことによる．

フィレット半径として，一定または可変の値が指定できる．

フィレット円弧を選択するためのベクトルは，フィレット円弧の候補となる円弧が複数存在するときに，適切な円弧を選ぶ手掛りとして使用される（図 5 (b)）．

処理の流れを図 6 に示す．

1)　案内曲線の構成点において案内曲線に垂直な平面を作り，その平面内でフィレット円弧を決定する（図 6 (a)）．

2)　1)で作られるフィレット円弧のみでフィレット曲面を作成すると，フィレット曲面とトリムとの間に穴や，食い込みができることがある．このため，各フィレット円弧

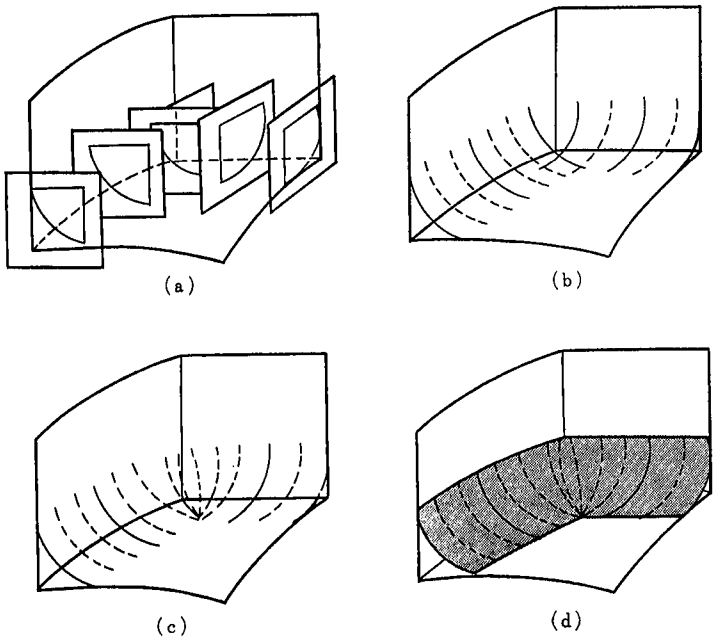


図 6 処理の流れ

Fig. 6 Process flow

間でトリムの形状を吟味し，必要な数だけフィレット円弧を補う（図 6 (b)）.

3) トリムとトリムが急角度に接続している箇所や，トリム内で形状が急激に変化している箇所では，フィレット半径が保障されないことがある．このような箇所ではフィレット円弧が交差し，この状態でフィレット曲面を作ると，曲面内で自己干渉する．このため，この箇所のフィレット円弧を作り直す（図 6 (c)）.

4) フィレット円弧を接続して，フィレット曲面を作成する（図 6 (d)）.

各処理の詳細を以下に述べる.

3.1 フィレット円弧の決定

3.1.1 断面曲線計算

トリムを構成する曲面および曲線と切断面との交線，交点計算を行い，切断面とトリムとの断面曲線を求める（図 7 (a)）.

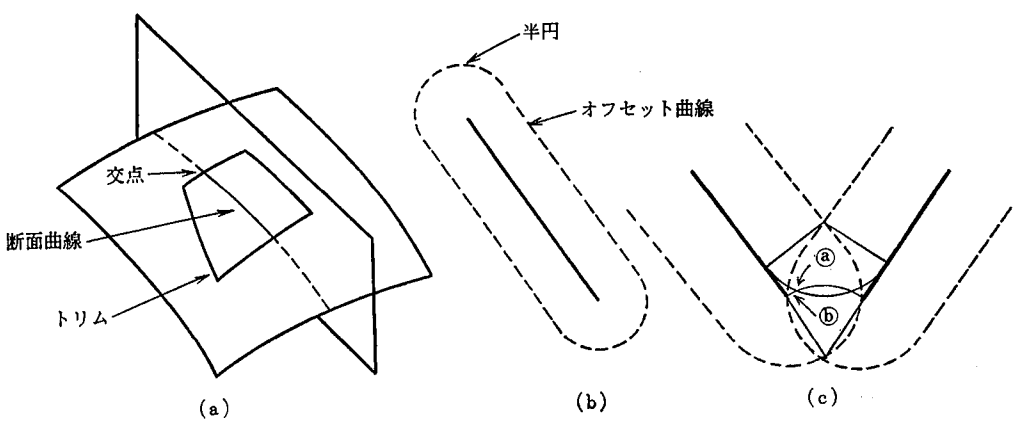


図 7 断面曲線とオフセット曲線

Fig. 7 Section curve and offset curve

3.1.2 オフセット計算

断面曲線を切断面内でフィレット半径分オフセットする．オフセットは両方向に行う．また，端点では半円を作り，オフセット曲線とつなぐ（図7(b)）．半円を作ることにより，図7(c)の円弧ⓑのようにトリムの端にかかるフィレット円弧も作成することができる．

3.1.3 交 点 計 算

左右のオフセット曲線間で交点計算を行い，交点をすべて求める．この交点が，それぞれのフィレットの円弧の中心点になる（図8(a)）．

3.1.4 各交点でのフィレット円弧の作成

交点を持つオフセット曲線の元の断面曲線上で，交点に最近な点を求める．この点が，フィレット円弧の端点となる．中心点と2端点が決まることによって，それぞれの交点でのフィレット円弧が決定する（図8(b)）．

3.1.5 フィレット円弧の選別

求めたすべてのフィレット円弧に対して以下の選別を行い，最適なフィレット円弧を決定する．

1) フィレット円弧の端点間の距離が非常に小さいと，この箇所でフィレット曲面が極端に狭くなる．このため，このようなフィレット円弧は採用しない（図8(b)では，2端点が一致し直線形状となる円弧ⓐ，ⓕ）

2) フィレット円弧の中心点との距離がフィレット半径未満となる断面曲線が存在する

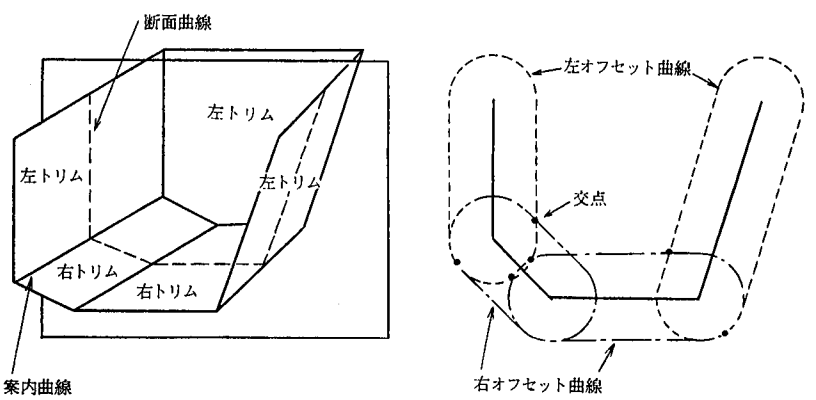
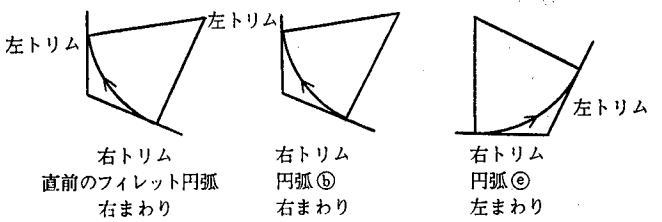


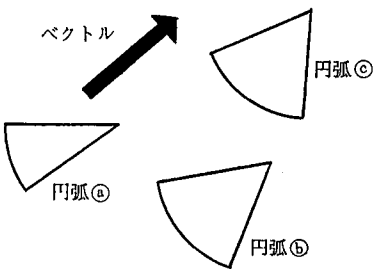
図 8 フィレット円弧の選別

Fig. 8 Fillet arcs selection



(紙面下の方向を案内曲線の進行方向とする.)

(a)



ベクトルにより円弧ⓒが選択される.

(b)

図 9 回転方向による選別とベクトルによる選別

Fig. 9 Fillet arcs selection specified by rotational directions and indication vectors

と，この箇所でフィレット曲面がトリムに食い込む．このため，このようなフィレット円弧は採用しない（図 8 (b)の円弧ⓒ，ⓓ)).

3) フィレット円弧の回転方向（案内曲線の進行方向に対して右まわりか，左まわりか）が直前の切断面でのフィレット円弧の回転方向と異なるとき，この区間でフィレット曲面にねじれが生じる．このため，このようなフィレット円弧は採用しない（図 8 (b) の円弧ⓔ，図 9 (a))．図 8 では，この段階で円弧ⓑが最適なフィレット円弧となる．

4) 1), 2), 3)の選別だけで，最適なフィレット円弧が決まらないとき，最終的な選別のためにベクトルを使用する．

ベクトルの方向において，最も遠い位置に中心点を持つフィレット円弧を，最適なフィレット円弧とする（図 9 (b))．

ベクトルは，最初の切断面では入力情報として与えられるベクトル，2 回目以後の切断面では直前のフィレット円弧の円弧中点から中心点へ向かうベクトルを使用する．

3.2 フィレット円弧間の吟味

トリムとトリムの境界では折れが生じており，フィレット曲線がトリムから離れる．これを防ぐため，トリムの境界の前後には必ずフィレット円弧を作る（図 10(a))．

また，トリム内で形状が急激に変化すると，フィレット曲面がトリムから離れる．トリム内の形状を得るために，図 10 (b) のように平面を作りトリムを切断する．得られた断面曲線の構成点の数が形状の複雑さを表す．左右のトリムで断面曲線を求め，構成点の数の多い方の数分，フィレット円弧を補う（図 10 (c))．

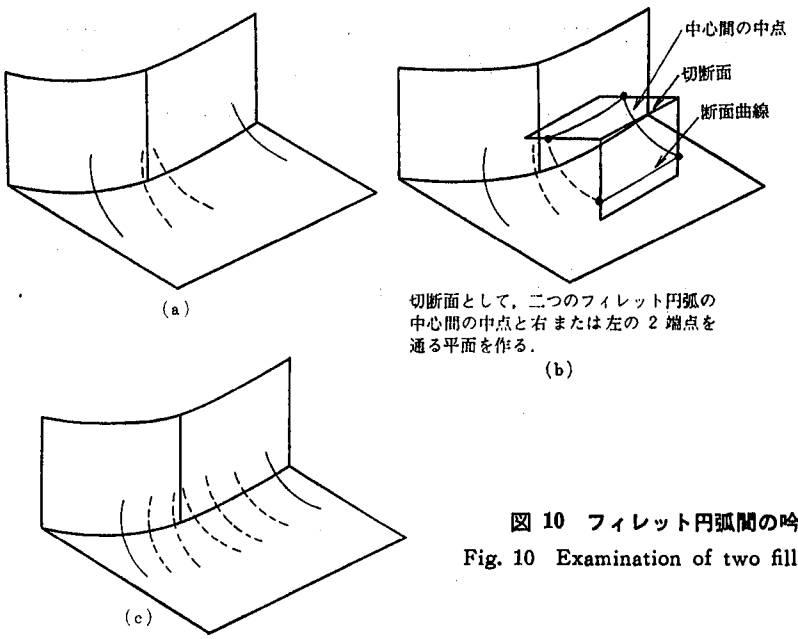


図 10 フィレット円弧間の吟味
Fig. 10 Examination of two fillet arcs

3.3 フィレット円弧交差箇所の除去

3.3.1 交差箇所の発見

二つのフィレット円弧を与えたとき，どちらかのフィレット円弧が相手の切断面と交われば，交差ありと判断する．この検査を該当フィレット円弧とその前後のフィレット円弧との間で行う．これにより交差箇所の範囲が決まる（図 11 (a)）．

3.3.2 補助曲線の作成

交差箇所の前後のフィレット円弧を何らかの方法で滑らかに接続しなければならない．この補助となる曲線を交差していないフィレット円弧の端点から作る（図 11 (b)）．

3.3.3 切断面の作成

交差箇所の前後の切断面の交線を求める．この交線を軸として，直前の切断面から直後の切断面まで等間隔に回転した平面を作る（図 11 (c)）．各平面間の吟味は，3.2 節と同じ処理を行う．

3.3.4 フィレット円弧の決定

補助曲線と切断面との交点を求める．この交点に最近なトリム上の点を求め，それをフィレット円弧の端点とする（図 11 (d)）．

3.4 フィレット曲面の作成

曲面を作成するには，各曲面構成点における座標値，U方向の接線ベクトル（U接ベクトル），V方向の接線ベクトル（V接ベクトル）が必要である．

ツイスト・ベクトル（UV 相互偏微係数）は零ベクトルとして扱う．

座標値は，フィレット円弧の端点を使用する．

U接ベクトルは，該当点とその前後の点で決まる円の接ベクトルを，トリムを構成する

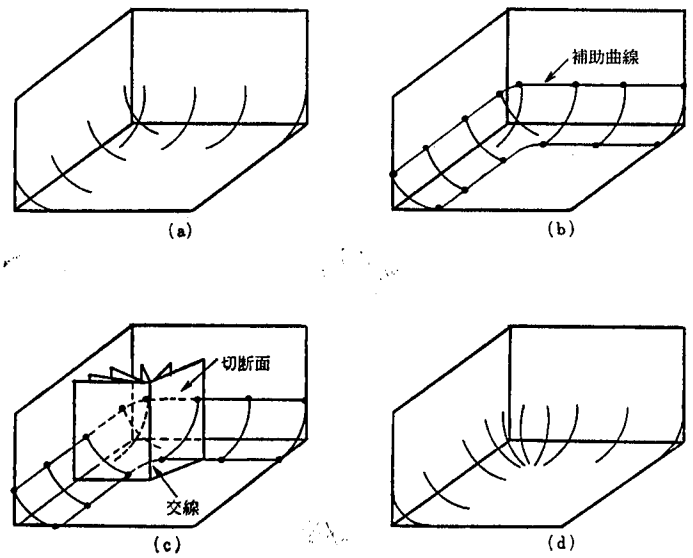


図 11　フィレット円弧交差箇所の除去

Fig. 11　Removal of crossing fillet arcs

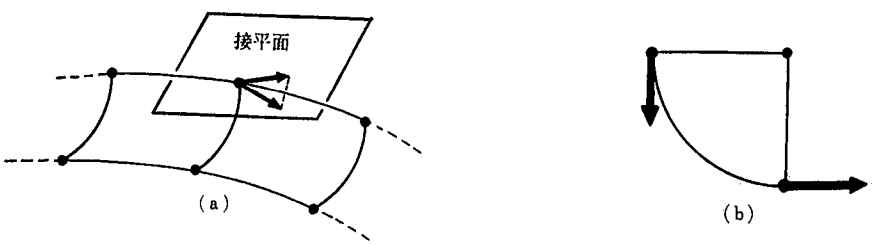


図 12　フィレット曲面の作成

Fig. 12　Construction of a fillet surface

曲面の接平面に投影したベクトルを使用する（図 12 (a)）.

U接ベクトルの大きさは，該当点，該当点におけるU接ベクトル，および直前または直後の点で決まる円上の弧長を使用する.

接平面に投影することで，フィレット曲面とトリムとの間の穴や食い込みの危険性が削減できた.

V接ベクトルは，フィレット円弧上の接ベクトルを使用する（図 12 (b)）.

V接ベクトルの大きさは，フィレット円弧の弧長を使用する.

4.　評　　価

複合フィレット曲面創成法を使用するプログラムについて精度と速度を評価する．また，この方法で未解決の問題について述べる.

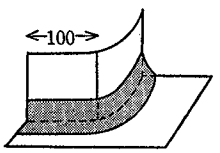
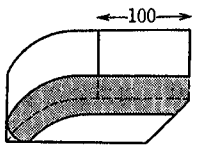
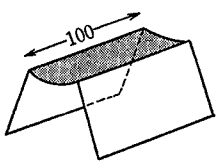
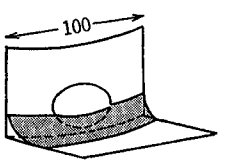
4.1　精度と速度

フィレット曲面構成点におけるトリムとの距離は，断面計算の精度がそのまま反映される．断面計算のトレランスとして 1/1,000 mm を与えており，このトレランス内の精度は確保されている.

構成点間でのトリムとの距離も，1/1,000 mm 程度の値が得られている.

表 1　複合フィレット曲面創成の例

Table 1　Examples of multi-fillet surface construction

テスト・モデル (2パッチ×2パッチ)	CPU 時間 (秒)	パッチ数 (U)	トリムとの 最大距離 (mm)	接平面との 最大角度
	1.13	21	0.00029	0.0028
	1.57	24	0.00078	0.0031
	0.79	6	0.00013	トリムの端にかかるため測定できない
	2.09	13	0.00085	0.0053

トリムを構成する曲面上の接平面と，フィレット曲面構成点における U 接ベクトルおよび V 接ベクトルとの角度は，1/100 度以下の値が得られている．

表 1 に，簡単なテスト・モデルに複合フィレット曲面を作成した例と，その精度，速度を示す．測定には UNIVAC 1100/92 システムを使用した．

4.2 問　題　点

複合フィレット曲面創成法で未解決な問題を以下に述べる．

非常に複雑な形状では不適当なフィレット円弧を選んでしまうことがある．

図 13 のようにトリムの幅が極端に狭い箇所ではフィレット円弧を求める必要はない．しかし，切断面内では円弧 ⓐ，ⓑ，ⓒ が得られる．円弧 ⓐ，ⓒ は直前のフィレット円弧

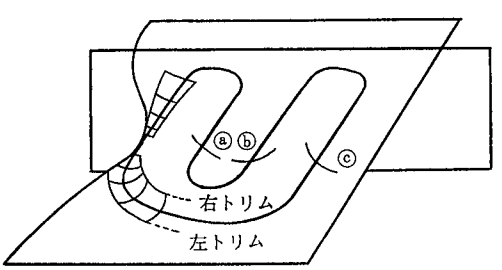


図 13　不適当なフィレット円弧を選ぶ例

Fig. 13　Example of wrong selection of a fillet arc

の回転方向とは逆に回転しているので除外されるが，円弧ⓑは残り，これが選択される．

この問題について，現在，直前のフィレット円弧との距離がフィレット半径の数倍以上（この値は使用者が与える）となるフィレット円弧は，選ばないようにして回避している．

また，非常に複雑な形状では，フィレット半径の保障されない箇所の形状が滑らかにならないことがある．

これは，複雑な形状では滑らかな補助曲線が得られず，これが直接フィレット曲面に影響することに起因している．

現在，これを回避するために，この問題が起こる箇所にはあらかじめフィレット曲面を作り，フィレット半径が保障されるようにしたり，なるべく小さな半径を指示したりしてフィレット曲面を作成している．

また，フィレット曲面が全体的に滑らかにならないのではないかという問題も考えられる．本方法では，案内曲線の形状がフィレット曲面の形状に直接影響する．左右のトリムの形状を考慮した案内曲線を与えることで，この問題を回避している．

5. おわりに

複合フィレット曲面創成法の特徴をまとめると，

1) 複数のトリムを同時に接続できる．
2) 複雑な形状に対しても高い安定性を持つ．
3) フィレット半径が保障されない箇所を滑らかな形状にすることができる．
4) 複雑な形状を対象にしているにもかかわらず高速であり，高精度である．

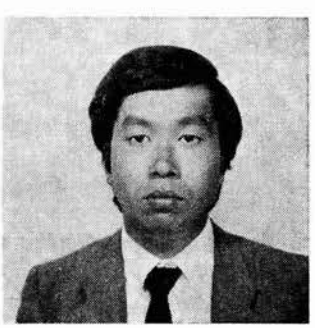
などの点をあげることができる．

開発当初，実用に耐えうる時間内でフィレット曲面が得られるかどうか危惧された．しかし，高速な断面計算プログラムを使用することによって，従来とほとんど変わらない高速性が得られた．また，可能性のある解はすべて求めそれを選別するという方法によって，高い安定性も実現できた．

評価で述べているように，複雑な形状を取り扱うゆえの問題もいくつかある．より高速で，より安定性の高いシステムを実現するために改良を推し進めるつもりである．

参考文献 [1] 図形演算パッケージ GEOPACK，日本ユニバック，資料コード：483205408，1984.
[2] 津田順司，上西博文，菊池純男，“3次元図形処理システム MDM-I の開発”，情報処理学会コンピュータ・グラフィックス研究会資料，Vol. 5, No. 3, 1982, pp. 1-9.
[3] 谷本茂樹，“点群近似による相貫線解法”，UNIVAC Technical Symposium '83 入選論文集，1984, pp. 451-459.

執筆者紹介 橋本可輝（Yoshiteru Hashimoto）
昭和33年生，56年筑波大学自然学類卒業，同年日本ユニバック(株)入社．CAD／CAM システムの開発に従事．現在に至る．



論文 2相流解析コードの開発

Development of Two-Phase Flow Analysis Code

二ノ方　寿, 岡野豊明

要　約　原子炉の工学的安全施設の性能評価のためには，事故時の冷却材の沸騰や凝縮の挙動，すなわち2相流動挙動の詳細な解析が不可欠である．従来は，これらの解析に気相と液相が平衡状態にあると仮定して解析する手法（HEM モデル）を用いていたが，この手法では気相と液相が分離している状態や非平衡状態の正確な解析ができない．そこで近年，気相と液相とを独立に扱う2流体モデルが考案され，この手法を用いた2相流解析コードの開発が日本や欧米各国で行われた．

本稿において，高速増殖炉の冷却材であるナトリウムの沸騰解析用に開発された SABENA コードの1次元版について，その計算モデルや実験解析の結果について述べる．

2相流解析においては，沸騰や凝縮のような物理的に不安定な現象を扱うので，より安定化した数値計算手法の採用が非常に重要である．また，蒸発量や凝縮量，2相間の摩擦係数，界面熱伝達係数などの相関式には不確定な面があり，これをどのように選ぶかが計算結果に大きな影響を与える．

Abstract　To evaluate the performance of engineered safety features of nuclear reactors, it is necessary to analyse in detail the thermodynamic behavior of two-phase flow such as boiling and condensation of reactor coolant for specific accidents conditions. These phenomena have been analysed so far by HEM model assuming the thermodynamic equilibrium between liquid and vapor phases, but HEM model is inaccurate for the analysis of nonequilibrium effects.

The most recent approach for analysing two-phase flow is based on two-fluid model and many LWR (Light Water Reactor) safety analysis codes have been developed in U. S., Japan and European countries.

In this report we describe the numerical model (6-equation two-fluid model) and calculational results of SABENA-1D which has been developed for the analysis of sodium boiling in a fuel assembly of LMFBR (Liquid Metal Fast Breeder Reactor).

For the analysis of physically unstable phenomena such as boiling and condensation, it is important to select highly stable numerical methods and accurate constitutive relations for phase change rate, friction factors and heat transfer coefficients.

1. はじめに

原子炉の工学的安全施設の性能評価のためには，事故時の冷却材の2相伝熱流動の解析が重要である．このような流動挙動は，体系内の流体の質量・運動量およびエネルギの保存則を記述する方程式と，相変化量や摩擦力や熱伝達を表す相関式を組み合わせて解くことによって解析することができる．これらの解析のために，HEM (Homogeneous Equilibrium Model)，ドリフト・フラックス・モデルや2流体モデルが考案されている．これらのモデルが扱う保存方程式の個数と相関式の個数を表1にまとめた．表からわかるように，HEM モデルは扱う保存方程式および相関式の数が少なく簡便であるが，気液2相間の平衡状態を仮定しているので非平衡現象の予測精度は著しく減少する．一方 6-equation 2流体モデルは，原理的にはどのような2相流動も予測することができるが，扱う方程式数も相関式数も多くなり計算が複雑化する．また，2流体モデルは相関式の選び方によっ

て計算結果が大きく左右される．しかしながら，近年の計算機の計算能力の増大と各種相関式に関する知見が実験等によって蓄積されてきたことで，2流体モデルによる2相流解析が主流になりつつある．

このような背景で，昭和57年から動力炉・核燃料開発事業団の委託を受けて，高速炉のナトリウム沸騰解析用に2流体モデルによる2相流解析コード SABENA の開発に着手した．高速増殖炉は軽水炉に比べて低圧で運転されるため，2相状態における気液の密度差が大きくなり，気液の分離が進展して2相間の速度差が大きくなる．したがって，この場合の解析には2流体モデルが適している．一方，低圧系の沸騰解析は，気液の密度差が大きいため沸騰および凝縮時の体積変化および圧力変化が大きくなり不安定性が増加するため，高圧系に比べてむずかしいといわれている．

SABENA コードの1次元版は，2相流解析上の数値計算モデルを検討するために開発されたものであるが，種々の改良の結果，最適な数値計算モデルを考案することができた．

以下において，2相流の基本方程式，数値計算モデルおよび種々の実験解析の結果について述べる．

表 1　2相流解析モデル[1]

Table 1　Two-phase flow models

解析モデル	保存方程式				相関式						当該モデルを使用している軽水炉コード
					外部相関		相関			計	
	M	K	E	計	F_W	Q_W	Γ	F_i	Q_i		
HEM	1	1	1	3	1	1	0	0	0	2	RELAP4
ドリフト・フラックス	2	1	2	5	1	2	1	0	1	5	TRAC
2流体 (5-equation)	2	2	1	5	2	1	1	1	0	5	RELAP 5/MOD 1
2流体 (6-equation)	2	2	2	6	2	2	1	1	1	7	TRAC, THERMIT

Mは質量保存方程式，Kは運動量保存方程式，Eはエネルギ保存方程式である．
保存方程式数では，①混合流体 (Mixture) とし，②気相・液相を別々に考える．
F_W は壁面摩擦，Q_W は壁面熱伝達，Γ は相変化量，F_i は相間摩擦，Q_i は相間熱伝達である．

2. 基本方程式

2流体モデルにおける基本方程式は，気相と液相それぞれの質量，運動量およびエネルギの保存則を表す方程式と，壁面と各相および2相間の相互作用（相変化量，摩擦，熱伝達）を記述する項によって表される．

2.1 保存方程式

2流体モデルの保存式は，系内で局所的になりたつ個々の相の保存方程式を，時間的，空間的に平均化することによって得られる[2]．2相流の方程式には，気相の体積比率を表すボイド率という量が現れるのが特徴的である．

質量の保存

$$\frac{\partial \alpha_K \rho_K}{\partial t}+\frac{\partial \alpha_K \rho_K V_K}{\partial X}=\Gamma_K \tag{2-1}$$

運動量の保存

$$\frac{\partial \alpha_K \rho_K V_K}{\partial t}+\frac{\partial \alpha_K \rho_K V_K^2}{\partial X}+\alpha_K \frac{\partial P}{\partial X}=-F_{WK}-F_{iK}-\alpha_K \rho_K g \tag{2-2}$$

エネルギの保存

$$\frac{\partial \alpha_K \rho_K e_K}{\partial t}+\frac{\partial \alpha_K \rho_K e_K V_K}{\partial X}+P\left(\frac{\partial \alpha_K}{\partial t}+\frac{\partial \alpha_K V_K}{\partial X}\right)=Q_{WK}+Q_{iK}+H_{SK}\Gamma_K \tag{2-3}$$

ここで用いた記号の意味は次のとおりである.

k は v または l (v : vapor, l : liquid)

α_v はボイド率, α_l は $1-\alpha_v$

ρ_K は密度 (kg/m³), V_K は速度 (m/sec)

P は圧力 (Pa), e_K は内部エネルギ (J/kg)

Γ_K は相変化量 (kg/m³·sec)

F_{WK} は単位体積当たりの壁面摩擦力 (N/m³)

F_{iK} は単位体積当たりの相間摩擦力 (N/m³)

g は重力加速度 (m/sec²)

Q_{WK} は単位体積当たりの壁面熱伝達 (J/m³·sec)

Q_{iK} は単位体積当たりの相間熱伝達 (J/m³·sec)

H_{SK} は飽和エンタルピー (J/kg)

なお, 上式においては, 簡単のために粘性項や熱伝導の項は省略してある.

上記方程式の他に, 流体の物性を表す状態方程式を用いる.

$$\left.\begin{aligned}\rho_K&=\rho_K(P, T_K)\\ e_K&=e_K(P, T_K)\end{aligned}\right\} \tag{2-4}$$

ここで, T_K は各相の温度 (°K) を表す. 未知数の数は 10 個 (α_v, p, ρ_v, ρ_l, V_v, V_l, e_v, e_l, T_v, T_l), 方程式の数も 10 個であるため, 方程式系(2-1)～(2-4)は解くことができる.

2.2 ジャンプ条件

式 (2-1)～(2-3) に現れる変数の間には, 質量, 運動量およびエネルギを保存するために, ジャンプ条件と呼ばれる以下の式が成りたつ.

$$\Gamma_v+\Gamma_l=0 \tag{2-5}$$

$$F_{iv}+F_{il}=0 \tag{2-6}$$

$$Q_{iv}+H_{Sv}\Gamma_v+Q_{il}+H_{Sl}\Gamma_l=0 \tag{2-7}$$

式(2-5), (2-7)から相変化量 Γ_v が求まる.

$$\Gamma_v=\frac{-Q_{iv}-Q_{il}}{H_{Sv}-H_{Sl}} \tag{2-8}$$

2.3 各種相関式

前節の保存方程式を解くには, Γ_K, F_{WK}, F_{iK}, Q_{WK}, Q_{iK} などの諸量を, 基本変数 α_v, p, ρ_K, V_K, T_K などの関数として表現する必要がある. このような関数を相関式と呼び, 主として各種の実験データから求められる. 相関式は, 2相流動様式に大きく依存するので, 各流動様式に対応した相関式が種々考案されている.

図1に2相流動様式の代表的な例を示した. パイプの壁面から熱量を受けて, 蒸気の割合が増えてゆくに従って, 流動様式がどのように変化してゆくかを表している. 相関式にはいろいろ不確定性があるが, 以下においては, 実験解析に用いた代表的なものを記述する.

2.3.1 壁面摩擦力 F_{WK}

1) Rivard-Torrey モデル[3]

$$F_{WK}=\alpha_K(f_K\rho_K\alpha_K^2V_K^2/2D_h)\phi_K^2 \tag{2-9}$$

ここで,

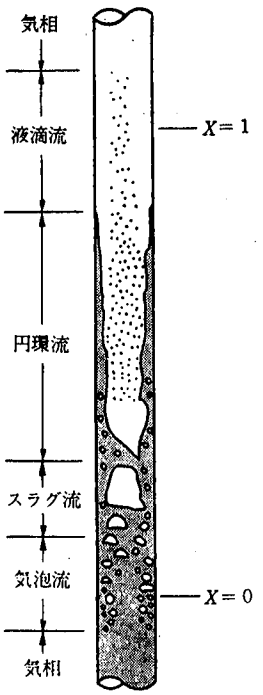


図 1 蒸発管の流動様式 (X はクォリティ (蒸気の質量比率))

Fig. 1 Flow patterns in a vertical evaporator tube (X: quality)

$$\left.\begin{aligned} &f_K=0.0055\{1+(20000\varepsilon/D_h+10^6/Re_K)^{1/3}\}\\ &\varepsilon/D_h=(\text{パイプの相対粗度})\\ &Re_K=\alpha_K\rho_K V_K D_h/\mu_K \quad (\mu_K:\text{粘性}) \end{aligned}\right\} \tag{2-10}$$

$\phi_l^2=1/\alpha_l^2$ (Lottes-Flinn parameter)

$\phi_v^2=X^2\phi_l^2$

$X^2=f_l\rho_l\alpha_l^2V_l^2/f_v\rho_v\alpha_v^2V_v^2$ (Martinelli parameter)

D_h＝等価直径＝4・流路面積/ぬれぶち長さ

2) 標準モデル

$$F_{WK}=f_K\rho_K V_K^2/2D_h \tag{2-11}$$

ここで，f_K は式(2-10)から求める．ただし，f_v は液相，気泡流 (Bubbly flow)，スラッグ流 (Slug flow) および管状流 (Annular flow) の場合には，0とする．

2.3.2 相間摩擦力 F_{iK}

1) Rivard-Torrey モデル

$$F_{ig}=-F_{il}=K_i(V_g-V_l) \tag{2-12}$$

ここで，各記号は次のとおりである．

$$\left.\begin{aligned} &K_i=8/3\,\rho_m A_i\{c_d|V_g-V_l|+12\nu_m/r_b\}\\ &\rho_m=\alpha_v\rho_v+\alpha_l\rho_l\\ &A_i=\begin{cases}\alpha_v^{2/3}(4\pi N/3)^{1/3}, & \alpha_v\leq 0.5\\ \alpha_l^{2/3}(4\pi N/3)^{1/3}, & \alpha_v>0.5\end{cases}\\ &N=10^7\ \text{bubbles/m}^3\\ &(A_i\text{ は，単位体積当たりの気液界面面積}) \end{aligned}\right\} \tag{2-13}$$

$c_d=0.5$ (Drag coefficient)

$\nu_m=\alpha_v\nu_v+\alpha_l\nu_l$ ($\nu_K=\mu_K/\rho_K$ は Kinematic viscosity)

$$r_b=\begin{cases}(3\alpha_v/4\pi N)^{1/3}, & \alpha_v\leq 0.5\\ (3\alpha_l/4\pi N)^{1/3}, & \alpha_v>0.5\end{cases}$$

2) Wallis モデル

$$F_{ig}=-F_{il}=K_i(V_v-V_l)$$

ここで，K_i は次のとおりである．

$$K_i=0.001\sqrt{\alpha_v}\rho_v|V_g-V_l|\{1+150(1-\sqrt{\alpha_v})\}/2D_h$$

2.3.3 相間熱伝達 Q_{iK}

気液2相間の熱伝達 Q_{iK} は次式で表される．

$$Q_{iK}=A_i h_{iK}(T_S-T_K) \tag{2-14}$$

ここで，A_i は気液界面面積（式(2-13)参照），h_{iK} は界面熱伝達係数 (J/m²·sec·°K)，T_S は飽和温度 (°K) である．

h_{il} は次式で与えられる．

$$h_{il}=\lambda\cdot\alpha_v\alpha_l\rho_l\sqrt{R_g/T_S}\cdot H_{fg} \tag{2-15}$$

ここで，λ は調整パラメタで 0.1，R_g はガス定数，H_{fg} は蒸発潜熱である．

h_{ig} は，気相温度 T_v を飽和温度 T_S 付近に保つため十分大きな値にする．

$$h_{ig}=10^4 \tag{2-16}$$

式(2-14)から Q_{iK} が求まると，式(2-8)から相変化量 Γ_K が求められる．

2.3.4 壁面熱伝達 Q_{wK}

壁面熱伝達については精度のよい相関式が種々考案されているが，以下の実験解析では Q_{wK} を無視できるので ($Q_{wK}=0$)，記述を省略する．

3. 数値計算モデル

流体方程式の近似解法として差分法がよく知られている．前節の保存方程式を semi-implicit 差分法で離散化して得られた非線形方程式系を，Newton-Raphson 反復法で解く手法について記述する．以下の方法では，質量，運動量およびエネルギの保存式を別々に逐次的に反復して解くのではなく，すべてを同時にカップリングして解くので非常に安定な数値解が得られる．

3.1 差分方程式

2相流の基本方程式には，時定数の非常に小さい変化（圧力波の伝幡，相変化）と大きい変化（移流項）が含まれている．このような場合，圧力および相変化を含む項を implicit に，移流項を semi-implicit に差分化することによって計算が安定化される．

また，staggered mesh（速度 V_K の計算点をメッシュ・セルの境界にとる）にして，移流項を上流差分することによっても計算が安定化される．このような差分法を用いることによって，タイム・ステップの制限はクーラン条件 ($V_K\Delta t/\Delta x\leq 1$) のみとなる．一例として，質量保存則(2-1)の差分式を以下に示す．

$$\frac{(\alpha_K\rho_K)_i^{n+1}-(\alpha_K\rho_K)_i^n}{\Delta t}+\frac{\langle\alpha_K^n\rho_K^n V_K^{n+1}\rangle_{i+(1/2)}-\langle\alpha_K^n\rho_K^n V_K^{n+1}\rangle_{i-(1/2)}}{\Delta x}=\Gamma_K^{n+1} \tag{3-1}$$

ここで，〈 〉は上流差分を表す．すなわち，

$$\langle Y^n V^{n+1}\rangle_{i+(1/2)}=\{f\cdot Y_i^n+(1-f)Y_{i+1}^n\}\,V_{i+(1/2)}^{n+1}$$

$$f=\begin{cases}1 & V_{i+(1/2)}\geq 0 \text{ のとき}\\ 0 & V_{i+(1/2)}<0 \text{ のとき}\end{cases}$$

メッシュおよび計算点の概念図を図2に示した.

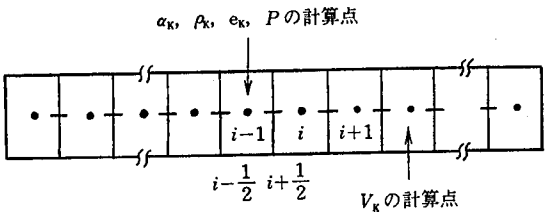


図 2 メッシュの概念図

Fig. 2 Mesh configuration

3.2 圧力反復法

運動方程式(2-2)を，移流項を explicit, P, F_{WK}, F_{iK} を含む 項をimplicit に差分化して整理すると，次の速度と圧力に関する方程式が得られる.

$$(V_K)_{i+(1/2)}{}^{n+1}=(\tilde{V}_K)_{i+(1/2)}+(d_K)_{i+(1/2)}(p_i{}^{n+1}-p_{i+1}{}^{n+1}) \tag{3-2}$$

ここで，$(\tilde{V}_K)_{i+(1/2)}$, $(d_K)_{i+(1/2)}$ はタイム・ステップ n の状態量から求まる既知量である.

式(3-2)を，質量およびエネルギに関する差分式に代入して $(V_K)_{i+(1/2)}{}^{n+1}$ を消去し，$(\rho_K)_i{}^{n+1}$, $(e_K)_i{}^{n+1}$ を状態方程式(2-4)を用いて $p_i{}^{n+1}$, $(T_K)_i{}^{n+1}$ の関数とみなすと，$4N$ 個 (N はメッシュ数) の質量およびエネルギ保存に関する差分式は，未知数 $(\alpha_v)_i{}^{n+1}$, $p_i{}^{n+1}$, $(T_v)_i{}^{n+1}$, $(T_l)_i{}^{n+1}$ $(i=1\sim N)$ に関する非線形方程式系となる．これを，通常の Newton-Raphson 反復法で解くと，$4N\times 4N$ の Jacobian 行列の逆行列を求めねばならず不経済であるため，$(V_K)_{i+(1/2)}{}^{n+1}$ が $p_i{}^{n+1}$ と $p_{i+1}{}^{n+1}$ のみに依存するという式(3-2)の性質を利用して計算手順を簡単化する.

反復の回数を r とし，

$$X_i{}^{r+1}=X_i{}^{r}+\Delta X_i \tag{3-3}$$

とおく．ここで，X は p, α_v, T_l, T_v を表す．式(3-3)をメッシュ i における質量およびエネルギに関する差分式に代入し，各項を増分 ΔX_i に関して Taylor 展開する．増分 ΔX_i に関する高次の項を無視して式を整理すると，次式が得られる.

$$A_i\begin{bmatrix}\Delta p_i\\ (\Delta\alpha_v)_i\\ (\Delta T_l)_i\\ (\Delta T_v)_i\end{bmatrix}+B_i\begin{bmatrix}\Delta p_{i-1}\\ \Delta p_{i+1}\end{bmatrix}=f_i \tag{3-4}$$

各記号は次のとおりである.

$A_i=4\times 4$ Jacobian 行列

$B_i=4\times 2$ Jacobian 行列

$f_i=4\times 1$ ベクトル

式(3-4)の両辺に A_i の逆行列をかけると，

$$\begin{bmatrix}\Delta p_i\\ (\Delta\alpha_v)_i\\ (\Delta T_l)_i\\ (\Delta T_v)_i\end{bmatrix}+C_i\begin{bmatrix}\Delta p_{i-1}\\ \Delta p_{i+1}\end{bmatrix}=g_i \tag{3-5}$$

ここで，記号は次のとおり．

$$C_i=A_i^{-1}B_i=\begin{bmatrix} c_{11}^i & c_{12}^i \\ c_{21}^i & c_{22}^i \\ c_{31}^i & c_{32}^i \\ c_{41}^i & c_{42}^i \end{bmatrix},\quad g_i=A_i^{-1}f_i=\begin{bmatrix} g_1^i \\ g_2^i \\ g_3^i \\ g_4^i \end{bmatrix}$$

式(3-5)の第1行目から圧力の増分 Δp_i に関する三重対角行列方程式

$$c_{11}^i\Delta p_{i-1}+\Delta p_i+c_{12}^i\Delta p_{i+1}=g_1^i \quad (i=1\sim N)$$

が得られる．これを与えられた境界条件のもとで解くと，Δp_i が求まる $(i=1\sim N)$．Δp_i が求まると，式(3-5)の2～4行目から，$(\alpha_v)_i$, $(\Delta T_l)_i$, $(\Delta T_v)_i$ が求められる $(i=1\sim N)$．

ついで，式(3-3)によって p, α_v, T_l, T_v を更新し，上記の手順を繰り返す．式(3-4)の右辺のベクトル f_i はメッシュ i における質量およびエネルギの残差を表しているので，反復はこの残差が十分小さくなるまで行う．

上記の反復法は，まず圧力増分を求め，それをもとにして他の変数の増分を求めているので圧力反復法と呼ばれる．

以下の実験解析ではどの計算でも，ほぼ4回の反復で残差が 10^{-7} 程度に減少し，非常に良好な収束性を示した．

3.3 境界条件

境界条件を与えるために，入口および出口境界の両端に仮想セルを設ける（図3）．

Inflow

1) 速度指定の境界条件

速度と圧力の関係式(3-2)において

$(d_v)_{1/2}=(d_l)_{1/2}=0$

$(\tilde{V}_v)_{1/2}=$指定速度

$(\tilde{V}_l)_{1/2}=$指定速度

$(\alpha_v)_0$, $(T_v)_0$, $(T_l)_0$ は入力で指定する．

2) 圧力指定の境界条件

p_0, $(\alpha_v)_0$, $(T_v)_0$, $(T_l)_0$ を入口で指定する．

Outflow

上流差分法を用いているので，出口の圧力を指定するだけで，すべての物理量が計算される．

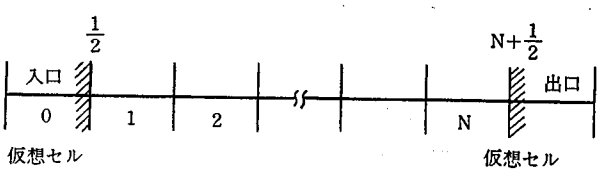


図 3 境界における仮想セル

Fig. 3 Fictitious cells at the boundary

4. 実験解析

SABENA コードは，もともとナトリウムの沸騰解析用に開発され，種々の実験解析を行ってきたが，ナトリウムに関する実験データは公開できないものが多いので，以下においては，水に関する2相流実験の解析結果について述べる．

4.1 Water-faucet flow

この問題は，水道の蛇口や樽の栓から流出し，自由落下する液体の形状および速度を求めるもので，摩擦力や相変化を無視すると解析解が得られるため2相流コードのチェック用に用いられる．問題の形状を図4に示した．

鉛直下方をx軸の正方向にとり，蛇口の出口の座標を$x=0$とすると，定常解析解は次式で与えられる．

$$\left.\begin{aligned} V_l(x) &= \sqrt{V_l(0)^2+2gx} \\ \alpha_v(x) &= 1-\frac{\alpha_v(0)V_l(0)}{V_l(x)} \end{aligned}\right\} \tag{4-1}$$

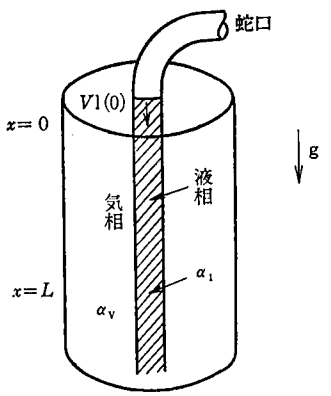


図 4 Water-faucet flow 形状

Fig. 4 Water-faucet flow geometry

この問題を以下のような初期条件と境界条件を用いて計算した．流速とボイド率の計算結果と解析解の比較を図5，6に示した．

1) 初期条件

圧力＝1 Bar

$V_l=3.0$ m/sec, $T_l=25$ ℃

$V_v=10^{-4}$ m/sec, $T_v=100$ ℃

$\alpha_v=\alpha_l=0.5$

2) 境界条件

L＝計算領域の長さ＝1.6 m

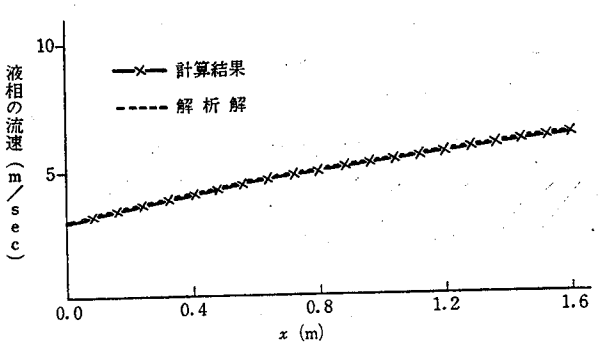


図 5 液相の流速

Fig. 5 Liquid velocity

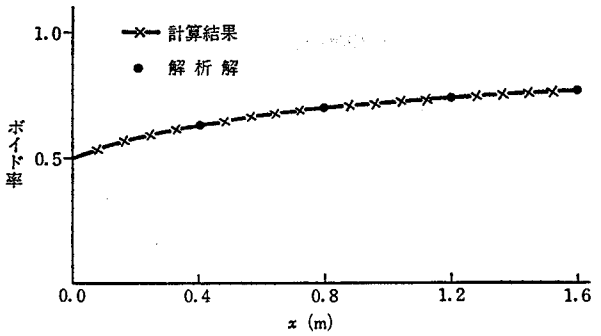


図 6 ボイド率の比較

Fig. 6 Void fraction

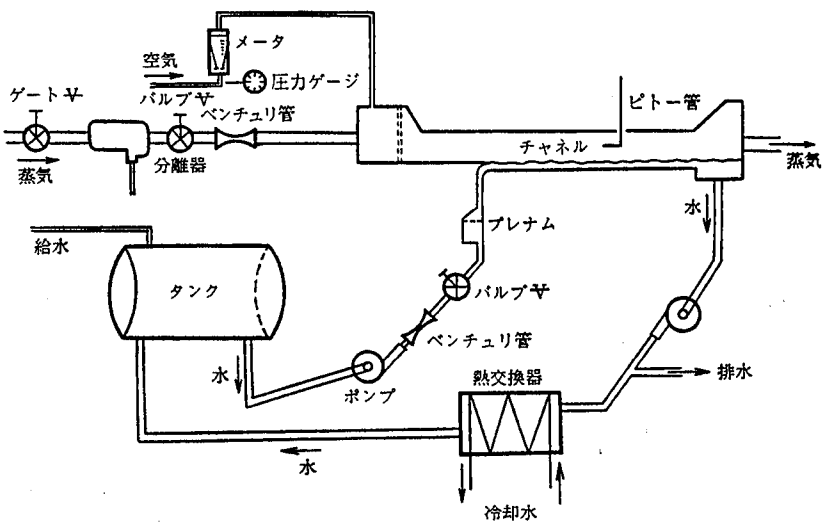


図 7 系の概念図

Fig. 7 Schematic diagram of the system

$$\begin{cases} x=0 : V_l=3.0\,\mathrm{m/sec},\ V_v=10^{-4}\,\mathrm{m/sec},\ \alpha_v=0.5 \\ x=L : P=1\,\mathrm{Bar} \end{cases}$$

4.2 蒸気凝縮実験解析

図7に，1981年米国の Northwestern 大学で行われた蒸気凝縮実験[4]の実験装置を示した．この実験は水平管の中で，高温の蒸気を水に接触させて流し，凝縮量を測定したものである．凝縮の際の相間熱伝達係数 h_{iK} については，まだよくわかっていない点が多いので，このような実験のデータから h_{iK} を推定しようという試みがなされている．

4.2.1 相間熱伝達係数 h_{il}

この実験の解析には，液相側の相間熱伝達係数 h_{il} として，実験データから求められた次の相関式を用いた．

$$\left.\begin{aligned} h_{il} &= \bar{N}_u \cdot \bar{K}_l / x \\ \bar{N}_u &= 0.0291 \cdot \bar{R}e_v^{0.58} \cdot \bar{R}e_l^{0.42} \cdot \bar{P}r_l^{0.3} \end{aligned}\right\} \qquad (4\text{-}2)$$

ここで，x は入口からの距離 (m)，$\bar{K}_l$ は液相の平均熱伝導率 (J/m·sec·°K)，$\bar{N}_u$ は平均ヌッセルト数，$\bar{R}e_K$ は平均レイノルズ数，$\bar{P}r_l$ は液相の平均プラントル数である．

$\bar{N}_u, \bar{R}e_K, \bar{P}r_l$ の計算法は繁雑[4]なので詳細は省略する．

4.2.2 計 算 条 件

1) 幾何形状

水平管の形状は，縦 6.35 cm，横 30.48 cm，長さ 160 cm である．したがって流路断面積 $A_f=1.94\times10^{-2}\,m^2$ となる．

2) 境界条件

人　口

$$W_l(0)=\rho_l V_l A_f=1.44\,kg/sec$$

$$T_l(0)=25\,℃$$

$$\alpha_v(0)=0.65$$

蒸気の流量 $W_v(0)=\rho_v V_v A_f$ と温度を変化させた次の 3 ケースの計算を実施した．

計算ケース	$W_v(0)$ (kg/sec)	$T_v(0)$ (℃)
ケース 1	0.065	116
ケース 2	0.090	121
ケース 3	0.126	125

出　口

圧力＝1 Bar

4.2.3 計 算 結 果

図 8 に各ケースの蒸気流量 W_v と凝縮量 W_c の計算結果を，図 9 に実験の結果を示した．W_c は蒸気凝縮量 Γ_c を積算したもので

$$\frac{dW_c}{dX}=A_f\Gamma_c$$

の関係がある．また $W_v(x)+W_c(x)=W_c(0)$ の関係も成りたつ．

計算値と実験値を比較すると，実験値の凝縮量のほうが計算値に比べて 30 パーセント程度大きい．これは，計算に用いた 2 相界面伝熱面積の値が小さかったためと思われる．

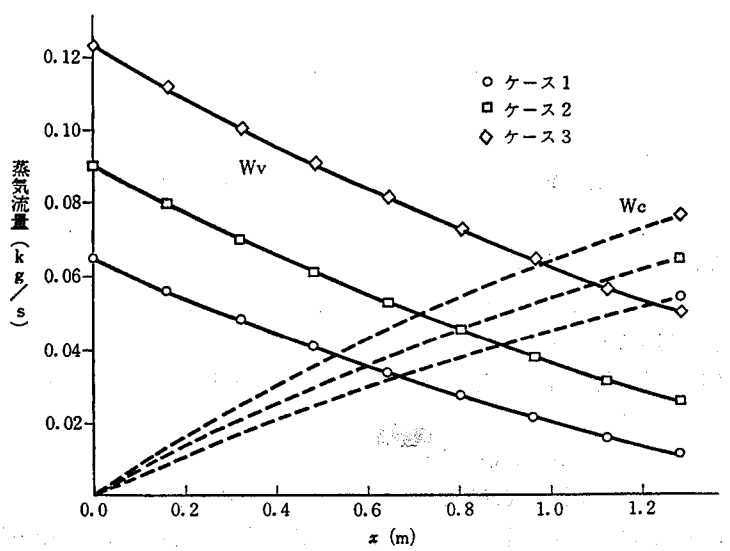


図 8　蒸気流量 W_v と凝縮量 W_c

Fig. 8　Steam flow rate W_v and condensation rate W_c

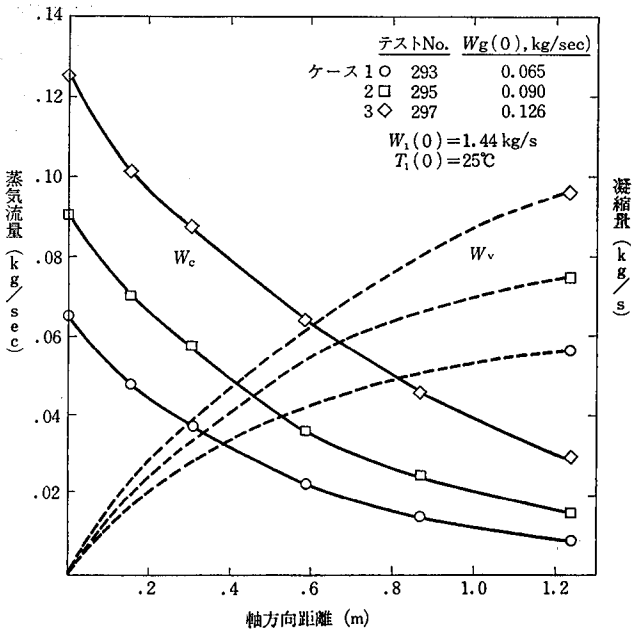


図 9 軸方向蒸気流量と凝縮量

Fig. 9 Axial steam and condensate flow rate profile as a function of inlet steam flow rate

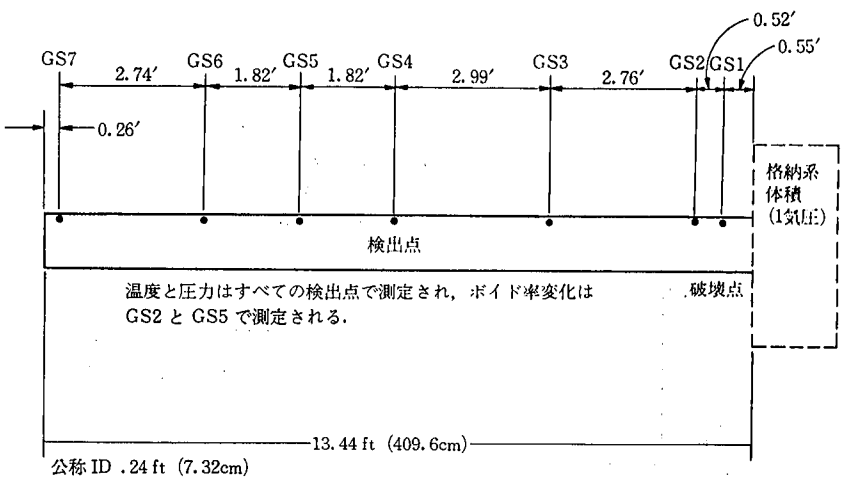


図 10 標準問題1 (Edward パイプ実験)

Fig. 10 Standard problem 1 (Edwards pipe experiment)

乱流状態になると気液界面に波動が生じ，伝熱断面積が大幅に増加するものと思われるが，水平分離流の界面伝熱断面積の相関式については，適当なものを見つけることができなかった．

また，入口付近（助走区間）の凝縮量の計算値が小さい点も検討の必要がある．

4.3 Edwards パイプ実験[5]

Edwards パイプ実験は，長さ 4 m 直径 7.3 cm の水平管に高温，高圧 (229 ℃, 70 気圧)

の水を封じ込め，管の一端を瞬時に破断して開いた時の水の流出挙動を計測したものである．破断後の圧力低下とともに沸点が低下して水が蒸気になり（フラッシング），2相流動となる．この実験は米国の NRC (Nuclear Regulatory Commission) Standard Problem 1 (標準問題 1) と呼ばれ，原子炉の配管破断の際の冷却材流動状況をシミュレートする計算コードの性能評価に用いられる有名な実験である．

4.3.1 計 算 条 件

1) 幾何形状

管の長さ＝13.44 ft＝4.1 m

管の内径＝2.88 in＝7.3×10^{-2} m

流路断面積＝4.2×10^{-3} m^2

なお圧力，温度，ボイド率などの計測点 (GS; Gauge Station) の位置を図 10 に示した．

2) 初期条件（パイプ内部）

p＝69.96 Bar

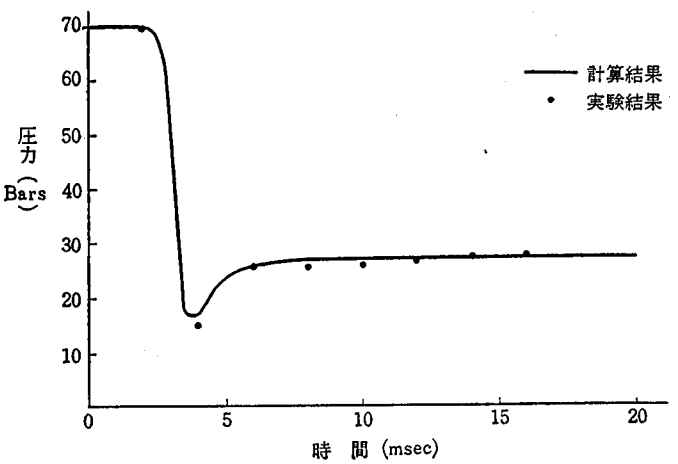


図 11 初期の圧力時間変化 (GS-7)

Fig. 11 Early time pressure history

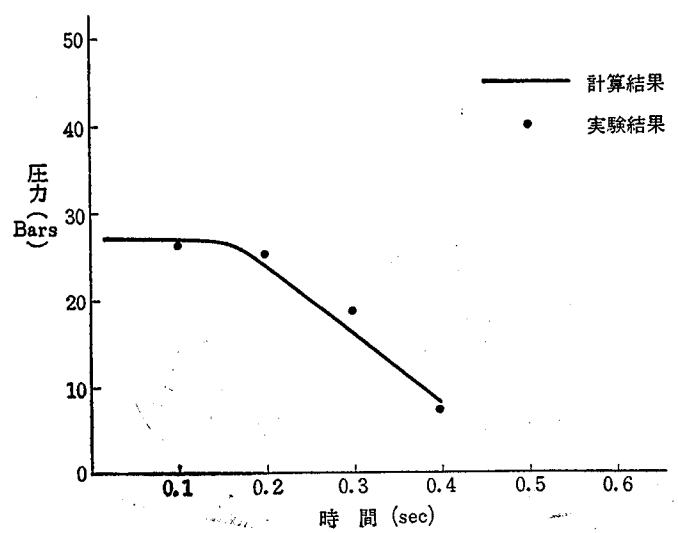


図 12 圧力の時間変化 (GS-7)

Fig. 12 Late time pressure history

$T_v = T_l = 502\,°K$

$V_v = V_l = 0$

$\alpha_v = 0$

3) 境界条件（パイプ出口）

$p = 1.0\,Bar$

$T_v = T_l = 373\,°K$

$\alpha_v = 1.0$

4.3.2 計 算 結 果

実験はほぼ0.6秒で終了するが，0.4秒までの計算値と実験値の比較を図11～14に示した．

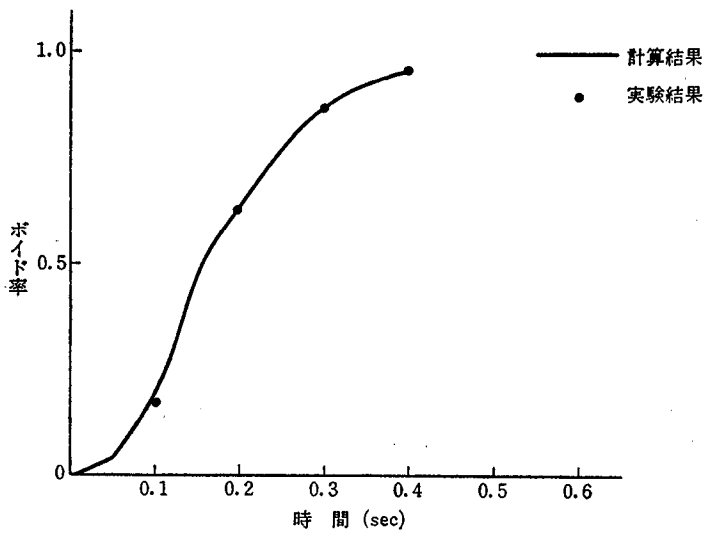


図 13 ボイド率時間変化 (GS-5)

Fig. 13 Void fraction history

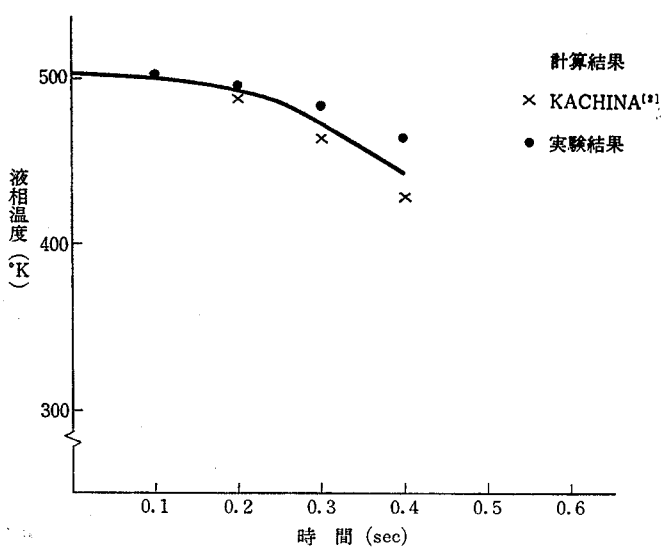


図 14 液相温度時間変化 (GS-5)

Fig. 14 Temperature history

計算結果と実験値とは非常によく一致している.

なお，パイプ出口付近では計算値と実験値に多少のずれが生じるが，これはパイプ出口で臨界流モデルを用いなかったためである.

5. おわりに

SABENA-1D コードは，不安定な2相流動を計算するのにどのような数値計算手法を用いるのが最適であるかを検討するために開発されたものであるが，上に述べた手法が最も妥当なものであることが，十分に証明された．この手法には次の利点がある.

- 計算手順が簡単である.
- 圧力反復法の収束性が非常によい.
- 2次元，3次元への拡張が容易である.

なお，本コードの3次元拡張版はすでに開発ずみであり，高速増殖炉のナトリウム沸騰解析において優秀な成果をあげている[6].

参考文献

[1] A. L. Schor and N. E. Todreas, "A Four-Equation Two-Phase Flow Model for Boiling Simulation of LMFBR Fuel Assemblies", MIT-EL-82-039, 1982.

[2] M. Ishii, "Thermo-Fluid Dynamic Theory of Two-Phase Flow", Eyrolles, Paris, 1975.

[3] W. C. Rivard and M. D. Torrey, "Numerical Calculation of Flashing from Long Pipes Using a Two-Field Model", LAMS-NUREG-6330, 1976.

[4] I. S. Lim, *et al.*, "Cocurrent Steam/Water Flow in a Horizontal Channel", NUREG/CR-2289, 1981.

[5] "Comparison of RELAP 4 Predictions with Standard Problems 1, 2 and 3", EPRI NP-205, 1976.

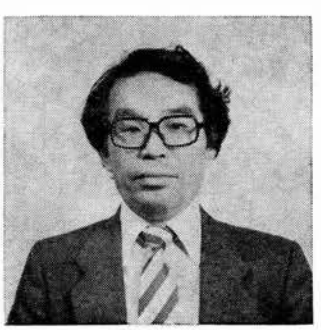
[6] H. Ninokata and T. Okano, "Results of the 11-th LMBWG Benchmark Calculation by SABENA-3D", LMBWG 11-th Meeting, 1984.

執筆者紹介 二ノ方　寿 (Makoto Ninokata)

昭和21年生．45年東京大学教養学部基礎科学科卒業，47年東京大学工学系大学院修士課程（原子力工学）修了．47～49年米国 Massachusetts 工科大学大学院原子力工学科留学，52年東京大学工学系大学院博士課程（原子力工学）修了．52年4月～55年5月東京電力(株)に勤務．55年6月動力炉・核燃料開発事業団大洗工学センタに出向．57年米国 Argonne 国立研究所を経て，現在動燃解析グループ・グループリーダ．LMFBR 熱流動解析コード開発・整備・改良等に従事.

岡野豊明 (Toyoaki Okano)

昭和42年東京大学理学部数学科卒業，44年東京大学理学系大学院修士課程（数学）修了，46年(株)日本ユニバック総合研究所入社，52年日本ユニバック(株)電力システム部に移籍．以来，原子力安全解析を始めとする種々の熱流動解析に従事.



報告 UNIVACシリーズ1100用Adaコンパイラ・システム

An Ada Compiler System for the Sperry Series 1100

G. Snyder, D. Wallace

要 約 Intermetrics 社は，Sperry 社の防衛機器部門との契約により，UNIVAC シリーズ 1100 用の高度最適化 Ada* コンパイラ・システムを開発中である．本稿では，Ada の主要機能の概略，コンパイラの使用者インタフェースと構造，並びに開発方法論について述べる．

Abstract Intermetrics, Inc. is building a highly optimizing Ada Compilation System (ACS) for the Sperry Series 1100, under contract to the Sperry Defense Products Group. This paper summarizes the major features of Ada, the user interface and architecture of the compiler, and the methodology of development.

1. はじめに

Ada を使う主な理由は二つある．一つは国防総省 (DOD) の要請であり，もう一つは，よいプログラミング言語だということである．

1.1 DOD の要請

新規の軍用アプリケーションに対しては，Ada の使用が公式に要請されている．これは，厳密に強制されているわけではなく他のプログラミング言語を使うための認可は，比較的容易に得られる．しかし，このような例外認可は，多くの機種で高品質の Ada コンパイラが使用可能になるにつれ，あまり行われなくなると思われる．すでに，Ada を使う契約になっている重要なプログラムがいくつかあり，次に示すのがその例である．

1) WIS (世界軍用管制システム情報システム—WWMCS (World-Wide Military Control System) Information System)
2) INEWS (統合電子兵器システム—INtegrated Electronic Warfare System)
3) Space Station (宇宙ステーション)
4) ATF (新戦術戦闘機—Advanced Tactical Fighter)
5) LHX (軽量ヘリコプタ—Light Helicopter)

1.2 Ada とは

Ada は，巨大システム用に設計されたプログラミング言語である．DOD 標準 (MIL-STD 1815A) であると共に，米国標準規格 (Ada LRM)[8] にもなっている．Ada の本来の目的は，組込み型計算機システムの設計，開発，および保守の費用を削減することにある．このため，Ada は DOD で使われている他のすべてのプログラミング言語を置き換えてゆくようになろう．組込み型計算機システムは，特徴として，①多数の開発要員 (5〜50 人)，②長期のライフサイクル (20 年)，③長期の開発期間 (2〜3 年)，④継続的な改良，⑤フィールドにおけるバージョン数の多さ，などの性質を備えている．

Ada は，これらの性質に対処するために設計されたが，他の設計作業と同様，設計上のトレード・オフが行われた．その結果，Ada は決して小さくも単純でもなく，ましてや完全でもないものとなった．しかし，組込み型のアプリケーションに限らず，巨大なソフト

* Ada は，アメリカ合衆国政府 (Ada Joint Program Office) の登録商標である．

ウェアの開発に適合したものとなっている.

1.2.1 Ada とは誰か

Ada という名前は，頭文字を集めた語ではない．この名前言語は，Lovelace 伯爵夫人で，かつ Alfred Byron 卿の娘である Augusta Ada Byron (1815～1852) に因んで名付けられたものである．彼女は，Babbage の解析エンジン用に Bernoulli 数を生成するアルゴリズムの詳細な記述を行った，いわば最初の計算機プログラマともいうべき人であった.

1.2.2 Ada の機能

Ada は，設計と開発の最新技法を支援する数々の機能を備えている．これらを要約すると，次のようになる.

1) 従来からの機能
   - 構造的な制御の流れ
   - 整数型，固定小数点型，浮動小数点型，論理型
   - 配列型，レコード型，アクセス型
   - ファイル入出力，テキスト入出力，装置入出力
2) 最新機能
   - 自己記述的な命名法と型付け
   - 広範囲な正当性検査
   - 分割翻訳
   - 例外処理
3) 抽象型と算体
   - パッケージ
   - 汎用体
   - タスク

1.2.3 Ada の自己記述機能

Ada は，部分的にせよ自己を文書化するようなコードを書くことを奨励する機能を有している．また，これらの機能によって翻訳時に広範囲な検査が可能である．たとえば，**列挙型**は値にそれ自身を説明するような記号や名前を与えられるので，コードがより読みやすくなる．副プログラムは，**名前付きパラメタ**を与えて呼び出すことができ，仮パラメタと実パラメタの対応付けがより明確になる．**部分型**は，前に宣言した型の変化形として新しい型を作る機構を用意している.

1.2.4 使用者の定義するデータ型

Ada では，使用者の定義する型について非常に柔軟性がある.

たとえば，配列は多次元が可能であり，副プログラムにおける配列の仮パラメタには制約がない．すなわち，副プログラムの側では配列の上下限を指定する必要がない．Ada では，**スライス**つまり配列の一部分に対する操作が可能である．配列の初期化や他の操作で，配列要素のリテラルを**集成式**の記法でいくつも書くことができる．**文字列**は，システムにおける既定義の配列型として扱われる.

Ada のレコードは，可変項目が可能で，制約付き，あるいは無制約の判別子で決定される．リテラルの場合は，レコードの値を集成式の記法で書いてもよい.

アクセスによって，強い型付けのポインタを利用できる．このため，プログラマがあるアクセス型の値を特定の寸法を持つ別のヒープ（集り）として宣言することなどができ，動的な記憶域の解放についても，ある程度の制御が行える.

列挙型，整数の部分域型，浮動小数点型および固定小数点型については，直接読み書きが可能であり，またパッケージ TEXT_IO を用いて，文字列の形に変換することもできる．

型，部分型または派生型の宣言によって，変数の適合性や型変換の必要性についてプログラマが制御できる．

1.2.5 正当性検査

Ada は，プログラムの正当性について，翻訳時および実行時に多くの検査を行う．

翻訳時には，型の適合性について検査される．これには，強い型付けのポインタも含まれている．副プログラムやタスクの呼び出しでは，仮パラメタと実パラメタの適合性が検査され，関数呼び出しでは返す値の型について適合性が検査される．Case 文の並びについては，実行時に予想外の値がくることがないように包含性が検査される．

実行時には，計算結果がすべて計算中の型の制約内に納まるかどうか検査される．数値の正当性については，桁あふれや下位桁あふれ，固定小数点数と浮動小数点数の精度落ち，などについて検査される．これらの検査は，最終結果だけでなく中間結果についても行われる．実行時にこれらの誤りが検出されると，Ada は**例外**（後述）を発生させる．

Ada では翻訳時に広囲にわたる検査を行っているためプログラムが正しく翻訳されると，そのまま正しく動くことが多い．たとえば，AIE コンパイラ[1]の意味解析部は1万行のコードから成っているが，最初の連結操作で正しい出力を生成し始めた．10個のフェーズから成る AIE コンパイラの統合作業においては，各個別テストの後，わずか10日間を費したにすぎない．

1.2.6 パッケージと分割翻訳

分割翻訳には2種類ある．WITH 句を用いた**横断的**なものと，IS SEPARATE を用いた**縦断的**なものである．WITH 句では，分割翻訳した汎用のユティリティ・パッケージや特定プログラムのモジュールなどの翻訳単位を参照できる．また，どの副プログラムの本体も SEPARATE として宣言でき，分割翻訳できる．それでもなお，論理的にはそう宣言したパッケージの一部であり続ける．

分割翻訳のシステムには，複数回の翻訳にまたがる完全な型の検査と，再翻訳の規則が含まれている．Ada は，翻訳したり，あるいは翻訳することを助言して首尾一貫性を保つようなライブラリ管理のソフトウェアの存在を**想定している**．たとえば，パッケージの仕様が変更されると，そのパッケージを WITH 句で参照している翻訳単位を再翻訳する必要が生じることもありうる．このため Ada は，再翻訳の必要性を判定する洗練された更新解析を可能にしている．

Ada のパッケージは，データと副プログラムのカプセル化 (encapsulation) を許し，データの抽象化能力の一部を担っている．パッケージの概念は，**算体指向型設計**[2]のような開発方法にとってとくに重要なものである．

Ada のパッケージは，仕様と本体を分離できる構造をもっており，型の定義を公開部 (public) と密閉部 (private) に分離できる．外部からは仕様が見えるので，再翻訳過程の助けとなる．一般的に，本体だけの変更では他のパッケージの再翻訳は必要でない．

1.2.7 汎用体とその具体化

汎用パッケージを使えば，アルゴリズムを特定のデータの型から独立させることができる．アルゴリズムは通常どおりコード化されるが，データの型はパラメタとして指定し，後に汎用体を具体化するときに与えればよい．

汎用体は何度も具体化されるデータやプログラムを静的にグループ化する機能を提供する．パッケージが**抽象データ算体**とみなされるのに対し，汎用体パッケージは**抽象データ型**とみなされる．

1.2.8 例外ハンドラ

例外処理は，ハードウェアとソフトウェアの誤りに応える構造的な機構を提供する．プログラマは，Ada で決められている例外や使用者の定義による例外に対し，容易にハンドラを書くことができる．発生した例外は，その例外に対するハンドラが見つかるまで有効範囲の上方レベルに向かって**伝播してゆく**．使用者がハンドラを用意していなければ，省略時用のハンドラにより，プログラムが終了させられる．高品質のコンパイラであれば，例外が実際に発生しない限り，効率上の負荷はかからない．

Ada の例外は，多重タスクの環境においても，誤り処理の方法を正確に定義できることに注意したい．

1.2.9 タスク実行

Ada のタスク実行はメッセージ主導であり，いくつかの利点を持っている．共有の記憶域は不要である．しかし，共有記憶域はパッケージ・データを介して，もちろんアクセス可能である．同期作業は**ランデブー中**に暗黙のうちに行われ，手続きのパラメタ同様にパラメタを受け渡すことができる．

複数選択肢を持つ SELECT 文によって，同時に複数のメッセージを待つことができる．

同期のとれたアクセスを必要とするデータを，タスク本体にカプセル化することができ，タスクをレコードのロック機構として使ったり，**算体フロー法**[4]のような方法論を実現するのに使うことができる．

タスク型は，バッファ・タスク，非同期応答タスク，資源管理モニタなどを作るために使うことができる．

1.2.10 コード最適化

Ada は標準的な最適化技術をすべて取り入れている．これらは，次のものを含んでいる．

1) 冗長な式の削除
2) ループ内不変式のループ外へのコード移動
3) 定数の折りたたみと伝播
4) 演算子の強さの軽減
5) 到達不能なコードの除去

さらに，Ada は次のような言語特有の最適化の可能性を用意している．

1) 副プログラムのインライン展開
2) 具体化された汎用体間でのコードの共有
3) 制約検査の除去
4) タスク関連の最適化

2. コンパイラの使用法

1100 ACS コンパイラは，Intermetrics 社が開発した AIE コンパイラ[1]をシリーズ 1100 上に移植したものである．AIE コンパイラは，大きな能力を備えた，効率のよい，高度な最適化 Ada コンパイラである．とくにプログラマの誤りに対して親切に対応するよう設計されている．

2.1 シリーズ1100特有の機能

1100 ACS コンパイラは，UNIVAC シリーズ1100に特有の機能をいくつか備えている．

1) バンク・サイズの制御……コード用のI-バンクと，ヒープや副次的なスタック用のD-バンクの大きさを使用者が指定できる．二つ目のI-バンクは静的なデータ，残りのD-バンクは基本スタックに使われる．
2) 自動ページング……コード領域，副次スタックおよびヒープは，ページング（バンキング）される．
3) Pragma インタフェース……ASCII FORTRAN (FTN), ASCII COBOL (ACOB) およびアセンブラ (MASM) に対するインタフェースが含まれる．
4) Pragma UCS-Representation……UCS Pascal, FORTRAN および COBOL と適合するデータ割り付け方法を用意し，これらの言語間で二進化形式のファイルが共有できるようにする．
5) Ada 入出力……端末，プログラム・ファイル中のエレメント，SDF ファイルおよび磁気テープに対応した入出力が用意される．
6) 算術データ……18ビット整数，36ビット整数並びに固定小数点数，36ビットと72ビットの浮動小数点数が支援される．
7) ブレーク・コンティンジェンシ……Ada の割り込みを引き起こすので，タスクのエントリを用意することで使用者が取り扱える．

2.2 コンパイラの効率

コンパイラの効率測定は，専用の UNIVAC シリーズ1100システム61の上で行う予定である．性能測定用のテスト・プログラムは，Sperry 社から提供される予定である．

翻訳速度の目標は，経過時間1分につき Ada の文500である．

命令語の展開目標は，Ada の文一つにつき7命令語である．

実行速度の目標は，同値なアセンブラ言語の25パーセント増以内である．ただし，同値性の定義には，次のものを含むものとする．

1) 同一の動作……アセンブラ・プログラムは，同一の使用者インタフェースを持つ．さらに，同じ入力データに対しては同じ出力結果を返す．これには，数値の桁あふれなどの誤りや例外に対しても同じ応答を返すことを含む．
2) 検査……必要なすべての制約検査を行う．不要な検査は行わない．
3) サービス……入出力のような実行時のサービスは，汎用のルーチンを使って行う．
4) プログラムの大きさ……比較には，大きなプログラムが望ましい．
5) プログラムの作成スタイル……悪いプログラム手法は使わない．

2.3 1100 ACS のコンポーネント

1100 ACS は，①Ada コンパイラ，②プログラム・ライブラリ（後述），③プログラム・ビルダ，④他のツール（プログラム・ライブラリ・マネジャ，翻訳単位リスタ），などのコンポーネントから成る．

2.4 Ada プログラム・ライブラリ

Ada のプログラム・ライブラリは，翻訳済みの Ada 翻訳単位を含むデータベースである．ライブラリの構造は，任意の翻訳単位が，WITH 句によって他の翻訳単位と一緒になれるという事実を反映していなければならない．また，単に目的コードや目的モジュールだけでなく，意味的な情報もライブラリに含まれる．これは，Ada 言語の定義から，複数の翻訳単位にまたがる検査のために必要となる．副プログラムのパラメタ，大域的な参

照，型の定義などがそのような検査を必要とする機能の例である．

2.5 ツールの開発

Intermetrics 社は，Ada のツール群を拡張中である．これらのツールは，原始プログラムそのものではなく，プログラム・ライブラリ中の情報を利用するものである．したがって，構文解析や意味解析を繰り返す必要がない．ツール群は，次のカテゴリに分類される．

1) Byron*……プログラムの設計・文書化・保守を支援するプログラム開発言語 PDL および，プログラム・リスタや相互参照ツールなどの支援ツールの集まりを含むプログラミング支援環境[7]で構成される．
2) Ada コンパイラ群……多種のホストと目的機械ごとに用意されている．
3) プログラム・ライブラリ・ツール群……これらは，構成管理，コードの共有や再使用，自動翻訳の支援などを含む．
4) 解析ツール群……①Halstead や McCabe の複雑性の尺度，②ライブラリ検索，③要求仕様の追跡，④データ辞書，⑤大域的な相互参照，を含む．
5) 検査とデバッグ用ツール群……①デバッガ，②時間解析，③検査網羅性チェッカー (Test Coverage Checker)，④原始プログラムの計測ツール (Instrumenter)，を含む．
6) コンパイラ生成ツール群……①IDL (Interface Description Language) ツール，②PQCC (Production Quality Compiler-Compiler) 表ビルダ，③Bonsai プログラム・ビルダ (木構造のパターン照合に基づく)，④構文解析表ジェネレータ，⑤診断用ツール，を含む．

2.6 使用者インタフェース

UNIVAC シリーズ 1100 上で Ada プログラムを実行するには，次の 3 段階を踏む必要がある．

1) Ada の翻訳単位を翻訳して，ライブラリに入れる．
2) ライブラリから，実行可能プログラムを作る．
3) プログラムを実行する．

翻訳の例を示そう．次の指令語で (必要に応じて MYLIB というライブラリを作り)，Ada のソース・エレメントを翻訳する．

```
@ADA MYFILE. SUB, MYLIB
```

次の指令語は，翻訳単位がすでにライブラリ中にあることを仮定した場合のものである．

```
@ADA, I MYLIB
Main_System_Driver
@EOF
```

次に実行可能プログラムの作成例を見よう．PBUILD は，内部でコレクタ (@MAP) を呼び出す．

```
@PBUILD MYFILE. MAIN, MYLIB, MYFILE. MYPROGRAM
```

または，

```
@PBUILD, I , MYLIB, MYFILE. MYPROGRAM
Main_System_Driver
@EOF
```

プログラム実行のための指令語は，次の形となる．入力データは，Ada の標準入力ファイル Standard_Input であるとする．

---

* Byron は，Intermetrics 社の登録商標である．

@XQT MYFILE. MYPROGRAM
入力データ
　:
@EOF

2.6.1 Ada コンパイラの呼び出し

Ada コンパイラの呼び出し用指令語は，次の一般形を持つ．

@ADA [，オプション文字列] [原始プログラム]，[ライブラリ] [，-オプション，値]…
[Ada 翻訳単位名]
[オプション=〉値]…
[@EOF]

具体例を挙げると，

@ADA, LY　XYZ. MYSOURCE, MYLIB

または

@ADA, I MYLIB
My_System_Driver
LIST=〉Assembly
@ADD MYOPTIONS
@EOF

2.6.2 プログラム・ビルダの呼び出し

プログラム・ビルダの呼び出し用指令語は，次の一般形を持つ．

@PBUILD [，オプション文字列] [主プログラム]，[ライブラリ]，アブソルート・エレメント [，-オプション，値]…
[主プログラム翻訳単位名]
[オプション=〉値]…
[@EOF]

具体例を挙げると，

@PBUILD MYFILE. MAIN, MYLIB, MYFILE. MYPROGRAM

または

@PBUILD, I　, MYLIB, MYFILE. MYPROGRAM
Main_System_Driver
@EOF

2.6.3 プログラム・ライブラリ・マネジャの利用

プログラム・ライブラリ・マネジャは，主に構成管理者が，カタログや連結編集についての各種操作を行うときに使われる．作成，削除，連結編集，連結解除，複写，印刷，派生，格上げなどの操作が含まれる．

2.6.4 プログラム・リスタの利用

リスタは，翻訳中あるいは翻訳後に，プログラム・ライブラリから直接，翻訳単位のリストをとる．プログラム・ライブラリがあれば，全翻訳単位のリストを，翻訳し直さないでも再生できる．また，プログラム・ライブラリ中にはDIANA 中間言語や目的モジュールと一緒に，翻訳情報や統計情報も格納されている．統計情報はリスタで検索でき，翻訳し直さないでもプロジェクトの進捗状況を監視することができる．リスタの呼び出し用指

令語の一般形は次のとおりである.

@PLIST [, オプション文字列] [原始プログラム] [, ライブラリ]

2.6.5 誤り診断メッセージと修復

構文上の誤りに対する正確な診断は，とくに重要である．まず誤りの発生場所とその性質を正確に報告すべきである．次にパーザーは，2次的な誤りが生じないように誤りから回復しなければならない．このほか，パーズ時のよい修復処置は，よい誤り診断メッセージ以上に手助けとなるものである．

ACS コンパイラは，誤り診断メッセージと修復処置について，Burke と Fisher[3] の方法を採用している．NYU Ada-Ed コンパイラでの経験によれば，111行の Ada プログラムの翻訳実験で，50以上の正しい診断を行ったということである．

ACS コンパイラの意味解析上の誤り処置は，可能な限りデバッグの助けとなるように設計されている．各メッセージは，誤りを含む原始プログラムの行を示し，誤りの性質を述べ，さらに Ada 言語参照マニュアル[8]の適当な章や段落を参照してくれる．

意味解析上の誤り処置機構は，モジュール構造を持ち，柔軟性がある．コンパイラは，

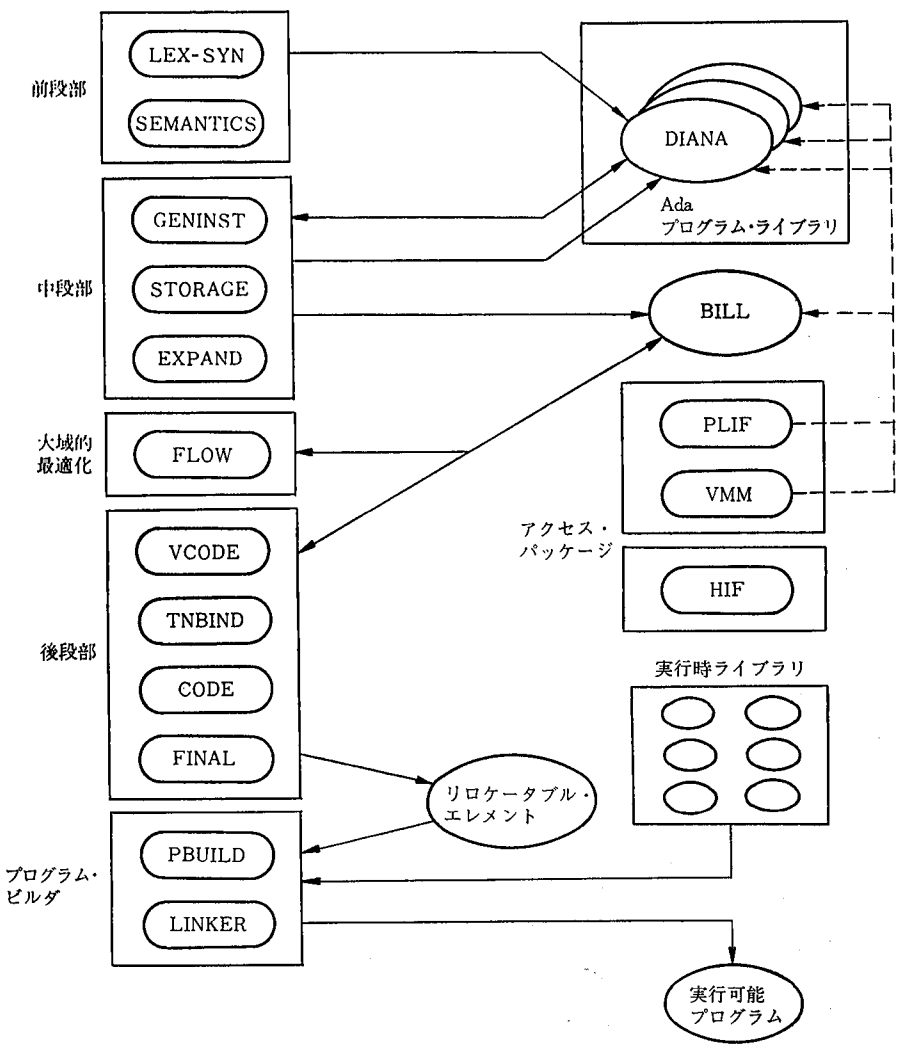


図 1 コンパイラの制御の流れ

Fig. 1 Compiler flow diagram

意味上の誤りを，宣言，文，および式単位で定義している．各意味上の誤りに対して唯一つの例外処置が対応し，どの誤りもどれかのレベルで処理される．誤り処理ルーチンは，同じ例外を再発生させたり別の例外を発生させたりすることがあり，複数レベルでの対応を引き起こすかもしれない．たとえば式中の誤りは，それを含む文を読み飛ばすような対応となるかもしれない．

3. コンパイラの構造

ACS コンパイラは，図1に示すように，いくつかのフェーズから構成される．

前段部は字句解析，構文解析および意味解析を行い，その結果を使用者のプログラム・ライブラリに DIANA（後述）と呼ぶ木構造の中間言語の形で格納する．

中段部は，汎用体の具体化や変数の記憶域割り当てを行い，木構造を UNIVAC シリーズ1100の実行時のモデルに対応した形に拡張する．その結果は，DIANA の低水準の部分集合である BILL の形で格納される．

後段部は，この木構造をコードの列に展開し，レジスタを割り当てて DIANA の変形である OBJMOD と呼ぶ目的モジュール（リロケータブル・エレメント）を作り出す．プログラム・ビルダは，いくつかの目的モジュールを連結編集して，実行可能プログラムを作る．このとき，古い Ada の翻訳単位をいくつか翻訳し直したり，まだ完成していない翻訳単位用の仮の目的モジュールを作ったりする作業を伴うこともある（図2）．プログラム・ビルダは，その最終段階でコレクタ（@MAP）を呼び出す．

プログラム・ライブラリは，PLIF (Program Library InterFace), VMM (Virtual Memory

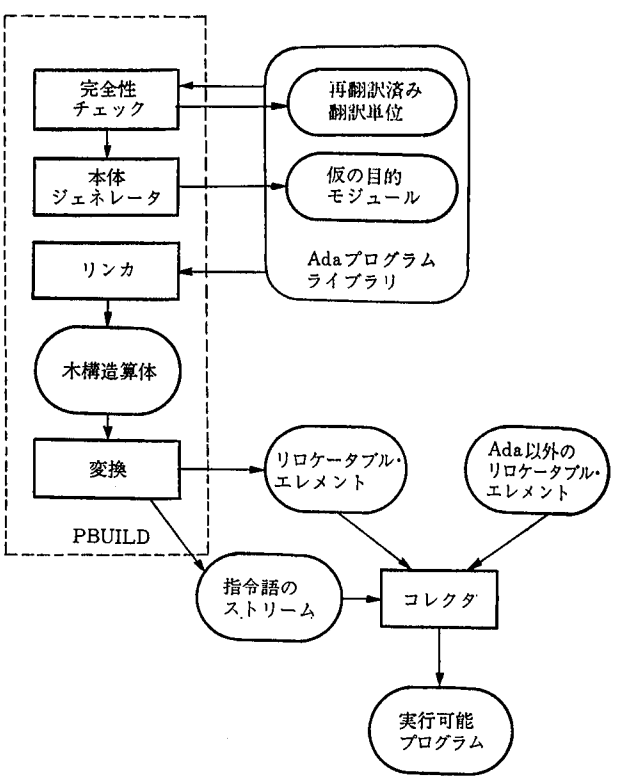


図 2　プログラム・ビルダの制御の流れ

Fig. 2　Program builder flow diagram

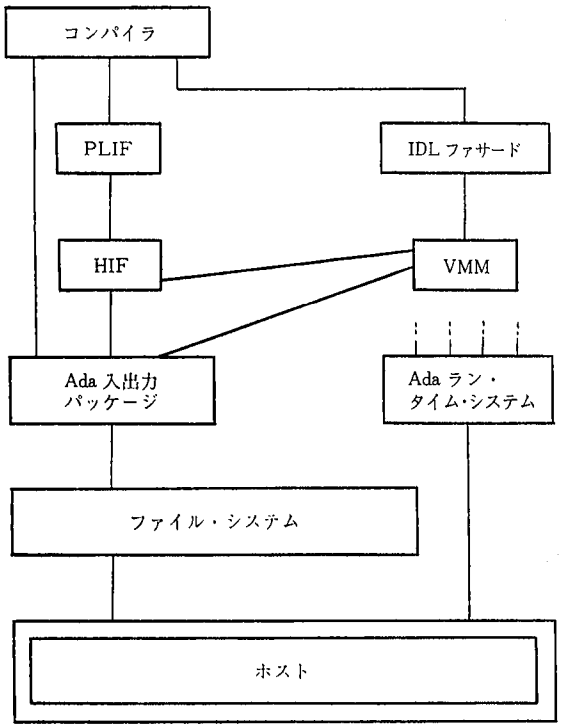


図 3　コンパイラ・インタフェース

Fig. 3　Compiler interfaces

Methodology), および HIF (Host InterFace) というパッケージを通してアクセスされる. これらとコンパイラの相関関係を図 3 に示す. なお, 以下に図中の用語について略述する.

PLIF—プログラム・ライブラリ・インタフェース

HIF—ホスト・インタフェース (CAIS (Common APSE Interface Set) の部分集合): プログラム・ライブラリ用データベース

IDL ファサード—中間記述言語フロント：VMM に対するインタフェース

VMM—仮想記憶法：ソフトウェア・ページング・システム

Ada 入出力パッケージ：アプリケーション・プログラムで用いる

Ada ランタイム・システム—実行時システム：アプリケーション・プログラムで用いる

3.1 DIANA

DIANA とは, "Descriptive Intermediate Attributed Notation for Ada" の略であり, 意味解析の結果として作られる Ada 用の中間言語である. 高水準言語に依存しない記法を採用し, プログラムの木と記号表を含んでいる. 木構造の形をし, 各節点に属性を含んでいる. 外部表現として, ASCII 文字で表現でき, Ada で定義できる.

DIANA は, Ada の形式的定義における抽象構文木に基づいている. これは, Carnegie-Mellon 大学の TCOL (Ada) と Karlsruhe 大学の AIDA[5] の結合により生まれた.

3.2 最 適 化

1100 ACS は, 高度な最適化 Ada コンパイラである. 大域的な制御の流れの最適化, およびコンパイラと実行時のライブラリの総合的な設計を行うことによって, 優秀なコード品質を達成している.

他の言語に比べて, Ada では最適化コードの生成に関し, 困難な面が四つある. 第 1 に

一つの文中にいくつかの制約検査が必要な場合，想像以上に多くのコードを生成しがちである．第2にインライン副プログラムを単純に展開すると，モジュール性のおかげで高価につき，Adaの基本目標の一つを損う．第3にタスクを単純に実現すると，ある種の同期操作で効率が悪化しすぎる．第4に汎用体の具体化では，汎用体本体の再翻訳に費やす時間や，重複したコードで費やす記憶域を最適化する必要がある．

3.2.1 組み込まれた最適化

コンパイラの基本設計で組み込まれている最適化に次のものがある．

1) PQCC コード生成……この方法は，テンプレート方式やパーザーに基づくコード生成方式より優れており，Carnegie-Mellon 大学で開発された PQCC に基づいている．
2) CASE 文……選択肢が密な場合には分岐ベクトルを生成し，粗な場合には IF-THEN-ELSE 構成をとる．
3) 例外ハンドラ……例外に対して翻訳時に対応表を作っておき，例外発生時にハンドラの存在場所を調べるとき利用する．この方法では，実際に例外が発生したときだけしか，負荷がかからない．
4) 汎用体……汎用体の具体化で共通なコードは共有するようにし，原始プログラムから翻訳し直すことは避ける．汎用体を DIANA（中間の木構造表現）として格納しておき，具体化のたびに原始プログラムを見るようなことはしない．
5) 配列とレコードの参照……常にインライン・コードとし，実行時ライブラリを呼び出さない．
6) 定形の配列……配列の限界が翻訳時にわかるものは，ドープ・ベクトルを生成しない．
7) レジスタ割り当て……大域的なレジスタ割り当てにおいて，先読みと後読みの効果を統合する“仮想レジスタ”を追跡して効果的に割り当てる．
8) レジスタ・パラメタ……副プログラムの一部あるいは，すべてのパラメタをレジスタで渡す．
9) タスク実行の最適化……ACS では，タスクの実現にいくつかの最適化を適用している．ランデブーの相手タスクに対するスケジュール上の負荷は，どちらかが待ち状態で，二つのタスクのいずれかが最優先の実行可能状態であることを利用し，ほとんど解消している．Habermann-Nassi[6] の方法により，ランデブー・タスクを呼び出し側のスタックに積み，呼び出しとパラメタ受け渡しの効率は，手続き呼び出しと同程度にしている．さらに，アクセプト文の本体の外側で重要なことを何もしないタスクについて，使用者が「監視タスク」と宣言できるようにした．このタスクでは，スケジュール上の負荷やほとんどすべてのスタック用記憶域が不要となる．

3.2.2 制御の流れと大域的最適化

大域的な最適化は，記憶域の割り付けとアドレス計算の後に行われる．これにより，添え字や他のアドレス計算において，共通部分式の検出が可能になる．

制御の流れの解析もインライン展開の後で行われ，副プログラムの最適化はそのインタフェースにも及ぶ．したがって，展開済みのコード中の共通部分式の検出や到達不能コードの除去が可能となる．これは，翻訳時に値がわかるパラメタを持つインライン展開の場合にとくに効果が大きい．

最適化の処理ではブロック構造ごとの最適化だけでなく，手続きにまたがった解析も行

われる．次の最適化処理が行われる．

1) 制約検査の除去
2) 定数の折りたたみ
3) 定数の伝播
4) 共通部分式の除去
5) 一定不変の関係式
6) 到達不能コードの除去
7) ループ不変部のループ外への移動
8) 演算子の強さの軽減：配列参照において，とくに重要
9) 代数的簡約化
10) 分岐の簡素化

3.2.3 制 約 検 査

制約検査の除去は，制御の流れを解析する段階で行われる最適化の好例である．最適化をしないと，すべての代入操作，添え字やポインタの参照で制約検査が必要となり，その負荷はとうてい受け入れられないものとなる．

しかし，制約検査は Ada の設計に取り入れられた信頼性向上のための重要な要素である．PRAGMA SUPPRESS を使えばもちろん制約検査を省けるが，それでは実行時に，**検出不能の誤り**を引き起こす可能性が高くなる．PRAGMA SUPPRESS を使うのは，ヒューズの代りに銅貨を使うようなものである．

ACS Ada コンパイラでは，各算体についてわかっている情報を追跡し，多くの制約検査を翻訳時に除去する．たとえば，ある変数から同じ型の他の変数への単純な代入では，元の変数が正当な値を含んでいることがわかっているので，制約検査は不要である．宣言で初期値を持つ変数を使用するときには，初期値が検査済みであるから制約検査は不要となる．範囲付きの二つの整数の加算では，範囲が大きいものでない限り，整数あふれの検査は不要である．

除去できずに残った暗黙の制約検査は，翻訳時の情報として使用者に報告される．設計に注意すれば，そのような検査のほとんどは取り除けるだろう．実際，暗黙の制約検査が残るということは，コード作成上の欠点を指摘してくれるものとみなしてよい．

3.2.4 最適化の利点

ある種の最適化は劇的な改善をもたらすが，中にはそうでないものもある．次に示すのは最適化を期待度から分類したものである．もちろん，アプリケーションによってバラツキがあることに注意されたい．たとえば，多次元配列を多く使うプログラムでは，演算子の強さの軽減や共通部分式の除去が大きな効果をもたらす．

1) 効果が非常に高いもの
   - レジスタの追跡
   - 制約検査の除去
   - Habermann-Nassi のタスク・ランデブー
2) 効果が高いもの
   - 基底に相対的な番地付け
   - 到達不能コードの除去
   - レジスタ・パラメタ
   - 汎用体の共有

3) 効果が中程度のもの
- 定数の折りたたみと伝播
- 共通部分式
- ループ不変部の移動

4) 効果が低いもの
- Case 文の粗密解析
- 演算子の強さの軽減

4. 開 発 方 法

4.1 AIE コンパイラ

AIE コンパイラは，図4に示すように Ada 言語を用いてブート・ストラップの技法により開発された．開発環境は IBM 3083 の UTS（Amdahl 社製の UNIX* 環境）である．

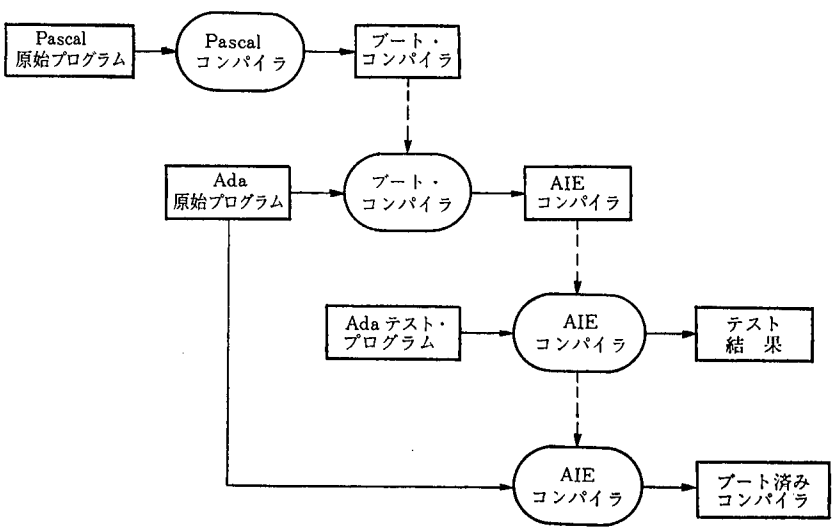


図 4　AIE コンパイラの開発

Fig. 4　AIE compiler development

第1段階では，Pascal を用いて，Ada のサブセット・コンパイラである“ブート”コンパイラを作成した．AIE コンパイラは，この Ada サブセットを用いて書かれている．

第2段階では，AIE コンパイラ用の原始プログラムをブート・コンパイラで翻訳し，中間段階の AIE コンパイラを作成した．このコンパイラに対し，引き続き Ada のフルセットとしての機能の完全性をテストした．この中間段階の AIE コンパイラは，最適化コンパイラではあるが，まだ自分自身は最適化されていない．すなわち，効率のよいコードを作り出すが，翻訳時間は遅い．

最終段階は，ブート・ストラップである．AIE コンパイラ用の原始プログラムを，ブート・コンパイラではなく，AIE コンパイラで翻訳し直した．その結果，得られたブート済みコンパイラは，機能的には中間段階の AIE コンパイラと同じであるが，いまや自分自身も最適化されたものになった．

4.2 1100 ACS コンパイラ

1100 ACS コンパイラは，Ada を用いて図5に示すように，UTS の下でクロス・コン

* UNIX は，Bell 研究所が開発したオペレーティング・システムで ATT の登録商標である．

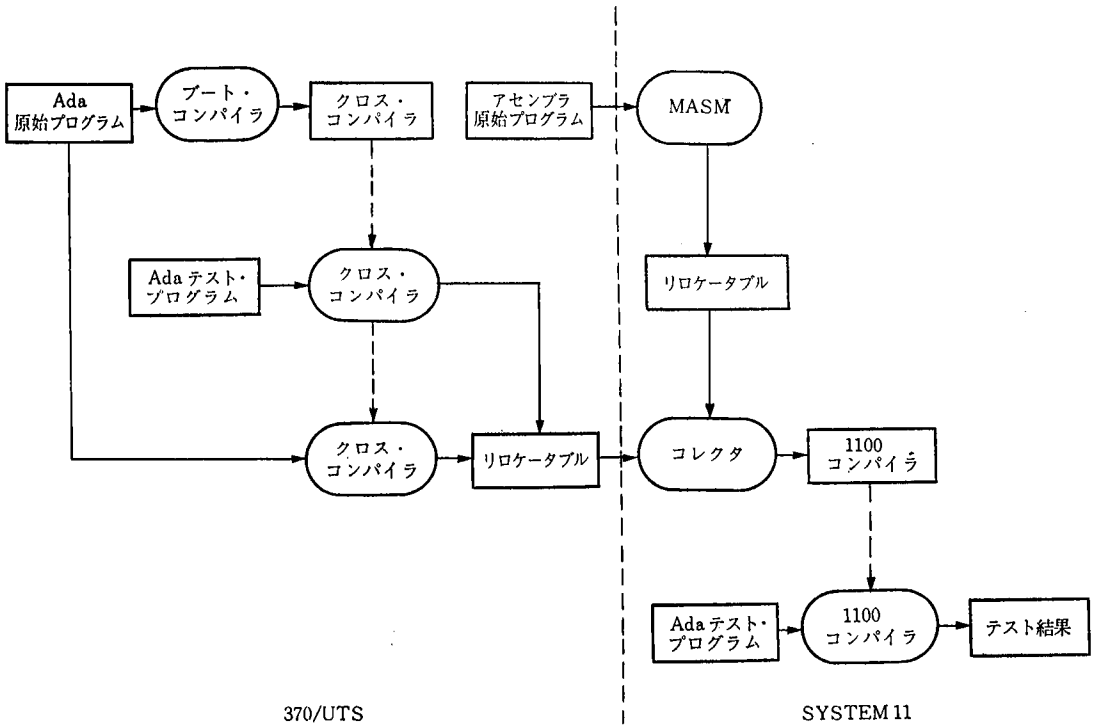


図 5 1100 ACS の開発

Fig. 5 1100 ACS development

パイラ技法によって開発された.

ACS コンパイラは, AIE コンパイラと同じ Ada サブセットで書かれている. UTS 上で動き, 1100 コードを作るクロス・コンパイラを作るのに, ブート・コンパイラを用いた. このコンパイラに対して UTS 上でリロケータブルを作り, CHAPARRAL シリーズ SYSTEM 11 上で連結編集をしてフルセット Ada としての完全性をテストした.

次に, クロス・コンパイラを用いて ACS コンパイラ用の原始プログラムを翻訳し, UNIVAC シリーズ 1100 上で動く Ada コンパイラを作り上げた. 最後に, 再び SYSTEM 11 上で ACS コンパイラのテストや効率改善の調整を行った.

4.3 コンパイラの移植性

このコンパイラは, 機種依存性を少なくし, 容易に変更できるように設計されている.

1) LEXSYN (字句解析と構文解析) では, 機種に依存する部分がまったくない.
2) CODEGEN (コード生成) では, 機種依存性を表だけに閉じ込めた表駆動方式を採用した. TABGEN (表生成) で, コードの骨組みから機種に依存した表を生成するようにした.
3) インタフェースは, 小さな Ada パッケージによって定義した.

実行時のモデルは, 一般性のあるアーキテクチャに対応できるように設計した. 実行時ライブラリのほとんどは, Ada で書いてある.

このコンパイラを別の機種に移植するためには, 次の作業が必要である.

1) 新しい表を作成する.
2) 機種に依存しているパッケージを書き直す.

3) 機種特有の実行時ライブラリ・ルーチンを書く.

4) 翻訳する.

（プロダクト企画部　真田 正二 訳）

参考文献

[1] "AIE-Ada Compiler Technical Description", IR-MA-335, Intermetrics, Inc., Cambridge, Massachusetts, July 1984.

[2] G. Booch, *Software Engineering in Ada*, Benjamin/Cummings Publishing Co., California, 1983.

[3] M. Burke and G. Fisher, "A Practical Method of Syntactic Error Diagnosis and Recovery", Proceedings of the SIGPLAN '82 Symposium on Compiler Construction, SIGPLAN Notices, Vol. 17, No. 6, June 1982.

[4] G. W. Cherry, "Parallel Programming in ANSI Standard Ada", Reston Publishing, Virginia, 1984.

[5] M. Dausmann, S. Drossopoulou, G. Persch and G. Winterstein, *Preliminary AIDA Reference Manual*, Universitat Karlsruhe, Inst. f. Informatik, Bericht Nr. 2/80, February 1980.

[6] A. N. Habermann and I. R. Nassi, *Efficient Implementation of Ada Tasks*, Computer Science Department, Carnegie-Mellon University, January 1980.

[7] M. Larsen, D. Ortmeyer, H. Turkle, and M. Gordon, "The Byron 1100 Programming Support Environment", Proceeding of 30th Anniversary USE Conference, November 1985.

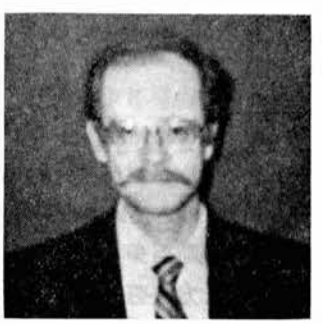
(邦訳) 真田正二訳, "Byron 1100プログラミング支援環境", 技報, 日本ユニバック(株), No. 11, 1986, pp. 42-54.

[8] *Reference Manual for the Ada Programming Language*, Version 1.1, ANSI Standard MIL-STD 1850A Document, U. S. Department of Defense, Ada Joint Program Office, September 1983.

執筆者紹介

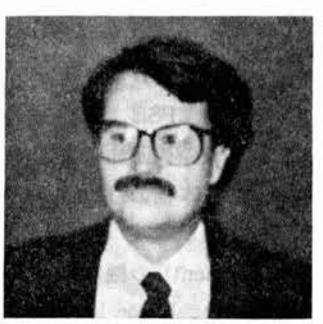
George Snyder

1972年に Massachusetts 工科大学にて物理学と建築学を専攻. 1980年に Boston 大学 Metropolitan 校にてコンピュータ科学の M. S. を取得. 1975年から1982年まで, Varian Associates 社製造機器部門に所属. 1982年以来, Intermetrics 社の Ada システム部門に従事.



David R. Wallace

Columbia 大学卒業. California 大学 Berkeley 校にて M. A., Tulane 大学にて Ph. D. を取得. Boston 大学を最後として, 多くの大学にて人工知能や言語理論のコースを始めとするコンピュータ科学を教授. Phoenix の GTE AE 研究所を経て, 現在, Intermetrics 社に勤務.



報告 Byron 1100 プログラミング支援環境

The Byron 1100 Programming Support Environment

M. Gordon, H. Turkle
M. Larsen, D. Ortmeyer

要 約 Byron 1100 PDL は，プロジェクトの全段階で統一された表現が可能になるように設計されたプログラム開発用語である．プログラム単位に関するすべての情報を一つのソース・ファイルに格納し，文書と実行可能プログラムの整合性を保っている．Byron 1100 のツール類には，Ada の構文や意味の検査を行うツールや，4種類の文書作成ツール，すなわち呼び出し木構造ジェネレータ，相互参照表ジェネレータ，データ辞書および使用者マニュアル・ジェネレータがある．また，書式テンプレートを設定できる汎用の文書ジェネレータも含まれる．本稿は，Byron1100 を新たに使用するユーザのためにその機能および使用方法を紹介するものである．

Abstract Byron 1100 is a PDL designed to provide a uniform means of expression which can be used through all phases of a project. Byron makes it possible to keep all information about a program unit in a single source file, and thus helps to insure that documentation is consistent with the executable code. Byron 1100's set of tools includes an Ada syntax and a semantics checker, and four document generator tools: a Call Tree generator, a Dependency Table generator, a Data Dictionary, and a User Manual generator. In addition, Byron 1100 includes a general purpose document generator, capable of accepting user-defined formatting templates. This paper is designed to provide the new user of Byron 1100 with an introduction to the product, its features, and its many uses.

1. はじめに

ソフトウェア工学の発展は，ソフトウェアの品質を改善しライフサイクル全般の費用を下げるためのツールや方法を生み出した．これらのツールは，多くの場合ある特定の開発活動やライフサイクル段階に焦点を合わせすぎていた．ソフトウェア工学の種々のツールと方法を一つのシステムに統合すれば，開発プロセス全体の標準化と支援が容易になる．Byron* 1100 プログラミング支援環境 (PSE: Programming Support Environment) は，Ada** をベースとしたプログラム開発言語 (PDL: Program Development Language) と統合プログラミング支援ツール群（たとえば，データ辞書，大域的相互参照など）からなる．PSE は，ソフトウェア開発ライフサイクルの種々のフェーズで生成された貴重な情報の体系的収集と検索を援助することにより，新しいプログラミング方法を支援する．

本稿の目的は，国防総省 (DOD) が定めた要求仕様ドキュメント「Stoneman」[1] に定義されている Ada プログラミング支援環境 (APSE: Ada Programming Support Environment) の機能の多くを取り入れた Byron 1100 PSE を読者に紹介することにある．ここでは，Byron 1100 PSE をいくつかの段階に分けて解説するが，最初に，PDL，Byron 1100 PDL アナライザおよびドキュメント・ジェネレータから構成される Byron 1100 PDL 中核パッケージから始める．中核パッケージは，主に設計および実現フェーズにおいて使わ

* Byron は Intermetrics 社の登録商標である．

** Ada はアメリカ合衆国政府 (Ada Joint Program Office) の登録商標である．

れるツール群からなり，Ada コンパイラと独立して使うこともできる．

Byron 1100 PDL は，Ada プログラミング言語を基礎にしている．このため，Byron の能力を完全に使いこなすには，Ada が単なる高水準の実現言語の一つとしての存在以上に支持されてきたのはなぜか，という理由を理解する必要がある．したがって，本稿では，最初に，Ada の設計言語としての特性のいくつかを解説し，その後で，ソフトウェア開発プロセスの全フェーズの支援において，Byron が Ada より優れている点を紹介する．ある意味では，Byron の価値を理解することは，Ada の価値を理解することでもある．しかし，Ada の価値を理解するには，まず Ada の開発の背景を知ることが必要である．

2. 背　景

過去 20 年間にわたり，ソフトウェア製品の生産者および消費者の間では，“ソフトウェア危機” として知られている状況に対する懸念が高まってきた．コンピュータ・ハードウェアが発達して，より多くのデータがより速い速度で処理できるようになるにつれ，解決できる問題の範囲も広がった．しかし，コンピュータ・ハードウェアの能力が急速に高度化しているため，Dijkstra[2] が指摘しているように，われわれの管理能力がそれに追いつかない．われわれの問題解決能力の制約は，信頼性が高く，費用効果に優れ，再使用・保守・移植が可能な効率のよいソフトウェア・ツールがないこと，および問題が複雑化の一途をたどっていることによっている．

高まるソフトウェア危機の一つの現れとしては，ある特定のニーズを満たすためのハードウェア費用とソフトウェア費用の関係が変化しつつあるということが良く知られている．Booch[3] は，国防総省の場合，コンピュータ関連費用の総額に占めるソフトウェアの割合は，1968 年には 20 パーセントであったが，現在は約 80 パーセントに増えていると報告している．危機を示すもう一つの症状として，ソフトウェア保守費用が開発費用を上回ることが頻繁にあるという認識がある．Boehm[4] は，コンピュータに関連する総費用の 40〜75 パーセントが保守費用であると報告した業界調査を引用している．もう一つの徴候は，ソフトウェア開発プロジェクトが絶えず「すでに発明されているのに，また再発明してしまう」事態を余儀なくされるためもあって，予定通りに完成しない場合が多いことである．

ソフトウェアによる解決方法がより大規模で複雑になるに伴って，ソフトウェア技術者が次のような課題に直面していることが明らかになってきた．

1) チーム編成の開発環境において，プログラマの生産性を向上させるソフトウェア・ツールの開発……この課題は，高度な技術を持ったプログラマの不足が顕著な環境では非常に重要である
2) ソフトウェアの保守性，再使用性および移植性を強化するソフトウェア・ツールの開発
3) 頑丈な製品を生産できるソフトウェア・ツールの開発

1970 年代初めに，国防総省は組込み型コンピュータ・アプリケーションの開発に使うソフトウェア・ツールの不備を認識・矯正する措置に着手した．Booch の報告にあるように，これらの巨大なアプリケーションは，耐故障性，寿命，効率，変更容易性および保守性という困難な要件を満足させなければならなかった．当時使われていたソフトウェア・ツールは，「質の高いソフトウェアを得ることなど到底できない」[5]ような代物であった．Booch は，国防総省のソフトウェア危機の要因となっている問題として，プログラミング

言語の多様化（その多くが組込み型アプリケーション・プログラミングに適合しない），新しいプログラミング方法に適さない言語（たとえば，COBOL，FORTRAN および多数のアセンブリ言語）の使用，有効なソフトウェア環境の不足を見出した．これに対処するため，国防総省は，1975年に，高位言語ワーキング・グループ（HOLWG：High Order Language Working Group）を設立し，共通プログラミング環境の利点を検討した．この中から，Ada と Ada プログラミング支援環境（APSE）が誕生した．

3. プログラム設計言語としての Ada

設計の優れたプログラムは，開発費用と保守費用が安価で，変更および移植が容易で信頼性が高いものが多い．Ada は，実現言語の一つというだけではなく，プログラム設計言語（PDL：Program Design Language）であり，かつ種々の設計方法（たとえば，段階的洗練，算体主導設計，構造的設計）を支援できるソフトウェア・ツールである．

3.1 従来の PDL と Ada の比較

従来から，PDL は疑似コード（すなわち，いくつかの構造文に加えて自然な英語の文章を書くことのできる一種の言語）の概念に基づいて作られてきた．これらの文章で動作を記述して，実現は後で行う．開発後の洗練された疑似コードは，最後にプログラマが翻訳するかトランスレータで処理して，なんらかの実現言語に変える．

疑似コードは，意図した実現言語が構造文を支援しない場合（たとえば，FORTRAN Ⅳ）にとくに有用である．このような場合は，実現言語を無視して構造的プログラミングの利点を享受できる．

一方，Ada も本来構造的プログラミングを支援している．また Ada による設計は，非常に読みやすい形式で表現できるうえ，決定を後に引き延ばすこともできる．たとえばミサイルの発射を制御するコードの断片は以下のように始まる．

```
launch_missile ; -- until enemy airplane is destroyed.
```

これは，後で以下のように洗練することもできる．

```
if not airplane=friendly then
  loop -- until no more missiles
    -- determine flight path of airplane
    launch_missile ; -- at airplane
    exit when target_destroyed ;
  end loop ;
end if ;
```

この設計で3回目の見直しを行うと，次のように洗練される．

```
if not friendly_aircraft (airplane) then
  while number_of_missiles> =1
  loop -- until no more missiles
    -- determine flight path of airplane
    flight_path :=acquire_tracking_info (airplane) ;
    launch_missile (flight_path) ; -- at airplane
    exit when target_destroyed (airplane) ;
  end loop ;
end if ;
```

Adaが読みやすく，かつ柔軟性があるため決定を引き延ばせることは明らかであろう．さらに，上記の不完全なコード断片の各々は正当な Ada 構文であるため，コンパイラは進行中に設計のいくつかの側面について検査を援助することができる．たとえば，コンパイラは launch_missile と target_destroyed という副プログラムが定義済みかどうか，このセクションのコードによって使用できるかどうか，ということを知らせることができる．

疑似コードを使う PDL の大きな利点は，自然な英語の構文を使って自由に表現できることである．しかし，これは大きな弱点でもある．自然な英語を使うと，曖昧な表現を招き検査の自動化ができなくなる．また，疑似コードは最終的には，実現言語に翻訳しなければならない．そうなると，二つのファイルを保持する必要がある．実現開始後に設計変更が余儀なくされることは不可避であり，このため一部の変更は疑似コード・ファイルにまで及ばないことがある．そこで，疑似コード・ファイルは，実現開始時のせいぜい歴史的な設計記録でしかなくなる．最悪の場合には，設計文書のように見えるが微妙に異なる，誤解を招きやすい紛らわしいものになるおそれがある．

一方，Ada は初期設計と最終実現の両方に使用できる．プログラムが進化しても一つのファイルしか保持しなくてよく，最終設計はコード自体から明らかになる．設計の進化を示す歴史的な記録は，時間の経過の中のさまざまな時点におけるスナップ・ショットによって得ることができる．

3.2 Ada の設計言語としての機能

本稿のような短い論文では，Ada の設計言語としての機能の詳細，ましてや実現言語としての Ada の能力の概要を説明することは不可能である．簡単にいえば，Ada はデータ抽象化，情報隠蔽およびパッケージングの機能を用いて設計フェーズにおける高水準の概念化を奨励する．Ada は，汎用体 (Generics) および例外処理機能を使うことによって再使用可能で，耐故障性のあるユーザ・フレンドリなソフトウェアの作成を援助する．設計および実現言語として Ada は，ソフトウェア危機の解決策を与えるという面で，他の言語よりはるかに進んでいる．

Ada でモジュール性，データ抽象化および情報隠蔽の設計原理を支援する方法が，パッケージという概念である．Ada のパッケージとは，関連する手続き，関数およびデータの集合を言い，これは仕様と本体の二つの部分からなる．パッケージの本体は実際の実現を含んでいるが，仕様はパッケージと残りのプログラムとの間のインタフェースを提供する．すなわち，仕様はパッケージ外からアクセスできるパッケージの部分，それへのアクセス方法およびパッケージ内で使用される外部データと副プログラムを記述する．もっと重要なことは，パッケージ内でしかアクセスできない宣言を含む密閉部 (Private Part) を仕様に入れることができることであろう．本質的には，密閉部はパッケージ外からは使用できないパッケージ内の部分を示す．このようにして，パッケージと残りのプログラムとの分離が厳密に定義され，かつ強制される．

設計中は，パッケージ本体とパッケージ仕様の分離がきわめて貴重である．これによって，設計者は操作されるデータ，操作を行う副プログラム，両者の間の依存関係の決定に集中できる．実現の詳細に関する決定は，本体をコード化するまで引き延ばせる．ただし，Ada は，外界との副プログラム・インタフェースを例外として，副プログラムの厳密な機能を指定する機構を提供しない．また，機能については注釈の中に非公式にしか記述できない．一方，Byron 1100 PDL では，これを形式的に表現できる．

密閉型と汎用体は Ada のパッケージの二つの特徴であり，ソフトウェア開発プロジェ

クトの設計フェーズにおいてとくに重要な価値を持つ．密閉型は，定義しているパッケージ外ではその名前しかわからない型をいう．そして，この型の実際の表現は，パッケージ内でしか見えない．このため，情報隠蔽とデータ抽象化が他の言語とまったく異なる方法で直接行える．汎用体は，パッケージの再使用性を高める方法である．たとえば，スタックと木構造は多種の型式のデータを格納することができる共通のデータ構造である．しかし，格納されるデータの型とは関係なく，スタックの実現は本質的には同じである．汎用体パッケージの具体化（使用するためにプログラムに含める）では，すべての演算で使う型をパラメタとして与えることができる．そこで，一つのスタック・パッケージを書いておくと，必要なスタックの型ごとに具体化できる．

パッケージの構造を用い，Adaでは，モジュール性，データ抽象化および情報隠蔽が自然にプログラムできる．他の言語でもこのような原理に従うことはできるが，Adaはこれらの原理を厳密かつ形式的な方法で支援するため，他の構文や意味を考慮しながら原理がチェックできる．このような原理により，モジュールの目的がはっきり定義でき，外的ニーズと相互関係を全般的に少なく明確に指定できるため，Adaパッケージを再使用できることが多くなる．さらに，データ抽象化，情報隠蔽およびモジュール性の各原理に従えば，パッケージの保守および改良の仕事が単純化される．パッケージで使えるデータは厳密に指定されるため，プログラマが保守およびデバッグ時に考慮しなければならない要素は少なくなる．モジュール性があるためにモジュールを他と分離して検査できるので，パッケージは検査しやすく，信頼性も高くなる．

強い型付けの利点は，Pascalの経験からも良く知られている．Adaは，この伝統に従いPascalの型をさまざまな方法によって強化してきた．派生型は，既存の型（たとえば，整数）を使って基本的には同じであるような新しい型を作り出す方法を提供し，新しい型が偶然に古い型と混じらないことを保障するものである．Adaの型のその他の興味深い機能としては，変数の初期設定（集成式を使って），動的で無制約な配列および固定小数点型がある．このような機能は，設計不備やバグの迅速かつ効率的な発見に貢献できる．

耐故障性があるユーザ・フレンドリなシステムを作るためには，異常条件を丁寧に扱う能力を持つことが必要となる．異常条件としては，実行時システム・エラー（たとえば，ゼロ除算エラー）または特殊処理（たとえば，固定長スタックにおけるスタックあふれ）を必要とする条件が考えられる．Adaは，この問題を解決するための例外機能を提供する．この機能は，誤りに備えるものとして非常に有意義であるため，言語機能として望ましいものであるが，役に立つ設計機能でもある．副プログラムまたはパッケージの機能を指定してしまえば起こりうる問題を知ることができるが，講じべき動作は必ずしもわからない．Adaでは，設計者が将来考慮しなければならなくなると思われる問題に対し，設計段階で例外を定義することもできる．ただし，実際の修復動作の決定は実現時まで引き延ばすことができる．

4. Byron 1100 PDL 中核パッケージ

Byron 1100 PDL 中核パッケージは，Adaをベースとしたプログラム開発言語（Byron 1100 PDL）を中心に構成されている．Byron 1100 PDLは，巨大ソフトウェア・システムの製作時に作成される情報を形式的かつ効率的に表現することにより，Adaの設計言語としての能力を高めることを目的に設計された．Byronのパッケージには，この情報を構造的なデータベース（プログラム・ライブラリ）に格納し，必要に応じてこのデータベース

からの情報を分析・抽出するツールが含まれる．Byron は，またフェーズ・チェック能力を使って情報収集を“強制”することもできる．

Byron 1100 PDL は Ada に基づいており，正しい Ada プログラムは，すべて正しい Byron 言語であり，その逆もいえる．これまで説明してきたとおり，Ada は設計を援助・改善する多くの機能をもっている．しかし Ada では不必要，あるいは表現不能であるが，明確に指示しなければならない情報があることも知られている[6][7]．この情報は，ほとんどが本質的に意味的なものであり，データ項目や副プログラムの使用法あるいは目的に関するものである．次の Ada の副プログラム仕様を考えてみよう．

function CopyLinkedList (List : in ListPtr) return ListPtr ;

翻訳するためにはこれで十分であるが，この関数を使うためには各リスト要素の物理的コピーが作成されているかどうか，あるいは単なるリストへのポインタのコピーかどうかなどの詳細を知る必要がある．Byron は，このような情報を組織的に格納・検索できるように設計されている．

Byron 1100 PDL は，二つの重要な点で Ada 言語と異なる．まず第1に，Ada の宣言に関する情報を宣言自体と関連付ける機構を持っている点である．この情報は，キーワードを使って任意の望ましい形式に構造化できる．2番目は，Byron 1100 PDL は構造化されており，初期のプログラム開発段階に特有な不完全で矛盾のあるコードからでもツールが有用な出力を作れるようになっている．Byron 1100 では，実行可能コード（正しい Ada コード）を作るように翻訳できる点が，従来のプログラム設計言語と異なっている．このため，Byron 1100 は，仕様から保守までのソフトウェア開発プロセスのすべてのフェーズで利用できる．

Byron 1100 PDL の使用には，四つの大きな利点がある．第1に，プロジェクトの全ライフを通じて使える統一された表現手段が用意される．第2に，多くのソフトウェア開発活動が自動的に支援されるようなプログラムを書くことができる．第3に，一つのプログラム単位に関するすべての情報を一つのソース・ファイルに保存できるようになるため，文書を実行可能コードと常に整合させるのに役立つ．最後に，一つのプロジェクトに関するすべての情報を一つのデータベース（プログラム・ライブラリ）に保存しておけば，必要なときにいつでも最新の文書を生成することができる．

4.1 Byron 言語

Ada の注釈は“--”で始まり，行の最後まで続く．Ada の構文規則と互換性を保つため，すべての Byron 言語の文は接頭辞“--|”で始まっている．こうすれば，同じソース・ファイルを Byron アナライザでも Ada コンパイラでも処理できる．

Byron の文章をつくるときは，すべて Byron 接頭辞“--|”を使う．Byron の注釈は識別子を宣言した直後に使い，記述文とその識別子を関連させる効果を持つ．たとえば，

```
FILE_ERROR : --|This exception is raised whenever     （この例外は，なんらかの理
        exception ; --|a file operation fails for any    由によってファイル操作が失
                --|reason.                               敗したときに発生する．）
```

ここでは，識別子 FILE_ERROR は，それに関連する3行の記述を持っている．下例のように Byron のマーカー（--|）の後ろにキーワードを付ければ，注釈に含まれている情報を識別できる．

```
Procedure Read-Line ; --|This line is the synopsis    （この行は，Read-Line の大
                --|of Read Line.                        意である．）
```

--|Raises : Read-Beyond-EOF	(Read-Beyond-EOF を発生)
--|Effects	(効果)
--|Reads a line from the default input file	(デフォルト入力ファイルから
--|and returns the text. If an attempt is made	1行読んでテキストを返す．ファ
--|to read past the end of file, the exception	イルの限界を越えて読もうとすると，例外 Read_beyond_
--|Rread-Beyond-EOF will be raised.	EOF が発生する．)

上例の一番最初の注釈は，手続き Read_Line の大意である．この Byron のテキストは2番目の Byron 注釈で終了し，Raise 指示語を開始する．この指示語は，次の Byron 注釈が別の指示語 Effects であるため，1行だけからなる．残りのすべての注釈は，この指示語の一部である．

Byron では11種類のキーワードが定義されている．そのうち1種は型と算体，残りの10種はプログラム単位（パッケージ，関数，手続きおよびタスク）に使われる．後述するが，これ以外のキーワードも定義できる．

4.2 フェーズ・チェック

Byron アナライザは，ソフトウェア開発プロセスを六つのフェーズに分けて認識する．

1) プロトタイプ
2) 仕様
3) 設計
4) 実現
5) 検査
6) 保守

各々のフェーズに合わせて異なるキーワードが必要となる．各フェーズでは，それ以前のフェーズで必要であったすべてのキーワードが必要となる．プロトタイプ・フェーズでは，すべてのキーワードはオプションである．

Byron アナライザは，プログラム・ライブラリ中のプログラム単位ごとに，最後に試みたフェーズと最後に完成したフェーズの両方を記録する．キーワード“Completeness (完全性)”について翻訳単位をチェックしたい場合は，アナライザを呼び出すときに試みたフェーズ（例，実現フェーズ）に入らなければならない．選んだフェーズが必要とするキーワードが存在しないと，アナライザが警告メッセージを出す．必要なすべてのキーワードが存在していれば，アナライザは選んだフェーズが完了したことを記録する．このように“Completeness”のチェックを行うことにより，Byron 1100 PDL では効果的にプロジェクトの進行を監視できる．

4.3 使用者定義キーワード

Byron は，Byron 指示語を追加定義できる機能を備えている．使用者定義キーワードは，作成時にプログラム・ライブラリと関連付けられる．これは，同じ Byron キーワード群を，ライブラリに集められたすべてのプログラム単位で使用できることを意味する．

新しいキーワードを定義するには，三つの情報項目，“キーワードの名前”，“キーワードを使用しなければならないフェーズ（初期フェーズではキーワードはオプション)”および“キーワードが妥当な文脈”を指定しなければならない．キーワード名は Ada の識別子に関する規則に合致し，32文字より短くなくてはならない．キーワードは一度しか定義できない．

必要なフェーズが“オプション”である場合は，キーワードもオプションとなる．Byron アナライザを呼び出すときに，使用者は開発のフェーズを指定する．そうすると，アナライザが，そのフェーズまたは以前のフェーズで必要とされるすべてのキーワードが存在しているかどうかを検証する．

使用者定義キーワードを使うことによって，与えられた設計方法を確実に支援し，その利用を強制するように Byron を調整することができる．また，意味的情報の追加を義務付け，追加すべき情報の種類を指定する方法も提供される．

4.4 既定義の Byron 1100 キーワード

Algorithm ……このキーワードの後ろには，副プログラムの実現またはパッケージ本体の初期化に使うアルゴリズムを記述したテキストが続く．この記述は，アルゴリズムに関する文献への参照，副プログラムの仕事のやり方についての英文の説明，または必要な結果の計算に使われるアルゴリズムを疑似コードで表現したものである．

Effects……このキーワードの後ろには，副プログラムまたはパッケージの初期化部分が実行する機能を説明したテキストが続く．このテキストは，副プログラムが実行する機能，例外が発生する条件（後述する Raises 指示語は，単に発生しうる例外の名前をリスト化するだけである），計算結果，ファイルまたは装置への入出力，大域的な値の変更および他の副作用，そして関数については返される結果を記述する．

Errors ……このキーワードは，副プログラムまたはパッケージの初期化部分によって，ユーザの端末またはログ・ファイルのどちらかに出されるエラー・メッセージを記述したテキストが後ろに続く．

Invariants ……このキーワードの後ろに続くテキストは，なんらかのデータ抽象化を実現するのに使われる表現の不変量を文書化したものでなければならない．このキーワードは，たとえば，型，算体または仮パラメータの宣言後に使われる．

Modifies ……このキーワードの後ろには，副プログラムの実行（またはパッケージ本体の初期化）時に変更される非局所的変数のリストが続く．変更は以下の方法による．

・パラメタを通す
・アクセス値を通す
・大域的算体

この構文は，プログラム単位の副作用という使用者の興味を引く情報を記述するために有用である．

N/A……このキーワードは，後ろに Byron キーワードのリストが続く．このリストは Byron アナライザに対し，並べたキーワードがこのプログラム単位に適用不能であるため故意に省略されていることを示す．このリストには任意のキーワードが入る．

Notes ……このキーワードの後ろには，次のような他のキーワードに適用できない情報を知らせるテキストが続く．

・設計者および実現者など
・問題があるときの連絡先

・ホストと目標の依存関係
・組み込まれている限界
・状況
・バージョン識別子

Overview ……このキーワードは，後ろにプログラム単位の概略説明が続く．Overviewは通常パッケージ仕様のなかで，たとえば新しいユーザのためのパッケージ紹介などに使われる．これを使うことによって，読者はそのパッケージが自分の探している機能を持っているかどうかを判断でき，大まかな使用コストおよびパッケージ内容の使い方の概略がわかる．パッケージ内の各副プログラムに関するEffectsの記述は，各副プログラムの動作を説明する．Overviewは，それらがどのような形で協力し合うかを示す．

Raises ……この指示語の後ろには，プログラム単位で発生する可能性がある（すなわち，発生はされるが処理はされない）例外のリストが続く．このリストには，副プログラムが直接発生させる例外だけしか入っていない．副プログラムが例外を発生させることのできる手続きPを呼ぶと，各々のPの呼び出し者ではなくPによって文書化されることになる．

Requires ……このキーワードの後ろには，プログラム単位がユーザに対して課した制約の記述が続く．たとえば，

・呼び出しの順序付け
・他のプログラム単位による初期化と終了処理
・大域的データに関する仮定
・同期化要件
・実行時環境に関する仮定
・手続き，関数またはエントリについてのパラメタ値の制約

Tuning ……このキーワードの後ろには効率の問題を記述したテキストが続く．この問題としては，効率が環境の性質，パラメタの値または任意の重要な定数宣言の値の変化に合わせてどのように変動するかが含まれる．

4.5 Byron 1100 PDL 中核パッケージ・ツール

Byron 1100 PDL 中核パッケージには六つのツール，すなわち Byron アナライザ，テンプレート方式文書ジェネレータ，使用者マニュアル・ジェネレータ，呼び出し木構造ジェネレータ，データ辞書ジェネレータおよび翻訳順序ジェネレータが含まれている．最初にあげた Byron アナライザは，Byron ファイルのテキストをプログラム・ライブラリに挿入するのに使われる．このアナライザは Ada コンパイラのフロント・エンドであり，Byron 用の注釈が認識できるように拡張されている．このため，Ada コンパイラの完全な構文および意味的チェックができると共に，指定した Byron の構文のチェックもできる．

また，テンプレート方式文書ジェネレータを使えば，Byron 用の注釈のなかに格納されている情報と共に，Ada のプログラムに関連する構文および意味解析上の情報の多くを引き出すことができる．これは高水準言語のインタプリタであり，これを使うことによってユーザはプログラム・ライブラリ内の情報にアクセスでき，その情報を指定した書式で出力することができる．この文書ジェネレータは，これまでも MIL-STD-483（システム，機器，資材，およびコンピュータ・プログラムの構成管理実務）(C 5) 生成テンプレートのような複雑なテンプレートを書くのに使われてきた．

中核パッケージの他のツールの一つであるデータ辞書ジェネレータは，Adaプログラムで宣言される型，変数，手続き，関数等のリストを作成する．翻訳順序ジェネレータは，Adaプログラムを構成するパッケージの翻訳順序を決定するのを助ける．呼び出し木構造ジェネレータを使って生成した文書は，どのルーチンが相互に呼び合っているかを知らせる．使用者マニュアルの文書は，Adaプログラムのなかの副プログラムの機能および使い方を記述する．

5. Byron 1100 PDL プログラミング支援環境

ツール群は，Byron 1100 プログラミング支援環境の本質的部分ではあるが，ツールの選択とツール間の関係こそが真に統合的なシステムの動作の詳細を特徴付けるものである．この章では，対象を広げて Byron ツール群全部を APSE との関係のなかで検討したい．完全な Byron 1100 プログラミング支援環境は，多数の独特のツールを包含している．たとえば，MIL 標準 C 5 （コンピュータ・プログラムのプロダクト仕様）文書ジェネレータなどのツールは，Byron 1100 自体の開発を助けるべく作られた．もう一つのツールである USERMAN は，プログラム・ライブラリ内の手続き，または外部パッケージの集合に対するインタフェースを記述する文書を作成する．このツールは Ada パッケージの再使用を奨励し，一つの開発フェーズから別の開発フェーズへの移行を助けるべく作成された．USERMAN の文書は，設計と実現が並行して行われるプロジェクトに対して情報を提供するのにとくに役立つ．Byron 1100 システムではこれらのツール群を，有用性が実証され，ライフサイクルのフェーズ間の円滑な移行を促進する能力を持っているという理由から選んでいる．

一方，使用者にとっては，ツール間の円滑な移行も同様に重要である．ユーザ・フレンドリな環境ではツールの部品の動作の詳細を使用者に伏せているが，これは使用者のツール操作能力を制限し，各ツールの柔軟性を限定するおそれがある．Byron 1100 システムでは柔軟性と固定した使用者インタフェースとをある程度妥協させることによって，ユーザ・フレンドリなものを実現した．Ada ソース・コードを処理するがプログラム・ライブラリに依存しないツール群は，一定のスタイルの使用者インタフェースを持っている．一方，プログラム・ライブラリにアクセスするツール群は，基礎にあるライブラリ管理システムに依存するが，これらも前と異なるものの一定のスタイルのユーザ・インタフェースを持つ．一般的に，使用者が各カテゴリ内の一つのツールの操作方法を知っていれば，他のものについても操作できる．

プログラミング支援環境は，各フェーズに対するツールを含んでいなければならず，これらは，二つの基本サブシステムに分けられる．すなわち，特定の方法論によらずプログラミングの作業を支援するツールと，設計および方法論を支援するツールである．最初のものについては，Byron 1100 は静的な分析のための種々の技術的プログラミング・ツールを提供する．このなかには，Ada コンパイラ，リンカ，再翻訳マネージャ，大域的相互参照，ソース・フォーマッタ，プログラム・リスタなどが含まれる．設計ならびに方法論のツールには前に述べた Ada PDL や，ソースや文書の構成管理，設計要求の追跡パッケージ，データ辞書システムなどが含まれる．

Byron 1100 支援システムの基本設計は，きわめて柔軟性が高く，広いアプリケーション分野に適合する．また，開放されたシステムであるため，個々の使用者が自分のツールをシステムに含めたり，既存のツールを構築ブロックとして使うことができる．たとえ

ば，プログラム・ライブラリ・アクセス・パッケージ (PLAP) は，ライブラリへのウインドゥを提供する．使用者は，自分のプログラムに関する情報をいつでも引き出すことができ，ライブラリの内部構造をまったく配慮する必要がない．使用者は，PLAP を使って自分のプログラム・ライブラリについて質問し，必要な文書を出力するような Ada プログラムを書くことができる．このパッケージを使うことにより，階層的チャートを書くツールやプログラム相互結合マトリックス・ジェネレータなどの複雑なツールを構築することができる．

各開発フェーズについての最新の情報が得られる能力は，ソフトウェア支援環境のもう一つの重要な特性であり，これは他の単一目的のツールとの大きな違いである．環境というのは，情報を獲得・普及できるツールを持った情報ユーティリティでなければならない．APSE モデルは，環境の中心にソフトウェアに関するすべての情報を含んだデータベースを必要とする．そして，このデータベースの周囲には，ツールとツールの部品および使用者インタフェース・コンポーネントからの要請に応じて，データベースをアクセスできる情報管理システムがある．

Byron 1100 システムは，Ada プログラム・ライブラリに重点を置いた包括的なデータベースを供給する．ライフサイクルの全フェーズに関連する情報は，このデータベースのなかに DIANA として知られる中間形式で格納される．プログラム・ライブラリには，要求仕様と設計仕様の表現，原始コード，目的コード，文書および状況の履歴を入れることができる．問題に対する違った解決法や種々の開発段階を表す，さまざまなバージョンを表現することもできる．プログラム・ライブラリ，またはライブラリの部品を同時に翻訳作業中の数人の開発者が共有できることも，もう一つの Byron 1100 の特徴である．

Ada のプログラムは，複雑かつ多岐にわたる相互関係を持つ多数の部品からなる．Byron 1100 は，プログラム・ライブラリの保守に必要な事務的管理業務を簡素化する．ライブラリ内の情報は，コレクションとカタログにグループ分けされるが，この二つは相互に連結されている．最高レベルでは，コレクションはデータ・ファイルおよびカタログを始めとするすべてのデータベースの内容を保守する．カタログは最低レベルでの抽象化であり，単独の Ada ライブラリ単位を表すこともできるが，一般にはプログラム全体の大きな部分を表現するのに使われる．カタログを使うことによって，使用者は自分のプログラムを作業単位，機能単位，プログラムの各版などに分割できる．

Byron 1100 プログラム・ライブラリは，複数の同時使用者，柔軟なプログラム構成および独立したプログラム・バージョンを支援すべく設計されたものである．またデータベースを管理し，ライブラリ内のライブラリ単位の内容や状況を表示するための種々のライブラリ支援ツールも用意されている．プログラム・ライブラリにアクセスするには，常にプログラム・ライブラリ・インタフェース (PLIF) を使う．たとえば，結合しようとする Ada の目的モジュールの集合を見つけ出すのにリンカは PLIF を使う．その他にも，前述のライブラリ・アクセス・パッケージ PLAP によって使われる．

6. Byron 1100 PSE の応用分野

Byron 1100 は全ライフサイクルを通じて，プロジェクトの組織化，計画，監視および見直しなどについて管理者を助けるための方法とツールを提供する．Byron 1100 PDL のパッケージの機能としては，たとえばサブシステムを体系的に分割して作業分解構造 (Work Breakdown Structure) による開発を援助することができる．パッケージの仕様部

で定義されている設計単位インタフェースは，クリティカル・パス分析に有用なタスク間の依存関係の確定に役立つ．Byron 1100 PDL のフェーズ・チェックは，プロジェクトの重要なマイルストーンの設定と監視を助けることもできる．Byron 1100 PDL の使用者定義キーワードを使えば，実現者や設計者の名前，プロジェクト進捗情況，プロジェクトの統計データなどの重要なプロジェクト情報を保持できる．

典型的な例では，プロジェクトの要求フェーズでは最初に解決すべき問題を決定し，その問題を解決するシステム設計を定義し，ハードウェアとソフトウェアに設計要求を割り当てる．このフェーズの活動により，MIL 標準 B5 （コンピュータ・プログラムの開発仕様）などの文書ができる．設計要求追跡ツールは，要求を設計要素およびモジュールに関連付ける機構を提供する．これは，後期のフェーズでとくに有用であり，変更の影響を迅速に追跡できる．

仕様および設計フェーズでは，話題の焦点は機能から分解に移行する．Byron 1100 PDL と Ada のラピッド，プロトタイプ機能を使えば，プログラム・アーキテクチャとデータ構造が作成できる．ここでの活動のほとんどは，文書化される．テンプレート方式文書ジェネレータは，MIL 標準 C5 仕様および種々の文書の作成に役立つ．このツールを使えば，パッケージや副プログラムの最新情報の要約が容易に作成できる．

データ構造の開発を助けるための会話型データ辞書も利用できるようになろう．このフェーズの早い時期に構成管理システムを使って，構成の制御が開始される．翻訳順序のツールはシステムの依存関係について報告し，またこれを使うとプログラムをコンパイラにかける順序を確立できる．Ada による仕様をコンパイラで解析すれば，初期のエラーを検出してプログラム・ライブラリに入れることができる．

プログラム設計言語として Ada を使えば，設計フェーズから実現フェーズへ円滑に移ることができる．コーディングでは，相互参照や翻訳リスティングなどを頻繁にとることによって援助される．McCabe の計測ジェネレータを使えば，非常に複雑なコード領域を識別することにより，発生しうるエラー数を推定できる．Ada のコンパイラは，任意のコンポーネントの実現が完了する時期を決定でき，自動的に目的コードを生成し，プログラム・ライブラリに格納する．再翻訳のサイクルは，再翻訳マネジャが援助する．最後に，リンカ (PBUILD) が検証の用意ができた実行可能プログラムを生成する．

システムが作動を開始すると，検証とチューニングにプロジェクトの焦点が絞られる．検査では，Byron 1100 の追跡と報告の機能を用いて追跡を行う．このシステムは，現時点ではデバッグとチューニングを完全に支援するツールは利用できない．しかし，自動パス・アナライザ，自己計測と解析のツール，および Ada 性能アナライザとデバッガは，すべて開発中である．既存の Byron 1100 システムに，これらいくつかの開発中のツールが追加されれば，国防総省が描いた完全な APSE の要件を満たすことになろう．

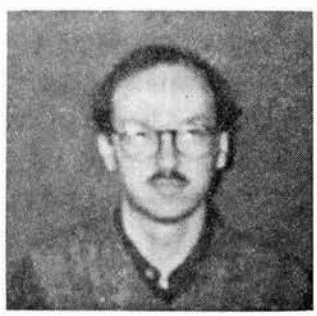
7. おわりに

本稿は，読者に Byron 1100 プログラミング支援環境，すなわちソフトウェア開発ライフサイクル時に生成される情報の体系的な収集，格納および検索を援助すべく設計されたプログラミング支援ツール群を紹介した．この環境は，進化し続ける開発方法を支援できる新しい使用者定義ツールの開発が可能となるように設計されている．Byron 1100 PSE は最新技術を反映した最初の商用 Ada プログラミング支援環境の一つである．

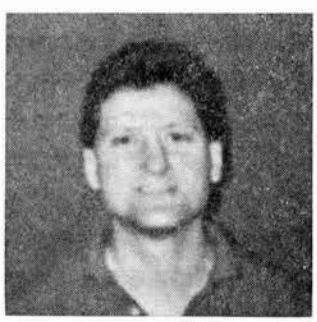
（プロダクト企画部　真田 正二 訳）

参考文献 [1] US Department of Defense, *Requirements for Ada Programming Support Environments, 'STONEMAN'*, US Department of Defense, Washington, 1980.
[2] E. W. Dijkstra, "The Humble Programmer", Communications of the ACM, Vol. 15, No. 10, October 1972, pp. 859-866.
[3] G. Booch, *Software Engineering with Ada*, Benjamin/Cummings Inc., Menlo Park, 1983.
[4] B. W. Boehm, *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, 1981.
[5] Booch, op. cit.
[6] F. W. von Henke, D. Luckham, B. Krieg-Brueckner and O. Owe, "Semantic Specification of Ada Packages", Ada in Use: Proceedings of the Ada International Conference, Cambridge University Press, Cambridge, 1985, pp. 185-196.
[7] B. Liskov, *Modular Program Construction Using Abstractions*, MIT Computation Structures Group Memo 184, September 1979.

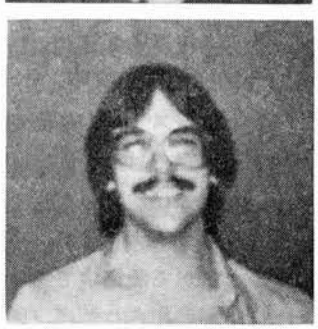
執筆者紹介 Michael Gordon
1978年に Massachusetts 工科大学にて電気工学とコンピュータ科学を専攻. 1982年より Intermetrics 社において, Ada のソフトウェア・ツール開発に従事. IEEE の専門委員会 P 990「Ada の PDL 用法」の委員である.



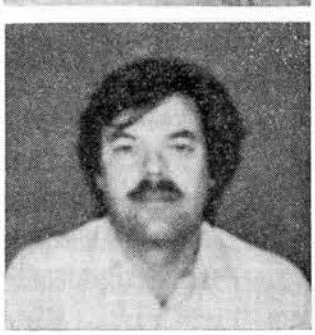
Haynes Turkle
1978年に Massachusetts 大学にて化学を専攻. 1984年より, Intermetrics 社において, Byron の開発と保守に従事. 他に電話交換システムの開発に用いるソフトウェア工学の自動化ツールの設計の経験がある.



Matt Larsen
1983年に Wisconsin 大学 Milwaukee 校にて数学で B. S を, 1985年に同大学 Madison 校にてコンピュータ科学で M. S. を取得後, Intermetrics 社に入社.



David Ortmeyer
1972年に Stanford 大学にて経済学で B. A. を, 1976年に Wisconsin 大学 Milwaukee 校にて経済学で M. S. を取得. 1980年に同校で経済学の Ph. D. を取得. 1978年から 1984年まで Tufts 大学の助教授を務め, 1984年以来, Intermetrics 社でプロダクト・マネージャを務めている.



報告 JSD 仕様の実行系の試作

An Experiment on Direct Execution of a JSD Specification

加 藤 潤 三

要 約 JSD (Jackson System Development) は新しいシステム開発方法である. M. A. Jackson[2] によれば, JSD の仕様は原理的には実行可能である. JSD の仕様を実行する道具があれば, われわれはそれをプロトタイピングの道具として使える. 本稿は, JSD の仕様を実行する道具の開発に関する実験報告である. JSD の仕様の実行にはいくつかの工夫が必要であった. それらは 1) 通信プリミティブ, 2) システム仕様図 (SSD) と行動一覧表 (Action List) を表現する構文の導入とに2分できる.

ここでは簡単な例を使い, JSD の仕様の実行のために導入した言語 J を紹介し, JSD の仕様の実行方法と実行結果を示す.

Abstract An approach to a direct execution of JSD specification is described. According to M. A. Jackson[2], a JSD specification is directly executable, in principle.

If we have a tool for executing a JSD specification, we can use it as a prototyping tool. This paper introduces a language, named "J" which is a modified JSD specification language. The points of the modifications are:

1) elaboration of communication primitives,
2) introduction of new declarations which represent a System Specification Diagram (SSD) and Action List.

This paper represents the main parts of "J", and a very simple example of JSD specification in "J" with results of its execution.

1. はじめに

近年, 伝統的なシステム開発のライフサイクルの概念は有害ではないかという議論[3]があり, 従来のライフサイクルとは異なる開発方法が注目されている.

その一つとしてラピッド・プロトタイプによる方法があり, これに関しては多くの提案が存在する. 筆者はラピッド・プロトタイピングの一案として, システム仕様のシミュレーション・モデルを作りその振る舞いを観察するということを考え, JSD に基づいた仕様をシミュレートする処理系を試作した. 本稿はその処理系の言語についての報告である.

2. 動 機

本実験の動機は, 仕様の直接実行にあった.

仕様を動かすことができるならば, その仕様のなかに潜んでいる誤りとか, 機能の過不足について開発者と利用者との間で議論ができ誤解を避けられる. これによって, 開発中のシステムに対して利用者の要求が十分に確認でき, 不必要な不信感を利用者に抱かせずにすむ. 一方, 仕様の記法は開発方法に依存することが多いので, 仕様記述言語と開発方法とは切り離せない.

開発方法として JSD を採った理由は, JSD の仕様がシミュレーション・モデルの記述と解釈できるからである. さらに, JSD は仕様作成から実現までの広い守備範囲を持ち, 実際のシステム開発での実績のある方法であり, われわれの現場では十分に役立つと考えているからである.

JSD でいう仕様は通信型逐次プロセスの考えに基づいている．これは，並列に動くプロセスを扱うので多くのシステム開発者にとって不慣れなため，思わぬ誤解を起しかねないものである．そこで，効率が悪くても JSD の仕様を直接実行できることが仕様の正しさの確認のために望まれる．

3. 言 語 J

説明の都合上，報告する言語をＪと呼ぶことにする．

3.1 JSD の仕様

システムはプロセス群から構成されているとする．プロセスは逐次プロセスで命令型言語によるプログラムであり，通信文を送受信する命令を持っている．プロセスはその属するクラスを持っており，プロセス・クラスの宣言によってシステムに導入される．プロセスは次のようにしてシステムに出現する．まず，通信文の存在によってプロセス・クラスの複写が一つ作られ，通信文の宛先プロセス名がその複写プロセスの名として与えられる．このとき，その名を持つプロセスがすでに存在していれば，新プロセスは作られない．通信文は宛名プロセス名，発信プロセス名，通信文型，通信文本体から構成されている．プロセスは通信文のシステムへの投入によって起動される．プロセス間の相互通信には通信路があると考える．これもプロセス・クラス間に通信路クラスがあると考えて宣言する．この通信路をデータ・ストリームと言い，通信路を通る通信文たちをも指すことがある．プロセスの状態(状態ベクトル)を随時，別のプロセスが参照することを許し，このための命令を用意している．さて，以上を JSD 流に次のように解釈して利用する．システム開発の対象世界に存在する個々の実体 (Entity) をプロセスとみなす．この実体の振る舞い，つまり事象 (Event) はトランザクションの発生を意味する．このトランザクションを通信文とするシステム内のプロセス間の通信を考えると，対象世界のシミュレーションが実行できる．また，このようなプロセス群の定義が JSD の仕様に相当する．

3.2 言語Ｊの概要

JSD の教科書[2]に記載されているプロセス・テキストがプロセス・クラスの宣言に相当する．これは作業用の中間文書なので，そのまま実行はできない．その理由として情報の不足があげられる．実は必要な情報がプロセス・テキスト以外の文書にあったり，暗黙に示されていたりするためである．そこで，一つの言語を設定し，それらを陽に書くようにしたのが言語Ｊである．

3.2.1 言語Ｊの構文

```
⟨System⟩::='SYSTEM' ':' ⟨Name⟩
          ⟨EventDcl⟩
          ⟨DataStreamDcl⟩
          ⟨StateVectorDcl⟩
          ⟨ProcessDcl⟩
          'SYSTEM' 'END'
⟨EventDcl⟩::='EVENT'
             ⟨Event Descript⟩
             {⟨Event Descript⟩}
             'EVENT' 'END'
⟨DataStreamDcl⟩::='DATASTREAM'
                   ⟨DataStreamDescript⟩
                   {⟨DataStreamDescript⟩}
                   'DATASTREAM' 'END'
⟨StateVectorDcl⟩::='STATEVECTOR'
                    ⟨StateVectorDescript⟩
                    {⟨StateVectorDescript⟩}
```

'STATEVECTOR' 'END'
〈ProcessDcl〉::='PROCESS'
〈ProcessDescript〉
{〈ProcessDescript〉}
'PROCESS' 'END'

以降，言語 J の構文について simple bank system[2] の例を使って説明する．

3.2.2 システム宣言

仕様の名前を宣言する．

SYSTEM : simplebank

simplebank という名前の仕様

3.2.3 事　　象

事象は仕様に現われる実体の振る舞いであり，そのすべてを宣言する必要がある．

〈EventDescript〉::=〈EventName〉〈processId〉
'(' {〈Atribute〉} ')'

simple bank system では個々の事象の発生は実世界における“取引”の発生であり，その内容を示すものとして一般に“レコード”が生まれる．このレコード型を構成するフィールドの並びを事象の宣言で示すのである．このレコード型はプロセス間の通信文になる．たとえば，「INVEST という名前の事象は，名前 (name), 日付 (date), 金額 (amount) 等を属性（フィールド）とする」を，次のように書く．ただし，事象名の後にある属性は事象を引き起こしたプロセスの識別子（プロセス名）となる属性である．

言語 J では，

INVEST name (name date amount…)

と書く．

なお，事象として，システムへの機能要求から派生したプロセス (Function Process) が必要とするものも書く．

3.2.4 データ・ストリーム

データ・ストリームの名前とその通信文となる事象（レコード型）の名前を書く．

〈DataStreamDescript〉::=〈DataStreamName〉
'(' {〈Event Name〉} ')'

「データ・ストリーム C は INVEST, PAY-IN, WITHDRAW, TERMINATE などの事象による通信文を含んでいる」というのを，言語 J では次のように書く．

C (INVEST PAY-IN WITHDRAW TERMINATE)

システムはプロセス“プリンタ”を常に持っており，それへのデータ・ストリームもここに宣言しておく．

3.2.5 状態ベクトル

状態ベクトルは，それぞれの具体的なプロセスの状態を他のプロセスから直接参照できるように用意されたもので，プロセスのテキスト・ポインタとプロセスの変数からなる．

〈StateVectorDescript〉::='('{〈StateVectorName〉} ')'
〈ProcessTypeName〉
'(' {〈LocalVariable〉} ')'

言語 J では，仕様中のプロセス・クラスのすべての状態ベクトルを宣言する．「プロセス・クラス CUSTOMER-1 の状態ベクトルは CV という名前で参照でき，balance という変数からなる」ということを，次のように書く．

(CV) CUSTOMER-1 (balance)

言語Jでは，プロセスの受け取った通信文本体の履歴とか，プロセスのテキスト・ポインタを状態ベクトルから得るための関数を用意している．

3.2.6 プロセス・テキスト

ここではプロセス・クラスの構造を書く．JSD ではプロセス・クラスの構造を構造図 (Structure Diagram) か構造テキスト (Structure Text) で表す．言語Jでは構造テキストを使う．

以下，JSD の構造テキストとの対応を列挙する．

1) 構造テキストのうちで，構造，演算，条件を個別に書くものを採った．将来，これらをマージしたテキストを出力する機構を追加する予定である．
2) 条件は条件式を書く．
3) 通信プリミティブ

① SREAD

・通常の READ

「データ・ストリームCから通信文を READ する」というのを，

SREAD (C)

と書く．

・Rough Merge の READ

Rough Merge とは，プロセスが2個以上のデータ・ストリームと結合しているとき，通信文が available になったデータ・ストリームから順に通信文を受け入れる通信方法をいう．

「データ・ストリーム A B から通信文を Rough Merge で READ する」というのを，

SREAD (A B)

と書く．

② SWRITE……事象をデータ・ストリームに書く．

「データ・ストリームのクラス C に結合されたプロセス・クラスで，識別子が JUN-KATO のプロセスに事象が INVEST で，日付 (1986年 2 月 28 日)，金額 (¥ 1000,000)，… の通信文を送る」というのを，

SWRITE (C 'INVEST JUN-KATO 860228 1000000…)

と書く．

また，報告書をデータ・ストリームに書く場合は，

SWRITE (EXCEPTION-REPORT "OVER DRAW" ……)

と表す．

③ GETSV……アクセスしたい状態ベクトルをプロセスの識別子を使い get する．

「状態ベクトルの名前が CVで あるプロセス・クラスに属し，識別子が CID (通信文本体の属性 CID) の値を持つプロセスの状態ベクトルの COPY を SVREC へ格納する」というのを，

GETSV (CV .CIT SVREC)

と書く．

同じプロセス・クラスに属するプロセスの状態ベクトルを逐次適当な順に get する．

また，「状態ベクトルの名前が CV であるプロセス・クラスに属するプロセスの状態ベクトルの COPY を最初に get し SVREC へ格納する」というのを，

GETSV (CV "FIRST" SVREC)

と記し，「以降，順次，状態ベクトルの名前が CV であるプロセス・クラスのプロセスの状態ベクトルの COPY を get し，SVREC へ格納する」というのを，

GETSV (CV "NEXT" SVREC)

と書く．

4) 予約語

言語 J では次のような予約語を使用している．

ITR SET ALT END SEQ WHILE UNTIL TRUE FALSE EMPTY
POSIT ADMIT QUIT DO

5) 利用者が定義する関数と手続き

利用者はあらかじめ用意しておいた関数と手続きをそのまま使用できる．

6) プロセスの実体化

シミュレートするには，プロセスの実体化が必要である．言語 J では，プロセスの実体化の方法として以下の 2 通りを用意している．① 3.1 "JSD の仕様" で述べたようにシミュレーションの実行中に通信文によってプロセス・クラスの複写を一つ作る．② Create 命令でプロセス・クラスの複写を一つ作ることもできる．

4. 例題 SIMPLE BANK SYSTEM[2]

4.1 課　　題

ある銀行での預金は，顧客は一口座しか持てない．顧客はまず口座を開設 (INVEST) し，以降引き出し (WITHDRAW)，預け入れ (PAY-IN) を繰り返す．口座を閉鎖する (TERMINATE) ときもあり，この場合はそれ以降はいかなる取り引きもできない．

この口座は過払い (OVERDRAW) を許すが，預金利息や貸越利息はない．

この課題での機能要求は，次のとおり．

1) 顧客の残高をいつでも問い合わせ (ENQUIRY) できる．
2) 過払いが生じたときレポートを出す．

4.2 JSD で書いた仕様

simple bank で対象とするのは，CUSTOMER という型の実体たちであるということを実体リスト (Entity List) に書く．

1) 実体リスト……CUSTOMER の振る舞いは，INVEST, PAY-IN, WITHDRAW, TERMINATE という事象によって表すので，これらの事象を動作リスト (Action List) に書く．また，その際各々の事象には付随する属性があるので，属性も列挙する．

2) 動作リスト

動作名 (Action Name)	属性
INVEST	NAME AMOUNT
PAY-IN	NAME AMOUNT
WITHDRAW	NAME AMOUNT
TERMINATE	NAME REASON

3) 実体の構造……CUSTOMER の振る舞いは事象の列であり，その事象の発生規則

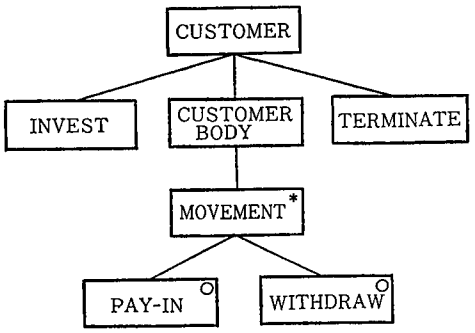


図 1 CUSTOMER の構造による表現

Fig. 1 Structure diagram of CUSTOMER

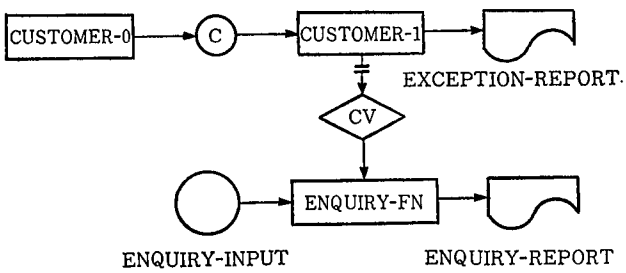


図 2 simple bank の SSD

Fig. 2 SSD for the simple bank example

を構造図か構造テキストで表現する.

CUSTOMER の振る舞いの規則は，INVEST という事象があり，CUSTOMER-BODY という事象の列が続いてあり，WITHDRAW という事象が終わりにある．CUSTOMER-BODY という事象の列は，MOVEMENT という事象の列の繰り返しであり MOVEMENT は PAY-IN という事象であるか，WITHDRAW という事象であるかのいずれかである．これを構造図で表現したのが図 1 である.

4) SSD (System Specification Diagram)……SSD とは，プロセス間の通信をネットワークとして図で表現したものである.

simple bank の SSD を図 2 に示す.

CUSTOMER-0 とは，実世界の CUSTOMER に対応するプロセス・クラスで，CUSTOMER-1 はモデル世界のプロセス・クラスである．ENQUIRY-FN とは，機能要求によって追加されるプロセス・クラスである.

プロセス・クラス間の通信は,

① CUSTOMER-1 は CUSTOMER-0 とデータ・ストリーム C によって結合されており，通信文を C から受け取る.

② CUSTOMER-1 はデータ・ストリーム EXCEPTION-REPORT に通信文を送る.

③ ENQUIRY-FN は CUSTOMER-1 と状態ベクトル CV によって結合されていて，多くの CUSTOMER-1 の状態ベクトルを一つの ENQUIRY-FN が参照する.

④ ENQUIRY-FN はデータ・ストリーム ENQUIRY-INPUT から通信文を受け取る.

⑤ ENQUIRY-FN はデータ・ストリーム ENQUIRY REPORT へ通信文を送る.

```
CUSTOMER-1 seq
  balance:=0;
  read C;
  INVEST seq
    balance:=amount;
  INVEST end
  read C;
  CUSTOMER-BODY itr while (PAY-IN or WITHDRAW)
    MOVEMENT sel (PAY-IN)
      PAY-IN seq
        balance:=balance+amount;
        read C;
      PAY-IN end
    MOVEMENT alt (WITHDRAW)
      WITHDRAW seq
        balance:=balance-amount;
        P-EXC-REPORT sel (balance<0)
          write 'overdrawn';
        P-EXC-REPORT end
        read C;
    MOVEMENT end
CUSTOMER-BODY end
  TERMINATE;
CUSTOMER-1 end

ENQUIRY-FN seq
  read ENQUIRYINPUT;
  ENQUIRY-FN-BODY itr
    ENQUIRY seq
      get state-vector of specified CUSTOMER-1;
      write 'balance is', balance;
      read ENQUIRYINPUT;
    ENQUIRY end
  ENQUIRY-FN-BODY end
ENQUIRY-FN end
```

図 3　CUSTOMER-1 と ENQUIRY-FN の構造テキストによる表現

Fig. 3　Structure text of process class CUSTOMER-1 and process class ENQUIRY-FN

5)　プロセスの構造

プロセス・クラス CUSTOMER-1 と ENQUIRY-FN を構造テキストで表現した結果として図 3 を得る.

```
SYSTEM : simplebank

EVENT
   invest     name (name amount)
   pay_in     name (name amount)
   withdraw   name (name amount)
   terminate  name (name reason)
   enquiry    (id)
EVENT END

DATASTREAM
   c   (invest pay_in withdraw terminate)
   exceptionreport ()
   enquiryinput  (enquiry)
   enquiryreport  ()
DATASTREAM END

STATEVECTOR
   (cv) customer_1 (balance)
   ()   enquiry_fn ()
STATEVECTOR END
```

```
PROCESS
  customer_1
  STRUCTURE
  customer_1 SEQ
    DO 1 DO 2
    invest SEQ DO 3  invest END
    DO 2
    customer_body ITR WHILE c1
      movement SEL c2
        pay_in SEQ DO 4 DO 2 pay_in END
      movement ALT c3
        withdraw SEQ
          DO 5
          p_exc_report SEL c4
            DO 7
          p_exc_report END
          DO 2
        withdraw END
      movement END
    customer_body END
    terminate ;
  customer_1 END

  OPERATIONS
   1: balance := 0
   2: sread(c)
   3: balance := .amount
   4: balance := balance + .amount
   5: balance := balance - .amount
   7: swrite(exceptionreport "overdrawn"
                my_identifier() balance)

  CONDITIONS
   c1:   withdraw
      or pay_in
   c2: pay_in
   c3: withdraw
   c4:  balance < 0

  customer_1 END

  enquiry_fn
  STRUCTURE
  enquiry_fn SEQ
    DO 1
    enquiry_fn_body ITR  WHILE c1
      enquiry SEQ
        DO 2 DO 3 DO 1
      enquiry end
    enquiry_fn_body END
  enquiry_fn END

  OPERATIONS
   1: sread(enquiryinput)
   2: getsv(cv .id svrec)
   3: swrite(enquiryreport .id "balance is "
                svrec.balance)
  CONDITIONS
   c1 : true
   enquiry_fn END
 PROCESS END
SYSTEM END
```

図 4　simple bank system の言語 J による仕様

Fig. 4　Specification of the simple bank system in language J

ただし，

① balance は，CUSTOMER-1 の状態ベクトルにおける局所変数である．

② read, write は通信用の命令（通信プリミティブ）である．

③ seq は連接，itr は繰り返し，sel alt は選択である．

```
"trace of statevector identifier processname " "j kato" customer_1
((BALANCE = 0))
((BALANCE = 1000000))
((BALANCE = 1200000))
((BALANCE = -50000))
((BALANCE = 5000))
"trace of statevector identifier processname " "a ito" customer_1
((BALANCE = 0))
((BALANCE = 120000))
((BALANCE = 132500))
"trace of datastreamname [identifier] " c "j kato"
(INVEST "j kato" 1000000)
(PAY_IN "j kato" 200000)
(WITHDRAW "j kato" 1250000)
(PAY_IN "j kato" 55000)
"trace of datastreamname [identifier] " c "a ito"
(INVEST "a ito" 120000)
(PAY_IN "a ito" 12500)
"trace of datastreamname [identifier] " enquiryinput enquiry_fn
(ENQUIRY "j kato")
```

図 5　仕様の実行結果

Fig. 5　Results of execution by language J processor

4.3　言語 J で書いた仕様

simple bank system の言語 J による仕様を図 4，その実行結果を図 5 に示す．

5. おわりに

実際に例題を実行してみると，よく確かめたにもかかわらず，仕様の入力に誤りがあった．これらの誤りは，構文解析で大半が検出され，残りは動かしてから発見された．仕様そのものが誤りなく書けたかどうかという点でも仕様を動かし，確かめる方法は有利である．

今回試作した処理系を，JSD を使うプロジェクトのための支援道具の中核とすることを考えている．それの前半(JSD による仕様作成)と後半（実現言語への変換）の支援系の開発を今後予定している．JSD 支援の道具としての評価はこれらの実現を待って行いたい．

処理系の実現に Lisp マシン KS-301 を使い，実現言語としては Common Lisp を使った．

なお，本稿は本年の 4 月に行われた「プロトタイピングと要求定義」シンポジウムでの発表[5]に手を入れたものである．

最後に，ご援助いただいた澤田晟司氏と，コメントをいただいた山崎利治氏，森澤好臣氏に感謝する．

参考文献

[1] J. R. Cameron, *JSP & JSD: The Jackson Approach to Software Development*, IEEE Computer Society, California, 1983.

[2] M. A. Jackson, *System Development*, Prentice-Hall International, 1983.

[3] D. D. McCracken and M. A. Jackson, "Life Cycle Concept Considered Harmfull", SIGSOFT SEN, Vol. 7, No. 2, 1982, pp. 22-32.

[4] 大野侑郎, "ジャクソンシステム開発法", 情報処理, Vol. 25, No. 9, 1984, pp. 955-962.

[5] 加藤潤三, "JSD (Jacksom System Development) 仕様の実行系の試作", 「プロトタイピングと要求定義」シンポジウム (1986 年 4 月 16 日), 情報処理学会.

執筆者紹介　加藤　潤三 (Junzo Kato)

昭和 46 年岡山大学理学部物理学科卒業．同年 4 月日本ユニバック(株)入社，現在，システム本部生産技術一部所属．

報告 ストリームを用いた論理型言語インタプリタ

A Logic Programming Interpreter Using Stream

大田一久

要約 データ・フロー*に基づく記述がソフトウェアの記述の方法として知られている．しかし，従来の命令的プログラミングの方法とは大きく異なっているため，ソフトウェア設計の初期の段階を除いて一般的ではない．データ・フローに基づく記述を使いやすくするためには，それが直接プログラムとして実行できることが望ましい．そのための一つの方法としてデータ・フローをストリームと呼ばれる“データとプログラムの複合オブジェクト”で表現する方法がある．データ・フローに基づく実行可能なプログラムをストリームを用いて書くことが可能で，この方法をストリーム指向プログラミングと呼ぶことにする．本稿では，ストリームの Lisp 上の実現について述べる．さらにデータ・フローに基づく記述による木探索の問題をストリーム指向プログラミングの例題として示す．この木探索の手法を論理型プログラミング言語の処理系を実現する際に応用することが可能であり，ストリームを用いた実験的なインタプリタについてその概要を紹介する．このインタプリタでは，逐次型と並列型の実行戦略を同一の枠組みで取り扱うことができる．

Abstract Data flow based description is known to be effective as a software description method. But because of the difference from the conventional procedural programming style, it is not commonly used except in the early stage of software design activities. It is desireble that the data flow based descriptions can be directly executed as programs. As one way to do this, data flows can be represented as streams which are the combined objects of program and data. Executable programs can be written in data flow based style using streams, and it should be called as the stream oriented programming.

In this report, an implementation of stream objects on Lisp is described. And a general tree search problem is shown as an example of the stream oriented programming. This idea is applicable to implementing logic programming systems, and an experimental interpreter using streams is shown. In this interpreter, both sequential and parallel resolution strategies are handled in the same framework.

Note ; The term “data flow” here means the macro data flow, which is somewhat different from the meaning used in the area of the dataflow architechtures and the dataflow machines.

1. はじめに

データの流れを中心にプログラムを記述しようとする試みがいくつかある．たとえばデータ・フローによるソフトウェアの設計技法[10]を始めとして，UNIX** のコマンド言語[22]，あるいはデータ・フロー・プログラミング言語と称しているもの[14]などである．これらの方法は，データの流れ間の関係からシステム全体の静的な性質，とくに入力と出力の関係が比較的容易に導き出せるという特徴を持っている．また，データの流れは，時間的系列とみなすことができ，従来の命令的プログラミングにおいて，繰り返しで記述されるようなアルゴリズムが簡単に記述できる．あるいはデータの流れを集合とみなすことにより，集合演算的な記述が可能である．

データの流れによる記述をプログラムとして実行するための一つの方法として，データ

* ここでいうデータ・フローとは，いわゆるデータ・フロー・アーキテクチャなどでのデータ・フローとは少し異なり，よりマクロなデータ・フローを指す．

** UNIX オペレーティング・システムは，AT&T 社が開発したものである．

の流れをストリームで表す方法がある．このようなプログラミングの方法をとくにストリーム指向プログラミングと呼ぶことにする．本稿ではストリーム指向プログラミングの特徴について述べ，要求駆動による無限リストを用いたストリームの実現を紹介する．さらにこれを用いて，試行錯誤的な探索型問題の記述，とくに後戻りを含む木構造を扱う問題の例を示す．また，論理型プログラミングの動作の形態がこの性質を持つことから，論理型言語のインタプリタをストリーム指向プログラミングで記述することを試みる．

なおストリームの実現法は H. Abelson らの文献[1]をもとに Common Lisp[11][18] で記述した．また論理型言語のインタプリタは，彼の文献[1]をもとに核心部分に3章で述べるアイディアを用いて Common Lisp に書き直した．組込み述語の扱い等は文献[13]を参考にした．なお，このインタプリタは Lisp マシン KS-301 上の Common Lisp で実際に稼動している．

2. ストリーム指向プログラミングとその実現

2.1 ストリーム指向プログラミング

ソフトウェア設計の際の技法，とくにモジュール分割の手法として，処理をいくつかの部分に分割し，その間をデータの流れで結ぶ方法が知られている[10]．この手法では，分割された各処理の間の制御関係は，入出力のデータの流れで結合されているだけであり，とくに順次データ列を扱うような場合に見通しのよい記述が得られる．ストリーム指向プログラミングはこの方法をより徹底させ，各処理モジュールの内容も同様の手法で記述しようとするものである．ストリーム指向プログラミングの特徴を見るために，簡単な例題を考えてみよう．データの流れの記述としては，図1に示すようなデータ・フロー・ダイヤグラムと呼ばれる視覚的表記を用いることにする．

たとえば，与えられた数列から7の倍数を取り出し，その2乗を計算するという問題を考える．これをデータ・フロー・ダイアグラムで表すと図2のようになる．この図では矢印のついた実線がデータの流れを表し，箱がデータの流れに対する処理を表す．また，ここでは現れていないが，矢印のついた点線は一つのデータ要素を表す．この記述ではデータ要素一つ一つについて繰り返すといった類の記述は現れず，各処理の入出力の関係は静的あるいは宣言的である．また，各データの流れを集合とみなせば，各処理は集合から集合への写像を与えることになり，数学的にも簡潔で美しい記述であるといえる．

つぎに，ストリーム指向プログラミングの記述をプログラムとして実行する方法を考える．このための方法としては，UNIX のコマンド言語[4],[9] で用いられている並列プロセスとその間の通信による方法，とコルーチン[17]による方法が考えられる．ここではスト

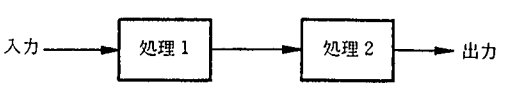


図 1 データ・フロー・ダイアグラム

Fig. 1 Data flow diagram

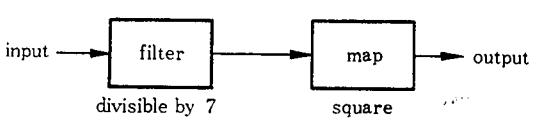


図 2 7の倍数の2乗

Fig. 2 Square of 7's multiples

リームを無限リストで直接表現する方法を用いる[1][8]．ストリームは，“無限リストの先頭の要素と残りを与えるプログラム（関数）から構成されるオブジェクト”として表現できる．ストリムを用いることによって，各処理はストリームを入出力とする操作として記述できる．ストリームについての基本操作は次の四つが考えられる．

head-stream (stream)

　　ストリームの先頭の要素を得る

tail-stream (stream)

　　ストリームの先頭を除いたストリームを得る

cons-steam (item, stream)

　　ストリームの先頭に要素を付加する

empty-stream (stream)

　　ストリームが空であれば真

ストリームそのものは，次のような組で表される．

s=[item, thunk]

ここで item は head-stream(s) に相当し，thunk を実行すると tail-stream(s) に相当する結果が得られる．すなわちストリームの二つ目以降の要素は，それを与えるプログラムの形で与えられ，データが実際に存在している必要はない．このようにストリームは，一種の不完全なデータ構造である．また，cons-stream を考えると，二つ目のパラメタはストリームそのものではなく，評価するとストリームが得られるプログラムとして扱わなければならない．すなわち，パラメタを評価しない名前呼び (call by name) のパラメタ渡し，および渡されたパラメタを後で評価する機能が必要である．このような機能は遅延評価 (delayed evaluation)[17] と呼ばれる．

この遅延評価のメカニズムによって，ストリームは無限に要素が流れるようなデータの流れを表すことができる．したがって，可算無限集合から要素を列挙するようなデータの流れ，たとえば図3のように1から始まる自然数の列が表現できる．また，要素は無限個でないにしても，概念的には集合全体を表しつつその一部について列挙，計算を行うことが可能である．

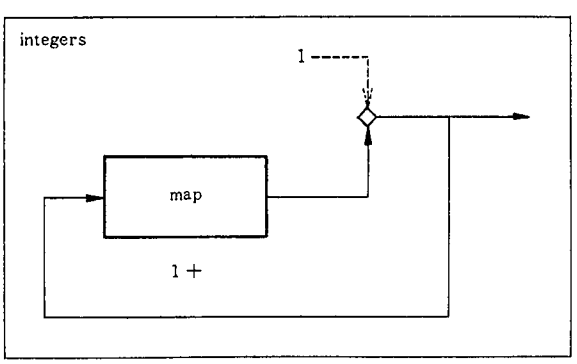


図 3　自然数の列

Fig. 3　A sequence of natural number

前述の基本操作に加えて，次のような標準的な操作を考えることができる．ここで，$\{S_i\}$ を要素 S_i からなるストリームとする．

filter-stream (predicate, stream)

条件を満たすものだけからなるストリーム i.e. $\{S_i \mid predicate(S_i)\}$

map-stream (function, stream)

各要素に操作を適用したストリーム i.e. $\{function(S_i)\}$

accumulate-stream (combiner, initial, stream)

ストリームの圧縮，次のように再帰的に定義できる

1) stream が空の場合 initial

2) そうでない場合 combiner (head (stream),
accumulate-stream (combiner,
initial,
tail (stream)))

以上の操作を用いると，図4のような例題を記述することができる．これは Eratosthenes のふるいで，一種のフィルタである．2から始まる自然数のストリームを入力に与えると，素数のストリームが出力として得られる．

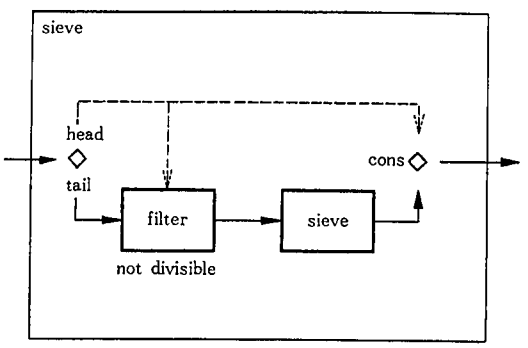


図 4 Eratosthenes のふるい

Fig. 4 Sieve of Eratosthenes

2.2 ストリームの実現

前節の考察に基づき，Lispを用いたストリームの実現について述べる．ストリームを実現するためには遅延評価のメカニズムが必要であるが，Lisp ではマクロおよび関数クロージャを用いてこの機能を実現することができる[1][2]．Lisp の関数名と表記法等については Common Lisp[11][18]に従う．

まず，遅延評価を制御する操作として，次の二つが与えられていると仮定する．

(delay 〈exp〉)

〈exp〉 の評価を一時中止する

(force 〈thunk〉)

一時中止された式の評価を再開する

delay では式の評価が一時中止されたオブジェクトがつくられ，これを force に与えるともとの状況で評価した場合と同じ結果が得られる．

この二つの操作を用いると，ストリームは先頭の要素と残りのストリームを与える遅延された式の組として実現することができる．Lisp で式を評価せずに関数に渡すためにはマクロを用いる．

```
(defmacro cons-stream (item stream)
  `(cons ,item (delay ,stream)))
(defun head-stream (stream)
```

```
    (car stream))
  (defun tail-stream (stream)
    (force (cdr stream)))
```

空のストリームを表すには nil を用いればよい.

```
  (defconst the-empty-stream nil)
  (defun empty-stream (stream)
    (null stream))
```

つぎに，delay と force の実現を考える．Lisp では関数をデータとして扱うことが可能で，関数引数関数，関数値関数といった高階関数を用いることができる．この機能を用いて評価が一時中断された式を表すオブジェクトを引数0個の関数，より正確には変数の値を与える環境を伴った関数クロージャとして取り扱えばよい．この方法は名前呼びの実現法としても知られている．したがって，force は与えられた関数を実行すること，delay は与えられた式を評価せずに関数を作るマクロとして表現される．実際には次のようになる．

```
  (defmacro delay (exp) '#'(lambda ( ) ,exp))
  (defun force (thunk) (funcall thunk))
```

この方法では，同じ式を表すオブジェクトに対して force が行われると，そのたびごとに実際の評価が行われ効率が悪い．そこで一度評価が実行されるとその結果を保存しておき，二度め以降は保存された結果を単に返すようにする．そのために次の関数を用いる．

```
  (defun memo-proc (proc)
    (let ((already-run nil)
          (result nil))
      #'(lambda ( )
          (if (not already-run)
              (progn
                (setq result (funcall proc))
                (setq already-run t)
                result)
              result))))
```

これを用いると delay は次のようになる.

```
  (defmacro delay (exp)
    '(memo-proc #'(lambda ( ) ,exp)))
```

このように一度評価されたものを再度評価しないようにした名前呼びのメカニズムを，とくに必要呼び (call by need) と呼んで区別することがある[2].

3. 探索型問題とストリーム

3.1 探索型問題の解決

問題の解を求める場合に，決められた方法に従って単純に計算するのではなく，複数の可能性からの探索が必要である場合も多い．このような探索は，解の候補の集合からある条件を満たす要素を見出すこととして抽象化される．このような問題を探索型問題と呼ぶことにする．これはデータ・フロー・ダイアグラムを用いて図5のように素直に表現できる.

探索型問題をデータ・フロー・ダイアグラムで表現する利点は，上記の抽象化をそのま

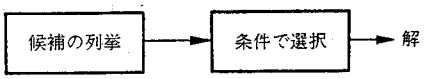


図 5 探索型問題の記述

Fig. 5 Description of search problem solving

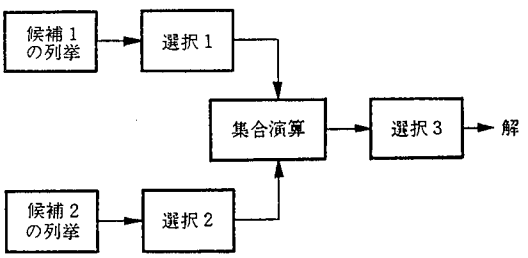


図 6 複雑な探索型問題の記述

Fig. 6 Description of complex search problem solving

ま記述できる点である．命令的プログラミングで記述する場合は，要素についての繰り返しを考慮する必要があり，候補の列挙と条件による選択を分離するためにはコルーチンや繰り返し子[17]などの工夫を必要とする．

もう一つはストリームを用いた場合，解の生成の制御が独立している点である．命令的プログラミングの場合，解をすべて求めるかどうかは繰り返しの構造そのものに影響する．

ストリーム指向プログラミングでは遅延評価のため，解の実際の生成は解法の記述とは独立であり，外部からの要求によって駆動される．このため全解探索も，解の存在を確かめるのもまったく同等の記述で扱うことが可能である．

これらの利点は問題が複雑になり，解の集合がいくつかの集合から直積や直和等の集合演算で与えられる場合にさらに顕著に現れる．このような場合は，図6のようなデータ・フロー・ダイアグラムでそのまま表すことができる．

また試行錯誤による問題解決もこの探索の一つの例と考えられる．すなわち，解に至る可能性の木のすべての節点の集合から解となる節点を選び出せばよい．とくにこの場合，木は完成されたものが与えられることは少なく，木を生成しつつ節点を探索することが多い．したがって木の生成と節点の判定を独立して記述でき，並行してパイプライン的に実行しうるストリーム指向プログラミングは有効であると考えられる．ただし，節点の生成・列挙の過程で，節点を調べる順序，あるいは枝の刈り込みといった工夫が必要である．この点については次節で扱う．

以上のような命令的プログラミングでは，繰り返しや再帰といった制御構造によって要素単位に処理していた問題を，ストリーム指向型では，集合演算に近い形で素直に記述できる．実際に問題解決に使用する際には無限に延びる枝が存在する可能性もあり，有限時間内に解が得られるためには要素を扱う時間的，空間的順序が問題となってくる．すなわち要素一つについての実際の処理に要する時間的，空間的資源は0ではなく，要素の間にある順序を仮定し解の存在およびそれに到達する資源が有限であることを確かめる必要がある．ストリームでは要素は順序を持って並んでおり，この問題について議論することが可能である．

3.2 ストリームを用いた木の探索

前節で述べたようにストリームを用いて木の探索を記述すると，節点の列挙と節点の判別を独立して考えることが簡単にできる．実際には節点の列挙の戦略，枝の刈り込み等を

考える必要がある．本節ではこの点を中心にストリームを用いた木の探索について述べる．

木の節点を列挙する際の戦略としては，深さ優先の縦型と幅優先の横型とがある．いずれの場合もある節点が与えられたとき，その節点から到達可能な節点の集合を得る手段を必要とする．ここでは，ある節点に対して到達可能な節点の集合をストリームとして得る関数 expand が与えられているものとする．節点の列挙は根となる節点をパラメタとし，節点のストリームを値とする関数 enumerate による．この関数を次に二つの戦略に従って詳細化する．

まず縦型の場合を考え，この関数を depth-enumerate と呼ぶ．これは一回の expand で得られた節点を根とする部分木に対して depth-enumerate を行い，得られた節点のストリームを要素の順序を保存して連結すればよく，図7のように表される．ここで connect はストリームを連結する操作である．

横型の場合は breadth-enumerate と呼ぶ．これは節点のストリームの各要素に expand を一回適用し，得られたストリームを再び breadth-enumerate に与えればよい．したがっ

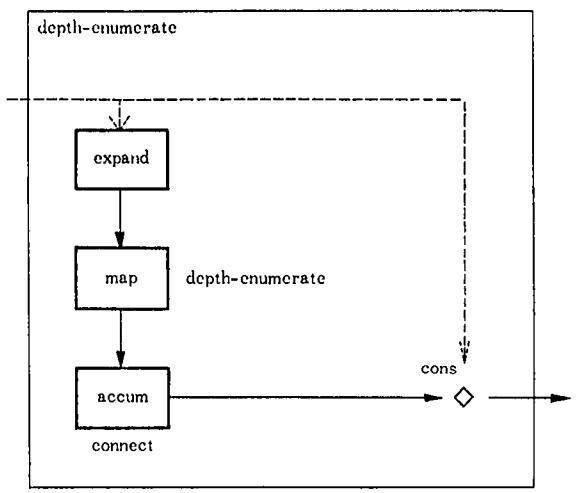


図 7　縦型の列挙
Fig. 7　Depth first enumeration

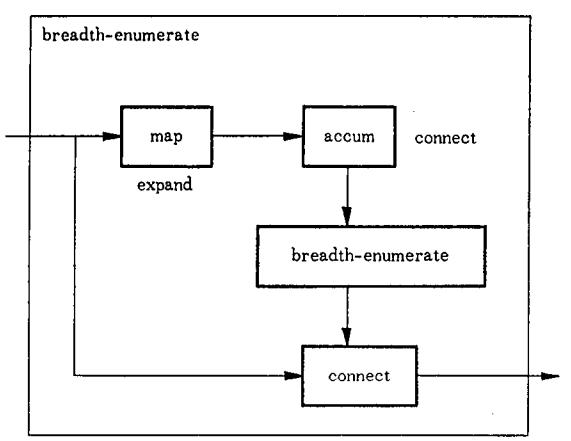


図 8　横型の列挙
Fig. 8　Breadth first enumeration

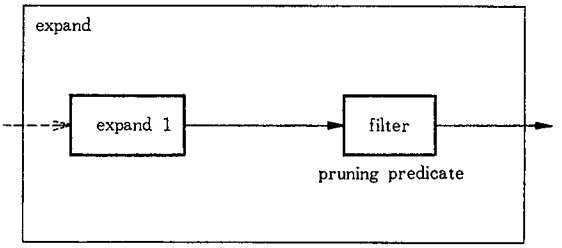


図 9 到達可能な節点の列挙
Fig. 9 Enumeration of reachable nodes

て，enumerate は根の節点一つからなるストリームを breadth-enumerate に与えればよい．図8のように表される．

つぎに枝の刈り込みについて考える．単純な方法としては，ある節点から到達可能な節点のストリームに対して，節点についての条件で制限する方法が考えられる．これは expand を図9のように，さらに2段階に詳細化し，節点のストリームを節点に対する適当な評価関数の値でソートすれば，最適優先の探索戦略を実現することができる．

4. 論理型プログラミングとストリーム

4.1 論理型プログラミングの計算過程

論理型プログラミングにおいては，その計算の過程は一階述語論理のサブセットによる定理の証明の過程と同等である[6][7]．論理プログラミングのプログラムはホーン節と呼ばれる論理式の集合で表される．

ホーン節は素論理式 (Atomic Formula) A_n から次のように構成される．

$$A_n \leftarrow A_1, A_2, \cdots, A_m \qquad (1 \geq n >= 0,\ m \geq 0)$$

このうち，とくに $n=0$ (つまり，A_n がない場合) の場合をゴール節，$n=1$ の場合を確定節と呼ぶことにする．プログラムは公理に相当する確定節と証明すべき定理に相当するゴール節からなる．計算はこのゴール節を背理法で証明する形で行われる．素論理式は述語 (Predicate) P と項 (Term) T_n から次のように構成される．

$$P(T_1, T_2, \cdots, T_n) \qquad (n \geq 0)$$

項Tは，変数Xかあるいは次のような関数 (Functor) F である．

$$F(T_1, T_2, \cdots, T_n) \qquad (n \geq 0)$$

定数は関数で $n=0$ の場合である．

ゴール節の証明は書き換え規則に基づき次のように行われる．ゴール節中の素論理式をそれと統一化 (Uunify) 可能な左辺を持つ確定節の右辺で書き換える．この書き換えを繰り返して空節が得られた場合，証明が成功したことになる．このとき一般には統一化可能な左辺を持つ確定節は複数個存在し，書き換えの可能性は枝分かれして木構造になる．この木をOR関係の木と呼び，この中から空節に至る経路を見出すことが，論理型プログラミングの計算過程ということになる．

もう一つ実行戦略にかかわる問題がある．すなわち，ゴール節中の素論理式を書き換える順序である．この場合，書き換えによって一つの素論理式が，それと統一化可能な左辺を持つ確定節の右辺によって0個以上の素論理式に書き換えられる．したがって，この関係も先程の場合と異なるがやはり木構造をなす．この木を AND 関係の木と呼び，すべての枝が葉の節点として空節を持つことが必要である．葉の節点に至るまで木を走査すると

きにやはり縦型と横型の順序が考えられる．前者はゴール節の最も左の素論理式を書き換えてゆくことに相当し，後者は各素論理式を一段階づつ均等に書き換えてゆくことに相当する．

実際に論理型プログラミング・システムを実現する際には，この木構造の探索の順序が実行戦略として問題となってくる[7][12]．すなわち，書き換えの可能性が複数ある場合，一つの可能性について可能な限り書き換えを行うか，すべての可能性について少しづつ同時に書き換えてゆくかである．前者は木構造の縦型探索に，後者は横型探索にそれぞれ相当する．

現存する論理型言語は，この二つの実行戦略によっておおまかに分類することができる．ここで，書き換えの可能性についての戦略をOR戦略，素論理式を書き換える順序についての戦略をAND戦略と呼ぶことにする．

まずPrologに代表され，OR戦略とAND戦略ともに縦型で行う逐次言語と呼ばれるグループがある．ある可能性について書き換えが失敗した際，OR関係の木を逆向きにたどることをバック・トラックと呼んでいる．書き換えの可能性を制限するためにカットと呼ばれる特殊な操作を持っているのが普通である．これは一つの可能性が選択されると他の可能性を捨てるような効果を持つ．

AND戦略を横型で行うのがAND並列言語と呼ばれるグループである．これはゴール節中の素論理式の処理が並列的に行われることからこのように呼ばれる．このとき各素論理式は変数を媒介として通信しながら並列に動作しているとみることができる．実際には，同じ変数が異なった値に統一化されることがないように，通信の方向を制限する工夫がなされている．

OR戦略を横型で行うのがOR並列言語と呼ばれる．これはゴール節に対して複数の解を並列的に求めてゆくとみることができる．実際には解の可能性を制限するためにコミットと呼ばれる特殊な操作を持つ．これは，非決定的に解を一つ選択する働きを持っている．解の選択を行うという意味では逐次言語におけるカットと同様の働きを持つ．

二つの戦略とも横型で行うのがAND-OR並列言語と呼ばれる．Concurrent Prolog, PARLOGなどがこれに相当する．さきに述べた通信の制御と解の選択に相当するメカニズムとを持っている．書き換えの際の統一化に方向性を持たせ，変数を媒介とした通信の問題を回避している言語もある[11][12]．

論理型プログラミングの計算過程では，無限に書き換えが続くような場合も考えることができる．この場合，他に空節に至る書き換えの経路があったとしても，縦型の実行戦略では停止しないことになる．このように実際のプログラミングに使用するためには実行戦略が大きな意味を持っている．

4.2 ストリームを用いたインタプリタ

前節で述べたように，論理型プログラミングにおける計算過程は木を生成しつつ走査するものとみなすことができる．またこの探索の順序によって実行戦略が決定されることを述べた．この過程の記述をストリーム指向プログラミングで行った場合，インタプリタの記述が見通しよくなり，また実行戦略を処理系の構造と独立にすることができる．本節ではこのアイディアに基づいたインタプリタの構造について述べる．

与えられたゴール節に対して書き換えを適用してゆく際に，統一化によって変数の値も決められてゆく．このことから書き換えが途中まで進んだ状態を表すためには，書き換えられたゴール節と統一化で与えられた変数の値が必要であることがわかる．この変数の値

を与える表に相当するものを環境 (Environment) と呼ぶことにする．また書き換えられたゴール節と環境の組をインスタンス (Instance) と呼ぶことにする．またこのときに同じ変数名で異なった値を表すことがあるので，確定節の本体に含まれる変数名を系統的に付け替えた上で書き換えを行う必要がある．

書き換えが枝分かれした木構造は，このインスタンスを節点として表すことができる．この節点から，到達可能な節点の集合はインスタンス中のゴール節の一つの素論理式を書き換えにより，得られるインスタンスの集合である．さらに空節を持つインスタンスが見つかった場合，証明が成功して解が一つ得られたことになり，そのインスタンスの環境に解としての変数の値が含まれている．このインスタンスを次々と列挙するようなストリームを考えることができれば，それに対して空節を含むインスタンスを選ぶフィルタをかければよい．これは図 10 のように表される．この図では，与えられたゴール節と空の環境から作られたインスタンスが入力となる．enumerate はこのインスタンスを書き換えて得られるインスタンスのストリーム〈instances〉を作り出す．enumerate として前節で述べた縦型を用いるが横型を用いるかによって OR 戦略が制御される．

インスタンスを列挙するストリームは，ゴール節と同じ述語を左辺に持つ確定節のストリームから作られる．この確定節のストリームを用いてインスタンスを一段階書き換える操作を expand とすればよく，これを図 11 に示す．この図では入力にインスタンスを与え，このインスタンスに含まれるゴール節中の一つの素論理式を選び，それと同じ述語を左辺に持つ確定節のストリーム〈rules〉を作る．さらにこのストリームの各要素で元のインスタンスを一段階書き換えたストリームを出力する．

この一段階の書き換えは図 12, 13 のように表される．インスタンスを一つの確定節で書き換える際に一般には統一化に複数の可能性があるので，unify は統一化によって決められた変数を含むように拡張された環境のストリーム〈environments〉を作り出す．この環境と書き換えの結果からインスタンスのストリームが作られ，apply-a-rule の出力となる．したがって apply-rules では二重になったストリームを connect によって一重のストリームにしている．

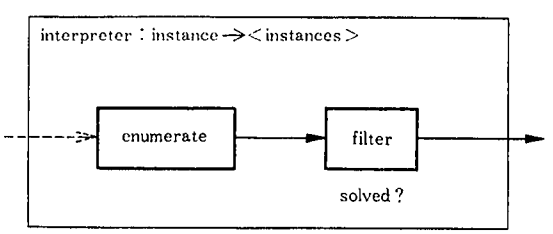


図 10 インタプリタ
Fig. 10 Interpreter

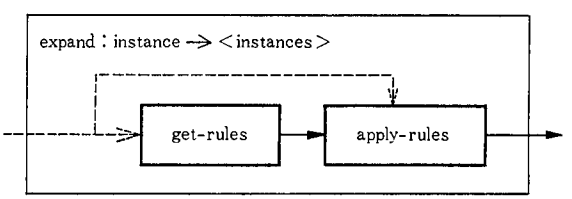


図 11 インスタンスの一段階の書き換え
Fig. 11 Rewriting of instances

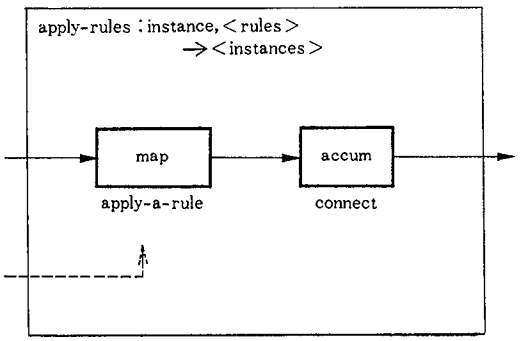


図 12　定節による書き換え
Fig. 12　Rewriting with definite clauses

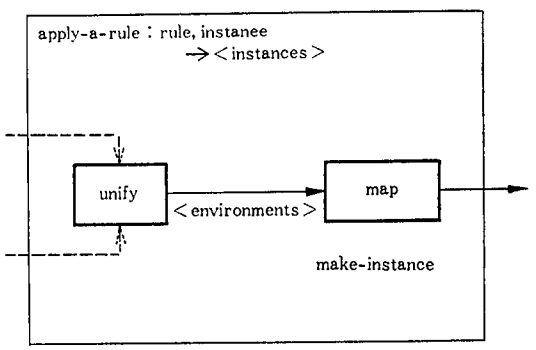


図 13　統一化とインスタンスの生成
Fig. 13　Unification and instance generation

一方，ゴール節中の一つの素論理式を選んで書き換える際の順序の問題がある．これは一つのインスタンスの中のゴール節に対して素論理式を節点とする木を生成し，走査するものとして考えることができる．このときある節点から到達可能な節点の集合は，その節点に対応する素論理式の書き換えで用いる確定節の右辺の各素論理式である．ただし書き換えの可能性は複数なので，素論理式を列挙するストリームを考えるのは困難である．そこで，AND 戦略を制御するためにインスタンス中のゴール節をキューとして表現し，常に先頭の素論理式を書き換えの対象とすることにする．書き換えの結果得られる素論理式の集合をキューの先頭に加えるか，末尾に加えるかで縦型，横型が制御できる．

4.3　インタプリタの外部仕様

インタプリタは登録モードと問合せモードの二つのモードを持っていて，それぞれ確定節の登録とゴール節による問い合わせを行う．また，特殊な処理のためにいくつかの組み込みの述語とコマンドが用意されている．

ホーン節の表現は Lisp の S 式を用いて次のようにする．ここで A_n は素論理式の表現である．

$(A_0\ A_1\ A_2 \cdots A_n) \quad (n \geq 0)$

上記の表現は，登録モードでは A_0 を左辺とし $A_1, A_2, \cdots, A_n$ を右辺とする確定節として扱われる．問い合わせモードでは左辺のないゴール節として扱われる．素論理式と項は S 式を用いて次のように表す．ここでは P は述語で T_n は項である．

$(P\ T_1\ T_2 \cdots T_n) \quad (n \geq 0)$

また，変数は先頭の文字が？であるシンボルを用いることとし，それ以外のシンボル，リ

スト，数値などはすべて定数として扱う．

組み込み述語は表1のとおりである．

表 1　組み込み述語

Table 1　Built-in predicates

組み込み述語	説　明
(:assert 〈clause〉)	〈clause〉で与えられる確定節を登録する．
(:retract 〈formula〉)	〈formula〉で与えられる素論理式と統一化可能な左辺を持つ確定節を削除する．
(:show 〈predicate〉)	〈predicate〉で与えられる述語として登録されている確定節を表示する．
(:unify 〈term〉 〈term〉)	〈term〉で与えられる二つの項を統一化する．
(:predicate 〈term〉)	〈term〉を Lisp の式として評価し，結果が NIL であれば失敗，そうでなければ成功する．
(:function 〈term〉 〈term〉)	一つめの〈term〉を Lisp の式として評価し，結果を二つめの〈term〉と統一化する．
(:true)	常に成功する．
(:false)	常に失敗する．

システム制御のためのコマンドは，表2に示すとおりである．

表 2　システム制御コマンド

Table 2　System control commands

コマンド	説　明
:halt	インタプリタを終了させる．
:user	登録モードにする．問い合わせモードに戻るには NIL を入力する．
:breadth	インタプリタを並列 OR モードで動作させる．
:depth	インタプリタを逐次 OR モードで動作させる．
:parallel	ゴール節内の素論理式を並列に書き換える．
:sequential	ゴール節内の素論理式を左から順に書き換える．
"〈pathname〉"	〈pathname〉で表されるファイルから確定節の集合を読み込む．

インタプリタは問い合わせに対して解が見つかった場合，問い合わせのゴール節をそれに含まれる変数を解として見つかった値で置き換えた形で表示する．この状態で，インタプリタは入力待ちになり，斜線(／)を入力すると他の解を探索する．それ以外の文字が入力された場合，問い合わせモードに戻る．インタプリタの使用例を図14に示す．

このインタプリタでは，カットに相当する解の選択を制御する機能はなく，また変数の統一化の方向に関する制限を与える機能もない．したがって論理型プログラミング・シス

```
Query〉〉 : user
Assert〉〉 ((append ?a . ?x) ?y (?a . ?z)) (append ?x ?y ?x))
Assert〉〉 ((append ( ) ?v ?v))
Assert〉〉 nil
Query〉〉 ((append ?m ?n (a b c d)))
((append ( ) (a b c d) (a b c d)))  /
((append (a) (b c d) (a b c d)))  /
((append (a b) (c d) (a b c d)))  /
((append (a b c) (d) (a b c d)))  /
((append (a b c d) ( ) (a b c d)))  /
Done.
```

図 14　インタプリタの使用例

Fig. 14　Interpreter sample usage

テムとしての性格は，pureProlog＋実行戦略とでも言うべきものとなっている．しかしインタプリタの構造のモジュール性が高く，実行戦略の切り換えが可能である．したがって，本節で述べた構想に従ってより実用的な機能を持ったインタプリタを作成すると，論理型プログラムの評価，検証，デバッグ等に有効であると考えられる．

5. おわりに

ストリーム指向型プログラミングの考え方，ストリームを用いた関数的アプローチとその実現を紹介し，問題解決とくに木構造探索への応用を示した．この場合，試行錯誤を含む解決法に対して集合演算的な記述が可能である点が特徴である．さらに，この点を応用して論理型言語のインタプリタを記述した例を示した．

この方式の記述ではシステム全体をデータの流れに基づいて分割し，その部分をさらに局所的なデータの流れに従って詳細化するというアプローチが向いている．また，データの関係についてのみの記述であり，命令的記述に比べると抽象度が高い．このため，一種の実行可能仕様としての性格を持っていると考えられる．また試行錯誤を含む問題解決をより宣言的に記述する方法として，論理型の記述と共に有力な手段を提供すると考えられる．

また，今回はストリーム指向型の記述を実行する方法としてはストリームを直接実現する方法を用いた．これに対して，問題の性質によっては，プログラムの等価変換によって繰り返しを用いた命令的プログラムに変換可能な場合がある．このアプローチでストリームによる記述の von Neumann マシンに対するコンパイラを考えることが可能であろう[14]．

表記法については本稿ではデータ・フロー・ダイアグラムおよび Lisp による関数的記述を用いた．データ・フローによる記述を本格的に用いるためには，表記法についての検討が必要であると考えられる．関数的プログラミングでストリームを扱った例[8]，データ・フロー言語として繰返しに相当する記述を持つもの[14]がある．さらには，視覚的プログラミング言語としてデータ・フロー・ダイアグラムをそのまま用いる方法も，ユーザ・インタフェースのためのデバイスの発達から考えて非常に有望である．

また，マシン・アーキテクチャとしてのデータ・フロー[3]との関係についての検討も必要である．現在のところ，データ・フロー・マシンの研究は，既存の命令的記述によるプログラムから並列性を抽出して高速に実行することに主眼が置かれているように見受けられる．本稿で採り上げたようなデータ・フローによる記述の効率の良い実行のためには，von Neumann マシンよりも，このようなマシンの方が向いていると考えられる．逆に新しいハードウェアの性質を生かすための処理の記述法としても有効であると考えられ，この点についての今後の研究が期待される．

参考文献 [1] H. Abelson and G. J. Sussman, *Structure and Interpretation of Computer Programs*, The MIT Press, 1985.
[2] J. Allen, *Anatomy of Lisp*, McGraw-Hill, 1978.
[3] 雨宮真人，“データフローアーキテクチャについて”，コンピュータソフトウェア，Vol. 1, No. 1, 1984, pp. 42-63.
[4] S. R. Bourne, *The UNIX System*, Addison-Wesley, 1982.
[5] W. F. Clocksin and C. S. Mellish, *Programming in Prolog*, Springer-Verlag, 1981.
[6] 後藤滋樹，古川康一，“論理型計算モデル”，情報処理，Vol. 24, No. 2, 1983, pp. 123-132.
[7] 後藤滋樹，PROLOG 入門：知識情報処理の序曲，サイエンス社，1984.
[8] P. Henderson, *Functional Programming : Application and Implementation*, Prentice-

Hall, 1980.
(邦訳) 杉藤芳雄, 二木厚吉共訳, 関数型プログラミング, 日本コンピュータ協会, 1985.
[9] H. McGilton and R. Morgan, *Introducing the UNIX System*, McGraw-Hill, 1983.
[10] 紫合治, "ソフトウェア設計法について", コンピュータソフトウェア, Vol. 1, No. 2, 1984, pp. 55-68.
[11] G. L. Steele, Jr., *Common Lisp: The Language*, Digital Press, 1985.
(邦訳) 井田昌之訳, "Common Lisp", bit 別冊, 共立出版, 1985.
[12] 竹内彰一, "論理型並列プログラミング言語—Concurrent Prolog", コンピュータソフトウェア, Vol. 1, No. 2, 1984, pp. 25-37.
[13] 梅村恭司, "Small Prolog インタプリタ移植のすすめ", bit, Vol. 15, No. 6, 1983, pp. 58-66.
[14] W. W. Wadge and A. E. Ashcroft, *Lucid, the Dataflow Programming Language*, Academic Press, 1985.
[15] P. H. Winston and B. K. P. Horn, *LISP: Second Edition*, Addison Wesley, 1984.
[16] 米沢明憲, "関数型計算モデル", 情報処理, Vol. 24, No. 2, 1983, pp. 113-122.
[17] 米沢明憲, 木村 泉, "算法表現論" 岩波講座情報科学, 第12巻, 1984.
[18] 湯浅太一, 萩谷昌己, "Common Lisp 入門 1-3" bit, Vol. 17, 1985, No. 4, pp. 51-61, No. 5, pp. 38-50, No. 6, pp. 54-63.

執筆者紹介 大田 一久 (Kazuhisa Ohta)
昭和56年, 慶応義塾大学工学部数理工学科卒業. 同年, 日本ユニバック(株)入社. エンド・ユーザ言語の開発に従事. 昭和60年から Lisp マシン関連の業務を担当. 情報処理学会, 日本ソフトウェア科学会会員.

報告 バックパネルの潜在的不良検出手法

Detection Method for Potential Back Panel Defects

小塩英造, 中條幸雄, 三位　潔

要　約　UNIVAC シリーズ 1100 システム 80 のトラブル・サポート担当として，長年にわたり種々のバックパネル・トラブルと取組んできたが，1984 年には静電発生器を利用したバックパネルの不良箇所検出方法を開発した．

この方法は，高電圧での放電現象により，バックパネル内の異常な近接箇所（電圧バスとシグナルピン間，シグナルピンとワイヤー間等）を検出するものである．

これにより，非常に難解な間欠障害を解決すると共に，潜在的な不良箇所を検出することで，障害の事前防止が可能となった．

われわれは，1984 年 1 月より，この手法を IOU チャネル・モジュールに対する保守に取り入れ，過去に障害の多発したモジュールの点検を進めると共に，緊急な障害の修復に活用してきた．

この結果，障害を効率的，かつ確実に修復するという点，および障害の事前防止による安定稼動の推進という面において，かなりの成果を上げることができた．

本稿は，この新しい手法とその評価について紹介している．

Abstract　A new method has been developed to discover the potential defects in back panels using static electricity generators.

The purpose of this method is to locate the defective points on the panel that cause very intermittent system errors in nets, and therefore, allow preventive corrective actions. This method utilizes the unique characteristics of static electro discharge, a high voltage but harmless current flow when discharged.

Anomalously close points that may intermittently fail during the normal operation can be easily detected by locating the discharging point when voltage is applied between signal pins and ground/voltage bus, or wire-nets and signal pins.

This method was very effective when used in 1100/80 IOU back panel troubleshooting.

At NUK, panels that showed instablilty in the field were checked and corrected effectively. Very successful results were achieved for hardware stability improvement and for troubleshooting.

1. はじめに

昨今，コンピュータ・ハードウェアのスタビリティに対する顧客の要求は，その社会的影響度の増大に伴い年々厳しいものとなっている．

一方，ハードウェア・トラブルの要因となるコンピュータ内の潜在的不良を検出する技法については，従来より種々のマージナル・テスト（電圧，温度，クロック，振動，他）が主流であり，これに代わる画期的技法については，われわれの知る限りでは存在していない．

われわれは 1984 年に UNIVAC シリーズ 1100 システム 80 における IOU バックパネル・トラブル対策を推進する過程で，静電気発生器を用いてバックパネルの潜在的不良を検出する，新しい検査手法を開発した．

この検査手法は，最近のマルチレイヤのバックパネルには適応できない点，またモジュ

ールを取りはずす必要があるためフィールドで簡単に実施できない点，等に難はあるものの既存のワイヤリング構造のバックパネルに対しては，100パーセント近くその潜在的不良を検出することが可能であり，今までにない新技法といえる．

また，この検査手法は，現在IOUチャネル・モジュールに対し実施し，後述するような成果を得ているが，当然他のユニットおよび他のプロダクトへの応用も十分可能である．

2. 背　　景

この章では，静電気発生器によるバックパネル検査手法の開発に至る背景として，従来のUNIVACシリーズ1100システム80におけるIOUのバックパネル・トラブルおよびその対策の概要について述べている．

2.1 バックパネル・トラブル

最近のプロダクトでは，新テクノロジとしてマルチレイヤ構造のバックパネルが採用されており，UNIVACシリーズ1100システム80においてもCPU, SIUではこのマルチレイヤのバックパネルを使用している．

これに対しUNIVACシリーズ1100システム80のIOUとMSUは，既存のワイヤリング構造のバックパネルを使用している．

バックパネルの品質面からこの2種の構造を比較した場合，トラブル発生状況のデータからマルチレイヤ構造のバックパネルが品質的に優れていること，およびとくにIOUの場合にバックパネル・トラブルが多いことがわかっている．

現状におけるこの品質面の相違は，その構造上の差に起因することの他に，ワイヤリング構造のバックパネルが一つの大きな問題を持っていることが最大の要因となっている．

この大きな問題というのが，バスプレートのスルーホールとシグナルピンの間のショート・トラブルである．

代表的なバックパネル・トラブルとしては，コールド・フローといわれるワイヤの被覆むけによるピンとの接触トラブルがある．しかしIOUの場合，バスとピン間のショート・トラブルが異常に多く発生していることが判明している．

2.1.1 バスとピン間のショート問題

この問題については，種々分析し検討を行ってきたが本稿の趣旨からはずれるため紹介にとどめる．

IOUのバックパネル構造を図1に，トラブルの状態を図2に示す．

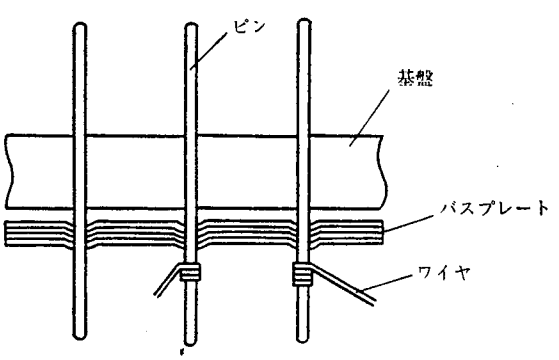


図1　バックパネルの構造
Fig. 1　Back panel structure

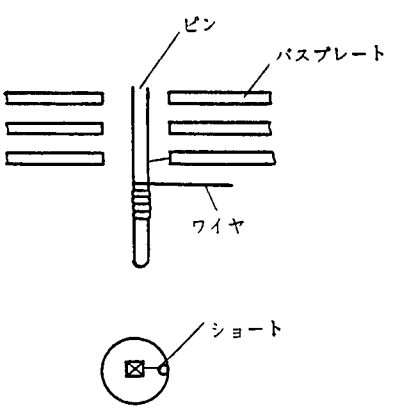


図2　ショート・トラブル
Fig. 2　Short-circuit trouble

このトラブルの根本原因については，現在まで解明に至っていないが，以下の二つの原因が考えられている．

1) ハンダくずによるショート……これはトラブルの多発したバックパネルを Sperry 社に調査依頼した際の Sperry 側の回答に示唆されたもので，電圧およびグランド・ピンをバスプレートと結合するために使用しているソルダー・ドーナツ（ハンダの小リング）から生じるハンダくずが原因となってショートが発生するという説である．

この問題について，Sperry 社では 1983 年にソルダー・ドーナツのベンダを変更して解決しているという解答をしている．

2) ウィスカ現象によるショート……これはすでに一般的に知られている問題で，スズやハンダメッキ等に発生する直径 1～5 μm，長さ 1～5 mm の針状の結晶でスズ・ウィスカと呼ばれ，成長速度は早いもので 1 か月に 1 mm とされている．

この発生原因および防止法については，まだ十分に解明されていない．

現在までのトラブル発生状況の分析結果では，ウィスカによるショートが大部分のトラブルに当てはまると判断している．これに対する根本的対策は非常にむずかしいが，後述する抑止効果がこの静電気発生器検査法で期待できる．

3. バックパネル検査手法の開発

ここでは，静電気発生器による検査手法の開発に至る経過について述べる．

3.1 検査手法のねらい

この検査手法を考える上で最も重要な点は，現実にトラブルを起こしている状態を検出するのではなく，今後トラブルを引き起こす可能性のある状態（箇所）およびインターミテントなトラブルとして現象が消えている状態等の，潜在的不良箇所を検出することである．

また，もう一つの要素としてはバックパネル・トラブルの内でも，最も多発しているバスプレートとピン間のショート・トラブルを確実に検出することである．

具体的には，図 2 に戻って考えると，ピンとバスプレートのホールの間がショートには至っていないが，きわめて近接した状態を検出することである．

3.2 検査手法の検討

検査手法の検討に際し，既存の検査法でこの問題に適応できるものの有無を調査したが存在しておらず，独自の手法を検討することにした．

前述したように，この検査の主眼はバスプレートのホールとピンの間が異常に近接した状態を検知することにあり，このことは何らかの方法でこの間のギャップが測れればよいことになる．

この方法として，ギャップと放電電圧がある程度比例するということを利用する測定方法を以前より考えていたが，この方法には二つの大きな問題点があり，実際に行うまで至らなかった．

つまり，問題点の一つは，高電圧（2～4 kV）を扱う必要から作業者に対する危険性があること，もう一つの問題は高電圧での放電によってピンおよびバスがダメージを受けることである．

今回，この高電圧源として静電気発生器の利用を思いつき試行した結果，人体に対する危険性も少なく，ダメージの問題も起きないとの結果を得て実現に至った．

4. 静電気発生器によるバックパネル検査手法

4.1 検査手法の原理

この不良箇所検出方法は非常に簡単な原理で，空気が絶縁破壊を起こし放電を開始する電圧が，その電極間の間隔にある程度比例するという性質を利用したものである．

現在すでにショートを起こしている箇所については，テスタ等により容易に検出できるが，われわれの問題としている，今後ショートを起こしうる潜在的不良箇所の検出が，この検査手法の目的である．

また，この潜在的不良箇所を想定すると，たとえばバスプレートの穴とピンの間隔が非常に狭くなっている場合，またワイヤの被履がむけた箇所がピンと近接しているような場合である．

こういった箇所に対しバスプレートとピンを各々電極とし，電圧を加えた場合，正常な箇所に比べ低い電圧で放電が開始することになる．

したがって，バスプレートを一方の電極として各々のシグナル・ピンに電圧を加えていき，一定の値より低い電圧で放電を開始するピンを選び出すことで，不良箇所を検出することができる．

図3，図4にこの原理を示す．

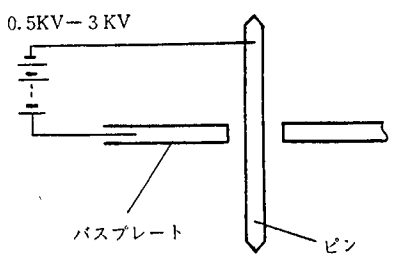


図 3　不良箇所検出の原理
Fig. 3　Principle of inspection method

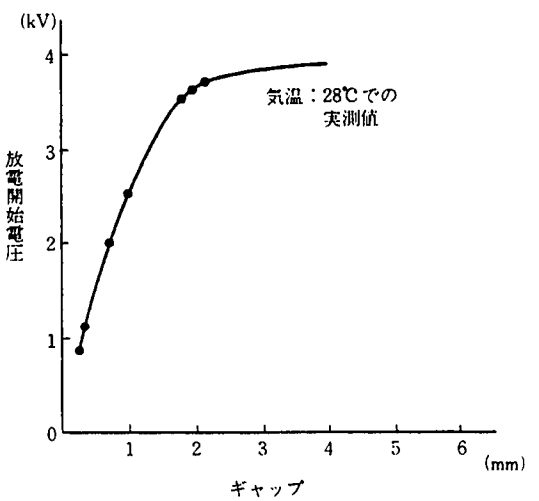


図 4　放電開始電圧の関係
Fig. 4　Discharge voltage vs spark gap

4.2 検 査 方 法

図5に示すように，静電気発生器のグランド側をバックパネルの各電圧バスと接続し，スイッチ・パッケージのソース・ポイントにプローブを接続する．

このスイッチ・パッケージをバックパネル上のスロットに挿入後，静電気発生器をオンにし電圧を上げてゆく．

これにより，スイッチ・パッケージを通しそのスロットのすべてのシグナル・ピンに対し電圧がかかり，バスプレートとの間隔が最も短い箇所から放電が開始される．

通常の放電は，火花放電と称されるパチパチという音と青白い火花を伴った放電であり，音と光より放電を開始を知ることができるが，間隔が狭い場合この放電音と光が弱くなるため，EMI ロケータを使用し放電の有無を確認している．

また，火花放電の他にコロナ放電といわれる音を伴わない放電が，バスプレートの穴が偏心しているような箇所でまれに起こる場合がある．このコロナ放電に対しては，EMI

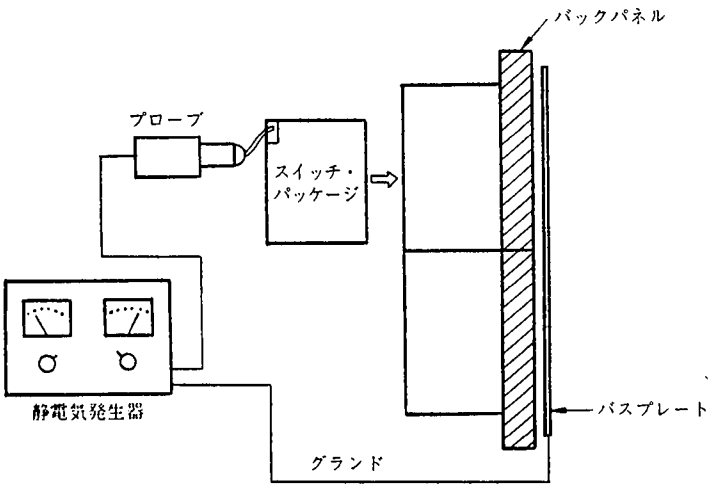


図 5 静電気発生器による検出方法

Fig. 5 Schematic of inspection method

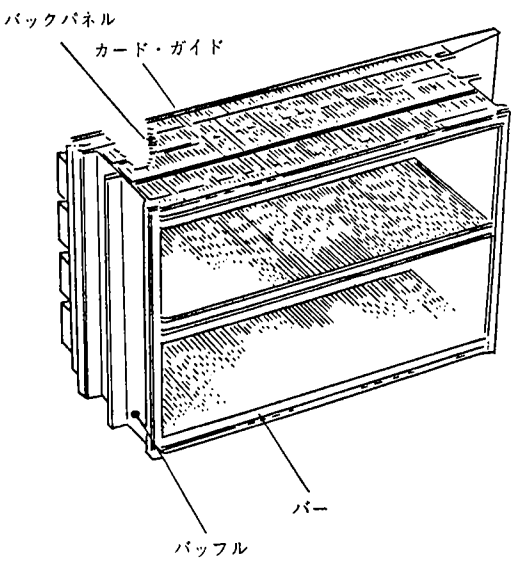


図 6 IOU チャネル・モジュール

Fig. 6 IOU Channel module

ロケータでも検出できないが，電圧を非常に高く上げても放電音がしないということで，ある程度推測ができる．

4.3 使 用 器 材

ここでは，この検出方法で使用する器材についてその概略を述べる．

1) IOU チャネル・モジュール……IOU バックパネルの部分構造については，すでに前章で述べているが，全体としての構造を図 6 に示す．カード・スロットは，A, B 各シェルフ 76 スロットで，各スロットには 80 本のピンが配置（2 列）されている．

2) 静電気発生器……これは，日本ユニバック(株)で開発した三基電子工業の静電気障害試験器 SET-15 を使用している．

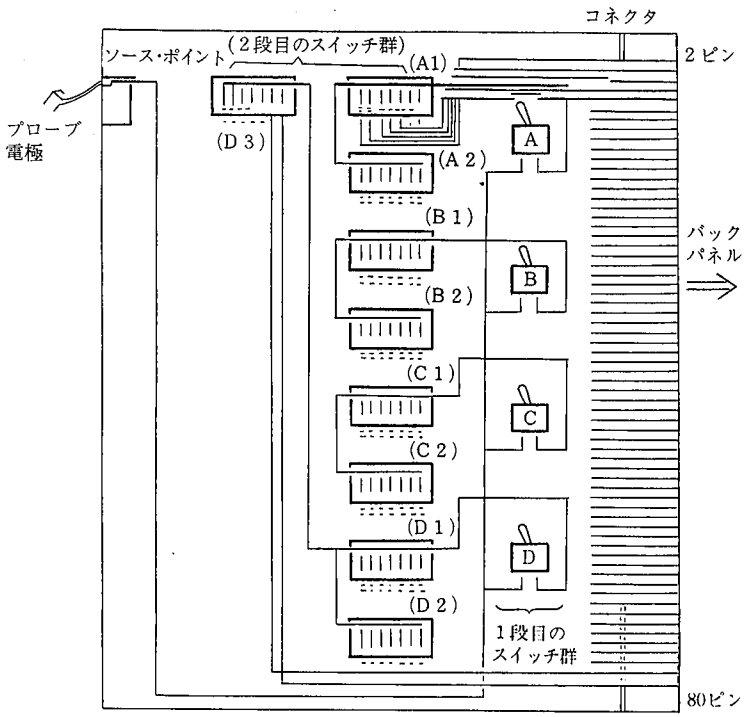


図 7　スイッチ・パッケージ

Fig. 7　Switch package

定格は，最大試験電圧 15kV，プローブのキャパシタ・ユニット 120PF となっている．

3)　EMI ロケータ……これも日本ユニバック(株)が開発したもので，三基電子工業の ES-81V を使用している．

使用形態は，AUTO-RESET および BUZZER スイッチをオンで使用する．

これにより，放電を検知すると1秒間ブザーの発信音が保持される．

4)　スイッチ・パッケージ……これは IOU のパッケージを改造したもので，使用目的はスロット単位での良否判定を行うことで検出作業の効率化にある．

構造は，図7に示すようにプローブの電極を接続するソース・ポイントより2段階のスイッチを経由し，コネクター上のピンに継がっている．

1段目のスイッチは，不良箇所を69本のシグナル・ピンを大きく四つのブロックに分類するためのものである．2段目のスイッチは各々のピンに対応して設けられており，不良箇所を一つのピンに絞るためと，シグナル・ピンがすでにグランドに配線されている場合，そのピンを含むスロット全体をテストするために該当するピンをカットすることを目的として設けたものである．

4.4　不良箇所の検出とリペア方法

ここでは，実際に不良箇所を検出してゆくプロセスとそのリペア方法について述べる．

1)　放電開始電圧……バスプレートの穴とピンの間隔は 0.8mm 程度で，正常な箇所の放電開始電圧は，環境条件（温度，湿度，気圧等）によって多少異なるが，2〜2.5kV である．実際には，穴の変形等により 1.7kV くらいで放電する箇所が多くあり，良否判定の基準としては，1.5kV 以下で放電開始する箇所を不良と判断した．

ただし，これは一つのピンに対し電圧をかけた場合で，スイッチ・カードを使用してスロット全体に電圧をかけた場合は，この値は正常なスロットで 3～4kV となり，良否判定の目安は 2kV 前後となる．これは，高圧電源として静電気発生器という非常に微弱な電源を使用していることによって，スイッチ・パッケージの持っている静電容量と絶縁性能が大きな影響を与えるためである．この影響度は湿度が高い場合顕著に現れ，雨天の日では正常スロットで 5kV くらいまで上がることがある．

したがって，この良否判定の目安は天候等の状況により，2～3kV の間で設定している．

ただし，最終的な良否の判断は，一つのピンに直接プローブを接続し行うことが原則であり，スロット単位での良否判定は作業面での便宜的手段である．

2) ピンとバスプレート間不良の検出とリペア……まず，音と火花によって放電箇所を探してゆくが，ワイヤリングが密集している箇所では，発見されにくい場合が多い．この場合，スイッチ・カードの2段のスイッチにより該当ピンを求め，さらにそのシグナル・ネットのワイヤをはずしながら不良ピンを求めるといった，非常に手間のかかる作業が必要となる．

このリペア方法としては，CPU，SIU のバックパネル・リワークに使用しているブッシュ（硬質の絶縁チューブ）が適当で，これをピンの根元に挿入しバスプレートと絶縁する方法を採っている．

3) コールド・フローの検出とリペア……当初この方法では，電圧バスおよびグランド・ピンに対するシグナル・ワイヤのコールド・フローは検出できるが，シグナル・ピンとのコールド・フローについては検出できないと考えていた．これは電極の一方をバスプレートに接続する方式に対して，個々のシグナル・ネットどうしを電極とするには膨大な組み合わせが必要となり，実際にこれを行うことは自動化されたシステムでない限り不可能なためである．

しかしながら，今回の検出方法による試行において，この検出方法でもシグナル・ネット間のコールド・フローが検出できることが判明した．

この理由は，電極とは直接に接続されていないシグナル・ネットに対しても，そのネット自体の持つ静電容量に対して，コールド・フローの箇所において弱い放電が間欠的（1～3秒間隔）に起こるためである．

シグナル・ネット間のコールド・フローを検出できることがわかったため，バックパネルにおける潜在的不良箇所について，ほぼ 100 パーセントに近い検出が可能であるとの自信を得た．なお，このコールド・フロー箇所の発見方法は，2)で述べた方法と基本的に同じである．

リペア方法については，被覆のむけたワイヤを交換するのが原則であるが，ワイヤの交換がむずかしい場合等では接触側のピンにスリーブ（ビニール・チューブ）を挿入する方法も併用している．

5. 実績と評価

1984年1月より IOU のバックパネル・トラブル対策を開始し，2月からこの検査手法を活用してきた．

1984年の9月末には18のモジュールに対し，この検査方法による点検を実施した．これら18のモジュールは，現在すべて観客および日本ユニバック(株)のデータセンタにて

無事稼動している.

予期した以上の不良箇所を検出でき，点検の成果として評価できると思われる.

ハードウェア・スタビリティ面での効果としては，本技法採用前の IOU トラブルを1とすると採用後は0.7となった.

また，従来よりバックパネル・トラブルは難解なものが多く，とくに間欠的なトラブルについては，机上での追求と再現テスト等で非常に多くの時間を費やすことが多かった. 今回，画一的な検査方法で不良箇所を検出できるようになったことで，フィールドでの長時間のシューティング時間は必要なくなった. IOU トラブルに対するトラブル・サポートのマンパワは本技法の採用後は半減しており，保守効率面での効果が現れている.

6. おわりに

ハードウェアの保守を担当する者として，大きな命題である潜在的不良の検出の分野において，バックパネルという一部分に関してではあるが，その手法を開発することができ，一つの自信が得られた.

今後も，PCA のトラブル，マルチレイヤ・バックパネルといった大きな部分に対して，新たな技法の開発に努力していきたい.

また，今回開発した手法の応用という面からも，MSU への適用，および UNIVAC シリーズ 1100 システム 80 以外のプロダクトへの応用，そして Sperry 社における出荷段階での検査への応用を考えている.

参考文献 [1] 静電気学会，静電気ハンドブック，オーム社，1981.
[2] 電気学会通信教育会，高電圧工学，電気学会，1981.

執筆者紹介 小塩 英造 (Eizo Koshio)
蔵前工業高校卒業，1965年日本ユニバック(株)入社，UNIVAC シリーズ 1110，494 のエンジニアを12年，1100/80 フィールド・サポートを8年，現在 1100/90 サポート1課課長.

中條 幸雄 (Yukio Nakajo)
葛西工業高校卒業，1971年日本ユニバック(株)入社，UNIVAC-Ⅲ のエンジニアを7年，その後 UNIVAC シリーズ 1100 システム 80 のインストレーションとフィールド・サポートを8年，現在 1100/80，1100/90 のフィールド・サポートに従事.

三位　潔 (Kiyoshi Mii)
東京理科大学卒業，1977年日本ユニバック(株)入社，以来UNIVAC シリーズ1100システム80のフィールド・サポート，現在 1100/80，1100/90 のフィールド・サポートに従事．

専門家システム構築ツール KEE

Expert System Development Tool KEE

橋本和博，佐藤公一
—K. Hashimoto, K. Sato—

1. はじめに

KEE (Knowledge Engineering Environment)* は，専門家システムを構築するためのさまざまな支援ツールを統合化したソフトウェアである．

KEE は，医療診断，機器の故障診断あるいは回路設計など，専門家の知識や経験則を使って問題を解決しなければならない分野でその威力を発揮する．

KEE には，次のような各種 AI 技法が組み込まれており，専門家の意思決定プロセスなど幅広い分野に適用できる．

① 専門家の知識を獲得し，知識ベースを構築するための知識表現技法
- 静的な知識を表現するフレーム
- 動的な知識を表現するプロダクション・ルール

② 知識ベースを利用して推論を行うための推論機構（前向き推論/後向き推論）

③ オブジェクト固有の処理を記述するためのオブジェクト指向プログラミング

④ 推論の過程や結論に至った根拠などを説明するための説明機能

⑤ 操作性を向上させるユーザ・インタフェース（メニュ，アクティブ・イメージなど）

KEE が持っているこれらの機能について，以下に概要を述べる．

2. 知識表現

KEE には，静的な知識を表現するフレームと，経験則等の動的な知識を表現するプロダクション・ルールの２種類の知識表現がある．

2.1 フレーム型知識表現

KEE では概念，あるいは対象（オブジェクト）を階層構造化されたユニットで表現する．(KEE ではフレームをユニットと呼ぶ)．

ユニットは複数のスロットから構成され，スロットはその属性を定義する複数のファセットから構成される．

1) ユニット……ユニットは階層構造で表現され，上位の階層は抽象的あるいは一般的な概念を表し，下位にゆくほどより具体的な概念を表す．ユニットには，クラス，サブクラス，メンバ・ユニットがある．

クラス・ユニットは，互いに関連するユニットの集まりを表現するユニットで，図１の例では CREATURE は BIRD, FISH, MAMMAL のクラス・ユニットである．

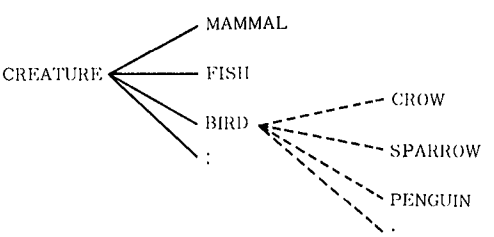
サブクラス・ユニットは，クラス・ユニットに属するユニットをさらに細分類したものを表すユニットで，図１の例では BIRD は CREATURE のサブクラス・ユニットである．メンバ・ユニットは，クラス，あるいはサブクラス・ユニットの具体的な物を表すユニットである．図１の例で，CROW，SPARROW，PENGUIN はそれぞれ BIRD のメンバ・ユニットである．

2) スロット……ユニットの持つ属性は，スロットで表現される．

スロットには，メンバ・スロットとオウン・スロットの２種があり，上位ユニットのメンバ・スロットは下位ユニットに継承される．

したがって，下位のユニットに共通な属性を，その上位ユニットのメンバ・スロットとして持つことによって，その上位ユニットに属するすべての下位のユニットは，そのスロットを引き継ぐことができる．一方，オウン・スロットはユニット固有の属性の表現に使用されるため，下位のユニットには継承されない．

継承方法には，UNION (UNION と指定したスロットおよび，すべての親ユニットの対応するスロットの値の和がそのスロットの値とな



実線：クラス・ユニットとサブクラス・ユニットの接続関係を示す．
点線：クラスあるいはサブクラス・ユニットとメンバ・ユニットの接続関係を示す．

図 1 ユニットの階層構造
Fig. 1 Hierarchy of units

* KEE, ActiveImages, TellAndAsk は，IntelliCorp. 社の登録商標である．

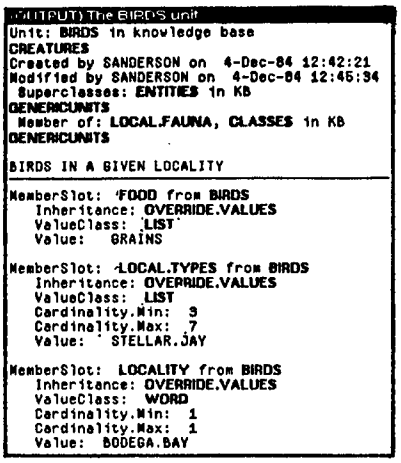


図 2 ユニットの例
Fig. 2 Example of unit

る）など 14 種類の方法が用意されている．

3) ファセット……ファセットはスロットの属性を表す．システムによって，あらかじめ定められている主なファセットには以下のものがある．

- 継承方法 (Inheritance)……スロットの値の継承方法を指定する．
- バリュクラス (Value Class)……スロットの取りうる値の種類を指定する．
- 最大カーディナリティ (Cardinality Max)…スロットの取りうる値の最大数を指定する．
- 最小カーディナリティ (Cardinality Min)…スロットの取りうる値の最小数を指定する．
- バリュ (Value)……スロットの値を指定する．

また，使用者自身のファセットを作成することもできる（図 2）．

2.2 ルール型知識表現

経験則，あるいは論理関係などの推論に関する知識は if-then 形式のルールで表現され，前向き推論や後向き推論で使用される．

1) ルールの形式

(IF 〈条件部〉 THEN 〈結論部〉 DO 〈アクション部〉)

〈条件部〉には，TellAndAsk 式（後述），Lisp 式および任意のテキストあるいは，それらを論理結合子（AND, OR…）で結合した式が書ける．

〈結論部〉には，TellAndAsk 式，任意のテキストあるいはそれらを論理結合子 (AND, OR…) で結合した式が書ける．

〈アクション部〉には，Lisp 式を書くことができる．

```
(IF ((?X IS IN BIRD) AND
     (THE FLY OF ?X IS NO) AND
     (THE SWIM OF ?X IS YES) AND
     (THE COLOR OF ?X IS BLACK, WHITE))
 THEN (THE NAME OF ?X IS PENGUIN))
```

（先頭に？がついた文字列は，変数を表す．）

図 3 ルールの例
Fig. 3 Example of rule

ルールは，その条件部が真であるときに実行される．ルールが実行されると，結論部で宣言された事実が知識ベースに反映され，次にアクション部が実行される（図 3）．

2) ルール・ユニット/ルール・クラス・ユニット……一つのルールは，一つのユニット（ルール・ユニット）に格納される．ルール・ユニットのクラス・ユニットをルール・クラス（ルール・クラス・ユニット）と呼ぶ．

ルールは，ルール・クラスのさらに上位のクラスというように構造化が可能である．推論を始めるときは，ルール・クラスを指定し，そのルール・クラスに属するルールのみが推論に使用される．したがって，問題をいくつかの小さな問題に分割し，それに合わせたルールの構造化が可能である．

3. 推論機構

KEE では，if-then 形式のプロダクション・ルールにもとづいて，前向き/後向きのいずれの推論も行うことができる．

3.1 前向き推論

前向き推論は，与えられた事実をもとに推論を開始する．

まず，与えられた事実と合致する条件を持ち，かつ条件部が真となるようなルールをすべて選び出す．選ばれたルールの集まりを，競合ルール集合 (conflict set) と呼ぶ．

つぎに，この競合ルール集合から一つのルールを選び実行する．競合ルール集合から実行すべきルールを選ぶことを競合解消 (conflict resolution) と呼ぶ．つぎに，実行されたルールの結論部（の一部）と合致する条件を持ち，かつ条件部が真であるようなルールを実行する．

これを繰り返し，適用できるルールがなくなったとき，推論が終了する．

3.1.1 前向き推論の起動

前向き推論は，知識ベース操作用言語 TellAndAsk の ASSERT 命令で，新しい事実や推論に使用するルール・クラスを指定して起動する（図 4）．

一般形式：(ASSERT TellAndAsk 式

ルール・クラス 根拠 知識ベース更新方式)

TellAndAsk 式 ：新しい事実を指定する.

ルール・クラス ：推論に使用するルール・クラスを指定する.

根拠 ：NIL を指定する.

知識ベース更新方式：ルールの結論部で宣言された事実を知識ベースに反映する方法(追加方式, 置換方式)を指定する.

```
(ASSERT '(THE COLOR OF ?X IS BLACK)
        'FORWARD. RULE. CLASS)
```

図 4 ASSERT 命令の例
Fig. 4 Example of ASSERT command

3.1.2 競合解消

競合ルール集合から, 実行すべきルールを一つ選ぶ方法には次のものが用意されており, ルール・クラス・ユニットの特定のスロットで指定する. とくに指定しない場合は, LEAST. PREMISE. COMPLEXITY となっている.

1) LEAST. PREMISE. COMPLEXITY……最も少ない条件を持つルールが選ばれる.
2) GREATEST. PREMISE. COMPLEXITY……最も多い条件を持つルールが選ばれる.
3) WEIGHTED. LEAST. PREMISE. COMPLEXITY……最も少ない条件を持つルールが二つ以上ある場合, ルールの重み (ルール・ユニットの特定のスロットで指定) が一番小さいものが選ばれる.
4) WEIGHTED. GREATEST. PREMISE. COMPLEXITY……最も多い条件を持つルールが二つ以上ある場合, 最も大きい重みのルールが選ばれる.

3.2 後向き推論

後向き推論は, まず与えられた質問 (仮説) が真かどうかについて知識ベースを調べる.

真であれば, 質問が正しいことがわかり推論は終了する.

真でなければ, 質問に合致した結論部を持つルールを探し, それぞれのルールについてルールの条件部を新しい質問として上記を繰り返す. 条件部が真となるルールが見つかったとき, 与えられた質問が正しいことがわかる.

なお, 質問が正しいかどうかが, 知識ベース内の事実からもルールからもわからなかった場合, 使用者に問い合わすことができる.

3.2.1 後向き推論の起動

後向き推論は, TellAndAsk の QUERY 命令で, 仮説や推論で使用するルール・クラスを指定して起動する(図5).

一般形式：(QUERY TellAndAsk 式 検索個数 ルール・クラス 質問指示 探索方法)

TellAndAsk 式 ：質問 (仮説) を指示する.

検索個数 ：質問に合致する例をいくつ求めるかを指定する.

ルール・クラス ：推論に使用するルール・クラス・ユニットを指定する.

質問指示 ：NIL 以外を指定すると, 条件が真かどうか知識ベースからもルールからもわからなかった場合, 使用者に真かどうかたずねる.

探索方法 ：後述

```
(QUERY '(THE FOOD OF PENGUINE IS ?X)
       'ALL
       'BACKWARD. RULE. CLASS
       T)
```

図 5 QUERY 命令の例
Fig. 5 Example of QUERY command

3.2.2 探索方法

探索方法には,

1) 縦型探索 (DEPTH. FIRST. SEARCH)
2) 横型探索 (BREADTH. FIRST. SEARCH)
3) 最良優先探索 (BEST. FIRST. SEARCH)

が用意されており, ルール・クラス・ユニットの特定のスロットで指定する. とくに指定しない場合は, 縦型探索 (DEPTH. FIRST. SEARCH) となっている.

最良優先探索では, 最も少ない条件を持つルールから調べられる.

3.2.3 ルールの適用順序

質問に合致する結論部を持つルールを取り出す順序には次のものがあり, ルール・ユニットの特定のスロットで指定する. 何も指定しない場合は, DEFAULT. BC. ORDER となっている.

1) BC. LEAST. COMPLEXITY……最も少ない条件を持つルールから取り上げる.
2) BC. GREATEST. COMPLEXITY……最も多い条件を持つルールから取り上げる.
3) BC. LEAST. WEIGHT……最も小さい重みのルールから取り上げる. ルールの重みは, 各ルール・ユニットの特定のスロットに指定する.
4) BC. GREATEST. WEIGHT……最も大きい重みのルールから取り上げる.
5) DEFAULT. BC. ORDER……ルールが作

られた順で取り上げる．

4. メソッドとアクティブ・バリュー

4.1 メソッド

KEE における知識表現を2章で述べたが，この他フレーム型知識表現の一種として，オブジェクト指向型プログラミング環境が提供されている．すなわち，一つのオブジェクトを一つのユニットとして表現し，そのユニットに固有の手続きを付加（スロット値に Lisp コードを指定）することができる．KEE では，このスロットに指定された Lisp コードのことをメソッドと呼ぶ．メソッドには，次のような特徴がある．

- ユニットに対して手続き的知識を付加できる．
- スロットとして指定されるので継承が行える．

1) メソッドの継承……メソッドは，親ユニットのメソッドをそのまま継承するだけでなく，さらに独自の Lisp コードを追加することができる．

たとえば，

CARS……LARGE. CARS
CARS……SMALL. CARS

という知識があり，CARS のメンバ・スロットとして MSG. METHOD があり，その中に次のメソッドがあるとする．

```
(LAMBDA (THISUNIT)
  (PRINT "A car is nice."))        …①
```

次に LARGE. CARS の MSG. METHOD スロットに，以下のようなコードを入れたとする．

```
((AFTER (PRINT "This car is, large.")))
                                   …②
```

これにより，LARGE. CARS の MSG. METHOD スロットのメソッドを起動すると

```
A car is nice.
This car is large.
```

の2行がプリントされる．すなわち，②の AFTER はメソッドのキー・ワードであり，親ユニットのメソッドを継承して，その Lisp コードを実行した後，AFTER 以降に書かれた Lisp コードを実行することを意味している．これを後置 Lisp コードと呼ぶ．AFTER と同様のキー・ワードとして，BEFORE（前置 Lisp コード），WRAPPER（間置 Lisp コード）がある．

2) メソッドの起動……メッセージ送出という形で起動することができる．それには，6.2 節で述べるコマンド・メニュの中の Send Message コマンド，あるいは UNITMSG 関数を用いて行う．UNITMSG の形式は次のとおりである．

```
(UNITMSG 〈ユニット名〉〈スロット名〉
  {〈引き数〉}…)
```

4.2 アクティブ・バリュー

スロットの値が変更されるとき，あるいは参照されるとき，Lisp 手続きを自動的に実行してそのスロット値へのアクセスを監視することができる．（一般にはデモン機能と呼ばれている．）KEE では，このような Lisp 手続きを特別なユニットのスロット中に記述する（すなわち，メソッドとして記述する．）この特別なユニットのことをアクティブ・バリュー・ユニット，または単にアクティブ・バリューという．アクティブ・バリューの特定のスロットに Lisp コードを書くことにより，上記の機能を実現できる．代表的な二つのスロットを次に説明する．

1) AVPUT スロット……スロットの値が変更されるとき，ここに書かれた手続きが起動される．
2) AVGET スロット……スロットの値が参照されるとき，ここに書かれた手続きが起動される．

次に監視されるスロットとアクティブ・バリューの概観を図6で説明する．

ここで，MONITORED. SLOT の値を変更しようとすると

```
invoke AVPUT
```

が印書され，参照しようとすると

```
invoke AVGET
```

が印書される．

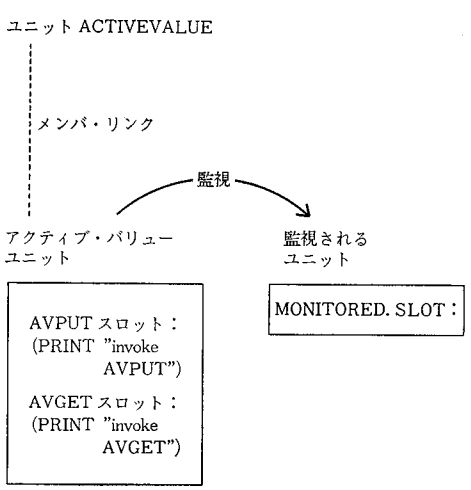


図 6 アクティブ・バリュー
Fig. 6 Active-value

5. 説明／デバッグ機能

5.1 説明機能

前向きおよび後向き推論で，「どのようにして，その結果が得られたのか？」という質問をシステムに行うことができる．また，後向き推論の場合には，条件の真偽をシステムが使用者に尋ねることがあるが，「なぜ，そのようなことを尋ねるのか？」という質問を行うこともできる．このような質問に対する応答を説明と呼ぶ．

質問の方法は前者の場合には，前向き推論，または後向き推論が終わった後，

(UNITMSG 'ルール・クラス・ユニット
'HOW. FORWARD. CHAIN)

または

(UNITMSG 'ルール・クラス・ユニット
'HOW. BACKWARD. CHAIN)

のようにメッセージをルール・クラス・ユニットに対して送出すると，その結論に至った一連のルールがグラフ表示される．

また，後者の場合にはシステムからの問い合わせに対して

WHY

と応答することにより，その説明をさせることができる．

5.2 デバッグ機能

デバッグ機能としてルールのトレース，実行の中断/続行などの機能が用意されている．

5.2.1 ルール・グラフ

ルール・クラスに属する各ルールの静的な相関関係をグラフで表示する．次に示すようにルール・クラス・ユニットにメッセージを送出することにより表示できる．

(UNITMSG 'ルール・クラス・ユニット
'GRAPH. RULES)

5.2.2 トレース

ルールのトレースとして次の四つがある．トレースを表示するには該当のスロット（以下の四つが，それぞれスロット名でもある．）の値をオンにすればよい．

1) TEXT. TRACE……推論の過程を文章で表示する．
2) OR. TRACE……後向き推論用のトレースであり，それまでの推論過程全体をグラフで表示する．
3) AND. TRACE……後向き推論用のトレースで，一つの導出に関する中間結果をグラフで表示する．
4) FC. TRACE……前向き推論用のトレースで，それまでの推論過程全体をグラフで表示する．

FC. TRACE の例を図7に示す．

この他のデバッグ機能として，

① 一つのルールごとに実行中断するステッパ・モード
② 指定したルールを実行中断するルールのブレーク

などがある．

6. ユーザ・インタフェース

6.1 TellAndAsk

TellAndAsk は，知識ベースの操作のための言語であり，ユニット，スロットの作成・更新・削除などを行うことができる．次に TellAndAsk の形式を示す．

(〈tellandask 関数〉〈tellandask 式〉
〈その上の引き数〉)

〈tellandask 関数〉は，知識ベースの操作内容を示す関数である．

〈tellandask 式〉には，知識ベースに登録したい事実，検索したい内容，削除したい事実を書く．これには，英語に似た形式(NL-form)，前置形式(prefix-form) の二つの形式がある．

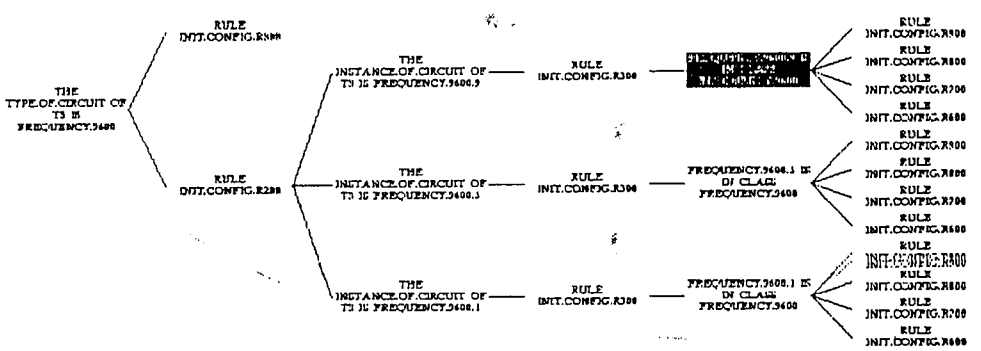


図7 FC. TRACE の例
Fig. 7 Example of FC. TRACE

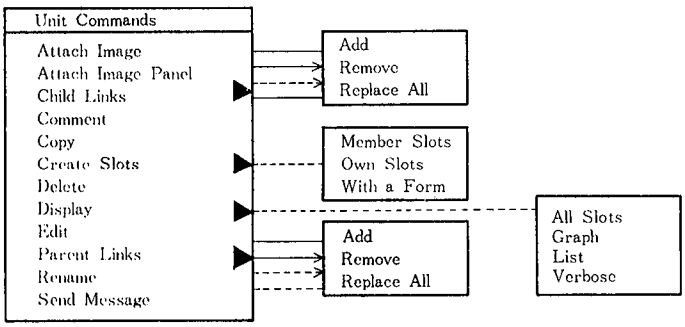
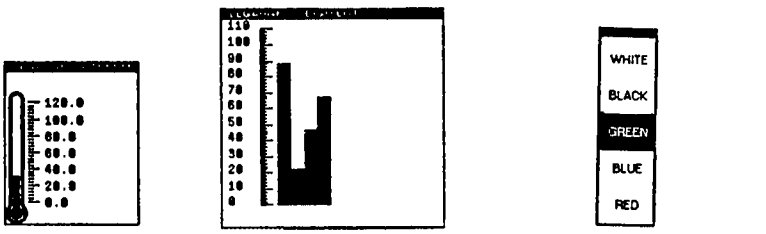


図 8 ユニット・レベル・メニュ
Fig. 8 Unit level menue



THERMOMETER HISTOGRAM VERTICAL.TRAFFIC.LIGHT

図 9 アクティブ・イメージの例
Fig. 9 Example of ActiveImages

〈その他の引き数〉は，関数により異なる．

TellAndAsk は，Lisp リスナ・ウインドゥから入力すること，およびメソッドの中に記述することができる．また，TellAndAsk 式は，ルールの条件部と結論部に書くこともできる．(2.2 ルール型知識表現の項参照)

また，TellAndAsk により前向きおよび後向きの推論を開始することができる．(3. 推論機構の章参照)

次に TellAndAsk を用いた知識ベースの更新とと検索の例を示す．ある知識ベースに MY. CAR ユニットがあり，その中に CAR. NAME スロットがあり，その値が SUNNY であるとする．そこで，

```
(ASSERT '(THE CAR. NAME OF MY. CAR
                          IS BLUEBIRD))
   ↑                  ↑
  関数          tellandask 式
(事実の宣言)
```

を実行すると CAR. NAME スロットの値は，BLUEBIRD になる．

次に，

```
(QUERY '(THE CAR NAME OF MY. CAR
                          IS ? CAR) 1)
   ↑                              ↑
 関数(検索)                    検索個数
```

を実行すると

(THE CAR. NAME OF MY. CAR IS BLUEBIRD)

という値が返される．

6.2 メニュ

知識ベースに関する操作には，TellAndAsk で行う以外にマウスとメニュを用いる方法があり，知識ベース，ユニット，スロットの作成・更新・削除・編集などが行える．図 8 に代表的なメニュを示す．

6.3 アクティブ・イメージ

スロット値をグラフィックを用いて表示や変更するものとしてアクティブ・イメージがある．アクティブ・イメージには，KEE があらかじめ用意している標準のイメージがいくつかある(図 9)．

アクティブ・イメージは，アクティブ・バリューの機能を使って実現されている．すなわち，スロットの値が変更されるとそれに連動してイメージ上の表示も変更される．また，いくつかのアクティブ・イメージは，マウス操作でイメージ表示を変えることによって知識ベース内のスロット値を変えることができる．

アクティブ・イメージには，スロット値を表示するものの他に，ユニットを表示するもの，および関連するイメージをまとめて表示するためのイメージ・パネルとがある．

7. おわりに

本稿では KEE の多彩な機能のうちの一端を紹介した．AI ビジネスも緒についたばかりで，これはという実用化システムは数少ない．実用的専門家システム構築を目指すための道具として，KEE は有効でかつ強力であると考えている．

参考文献

[1] M. J. Coombs, *Development in expert system*, Academic Press, 1984.

[2] R. E. Fikes and T. P. Kehler, "The Role of Frame-Based Representation in Reasoning", Communications of the ACM, Sept. 1985. (邦訳)知識ベースを階層化し推論を制御するフレーム・システム，日経エレクトロニクス，1986. 3. 10.

[3] 上野晴樹，知識工学入門，オーム社，1985.

[4] 白井良明，辻井潤一，人工知能，岩波書店，1984.

(知識システム開発部)

事務文書体系と メッセージ指向文章交換系

An Orientation for Office Document Architecture and Message Oriented Text Interchange Systems

若 鳥 陸 夫

R. Wakatori

1. はじめに

開放型システム間相互接続（略称：OSI）の参照モデルの思想の拡がりにつれて，異系統間の情報交換のシーズが熟成されてきた．現在，普及している従来形の統合通信というのは，同一系統内から直接異種端末装置類(例テレテックスやファクシミリ)を用いて情報交換を行うという形態であって，けして世界水準で異系統間を経由して，異種端末装置類と情報交換できる水準ではない．しかし，活動の広域化や即時性の要求の高まりにつれて，特定メーカの提供する「限定範囲付の統合化通信」では，将来の要求に答えられないことが明白になり，1981年春，世界の電算機本体製造者・通信装置製造者・端末装置製造者・通信業者が Ottawa (Canada) に会合し，異系統間で事務文書交換するための国際規格の制定準備にとりかかった．その会合は，国際標準化機構/TC 97/SC 18，"文章およびオフィス・システム"」の第1回総会であった．その後，1年ごとに，1982年 London (England)，1983年 Paris (France) 1984年 Berlin (West Germany)，1985年 Washington D. C. (U. S. A)，1986年東京と，総会が開かれてきた．この6年間の国際標準化機構の活動の結果「メッセージ指向文章交換系」や「事務文書体系」の国際規格案が作成された．これらの国際規格案は6か月間の国際投票の手続を経て，賛成国多数となれば国際規格となる．また，国際規格が制定されると，それを日本語に翻訳して日本工業規格として制定される見通しである[1]．

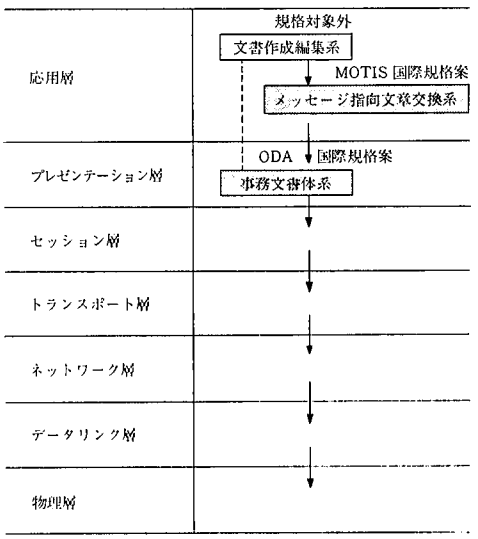


図 1 事務文書体系とメッセージ指向文章交換系の OSI における位置

Fig. 1 Position of office document architecture and message oriented text interchange systems mapped on open systems interconnection reference model

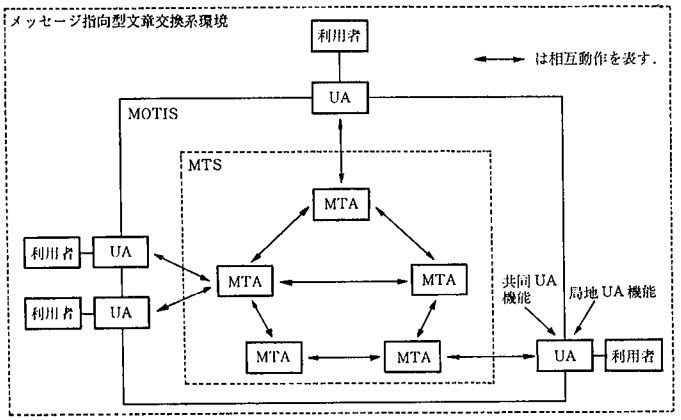
事務文書体系(略称 ODA)およびメッセージ指向文章交換系（略称 MOTIS）は，開放型システム間相互接続（略称 OSI）の最初の応用である．このうち事務文書体系は，交換対象とする事務文書の構造の「規則の集合」であり，開放型システム間相互接続の「プロレゼンテーション層」の規格である．また，メッセージ指向文章交換系は，「応用層」の規格である（図1）．

応用層の MOTIS の封筒内部のデータ形式は，プレゼンテーション層の事務文書体系を写した形となる．

現時点では，国際規格文章そのものは編集修正の途上にあるが，その内容面は概して固定しているので，その内容をここで概説する．

2. メッセージ指向文章交換系（略称 MOTIS：モーティスと発声）

メッセージ指向文章交換系 (Message Oriented Text Interchange System) は，端的にいうと国際通信諮問委員会で検討されていたメッセージ搬送



MTA：メッセージ転送代行，UA：利用者代行

図 2 MOTIS モデルの機能図
Fig. 2 Model for MOTIS

図 3 メッセージの基本構造
Fig. 3 Conceptual structure of message

系（略称 MHS）に，国際標準化機構で利用者側からの要求を加えて拡張し，国際規格としての体裁を整えたものである．その系統図は，図2のように，国際通信系統内に「メッセージ転送代行（MTA)」が複数あり，利用者の近くに「利用者代行（UA)」がある．利用者代行者は，通信端末装置でもよいし，コンピュータ本体であってもよい．メッセージ指向文章交換系のサービス種目は，「個人間メッセージ転送サービス」や「メッセージ転送サービス」などが国際規格案として表面に出てきている．このメッセージ指向文章交換系の形態は蓄積交換方式である．この形態は，現在のディジタル・データ交換網にその原点をみることができる．このメッセージ指向文章交換系のデータ様式は，図3のように封筒部と内容部からなり，内容部としてテレックス・テレテックス・ファクシミリに加えて，事務文書交換様式によるものを取り扱う．

3. 事務文書体系

事務文書系体（Office Document Architecture）は，メッセージ指向文章交換系を利用した応用の一つであって，文書概要，論理構造，および割付け構造からなる（図4）．

3.1 文書概要

文書概要(Document Profile)は，文書全体の概要を記述した部分で，最小限，表現能力と文書管理属性を記述する．この表現能力の記述は，文書概要水準，文書体系水準，内容形式（文字内容・ラスタ図形内容・幾何学図形内容），などが必須記述項目である．日本国内での応用では，その他に非基本表現能力として，情報交換用漢字集合を指定するエスケープ・シーケンスなどを記述する．非基本表現能力としては，その他に代替文字集合・ページ寸法・符号化属性・表現属性・書体一覧表（一度に10種類まで指定可能）などがある．

文書管理属性は，表題・日時・原作者・他利用者情報・外部参照・ファイルおよび検索・内容属性・機密属性がある．これらの文書管理属性のうち，表題を欠くことはできない．

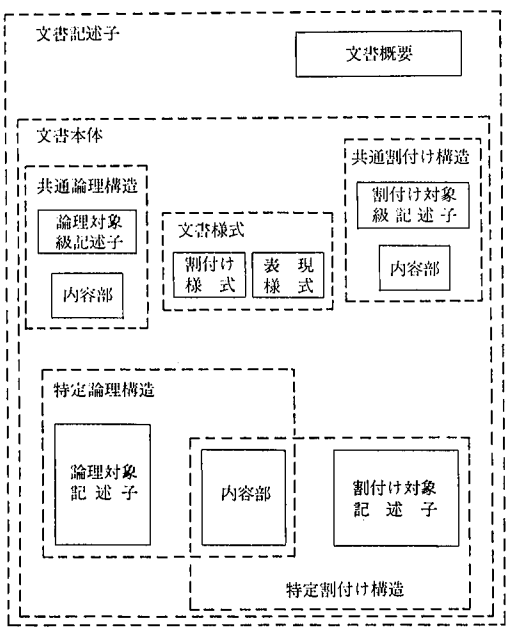


図 4 文書の要素
Fig. 4 Element of document

3.2 論理構造

論理構造は，文書・章・段落・図・表などの文書の内容的な構造であり，原稿用紙に書いた文章のようなもので，一般的に最終的な割り付けがなされる以前の形態をしている．これらの最下位構造に内容部があるときと，ないときがある．

3.3 割付け構造

割付け構造は，文書割付け根・ページ集合・ページ・枠・区の単位に分割された構造で，再生する用紙のページへ割り付けされた構造を表す．割付け構造には共通割付け構造と特定割付け構造がある．共通割付け構造は，業務書簡などに対して共通に定めた割付け構造を示し，特定割付け構造は，ある特定の書簡を割付けした構造を示している．(図5)．

3.4 文書処理モデル

「文書処理」は，「編集処理」，「割付け処理」，「可視化処理」の3段階を想定している．編集処理は，論理構造や内容部の生成および割付け構造を生成・修正する．「割付け処理」は，論理構造を割付け構造に変換する(図6)．「可視化処理」は，機械可読状態を人間可読状態へ変換する処理をいう．たとえば，書式制御符号により最終出力の書式を整えたり，区の位置や寸法により紙面に展開して，割り付ける操作などを行う．この「可視化処理」は文書生成編集機の印書処理と類似している．

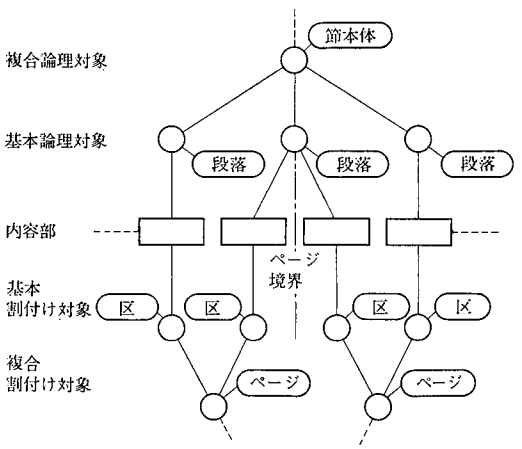


図 5 論理対象と割付け対象の関係
Fig. 5 Logical object vs. layout object

3.5 内容体系

割付け対象の区は「基本割付け対象」と呼ばれ，概して内容部が付属する．「内容部」は，符号種類や情報内容部からなり，情報内容部は「内容体系」からなる．内容体系は文字内容体系・ラスタ図形内容体系・幾何学的図形内容体系・音声内容体系が検討されている．このうち，基本的な文字内容体系は，文字列と機械符号群の組み合せからなり，日本の情報交換用漢字符号系による横書き・縦書きも支援している．割付け処理は，基本割付け対象である区に対して行う他に，区の集合である枠を割付け対象としたり，ページやページ集合を割付け対象としたりできる．基本割付け対象である区が，1ページの大きさである特別の様式は，従来からのコンピュータ間での文書交換の様式に類似している．

3.6 文書様式

文書体系には，論理構造と割付け構造の他に，文書様式がある．文書様式は，論理構造を自動割付けされる際に参照される「割付け様式」，および割付け構造を可視化する際に参照される「表現様式」がある．

3.7 原作者・編集者・割付け者の意図の伝達

論理要素や割付け要素などの性質は属性として記述し伝達する．その属性の種類は，論理要素と割付け要素の両方に適用する「共通要素」，割付け要素

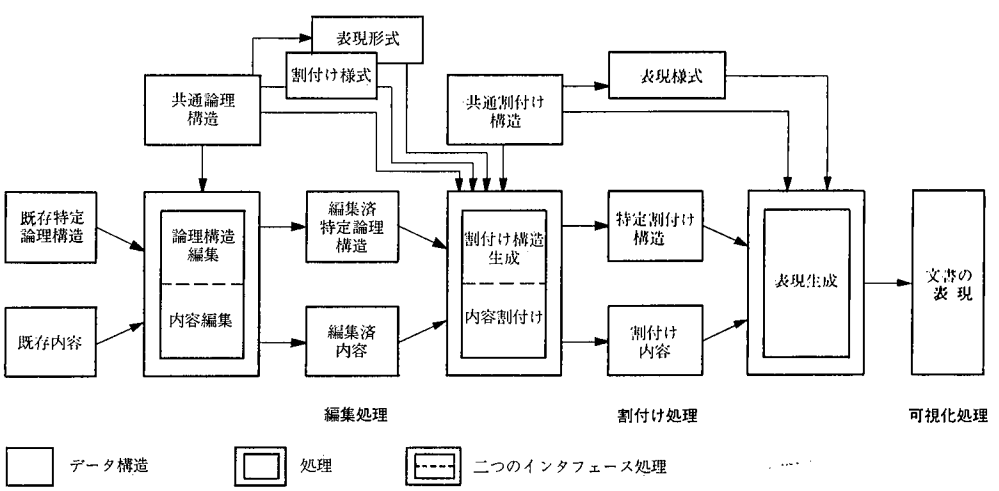
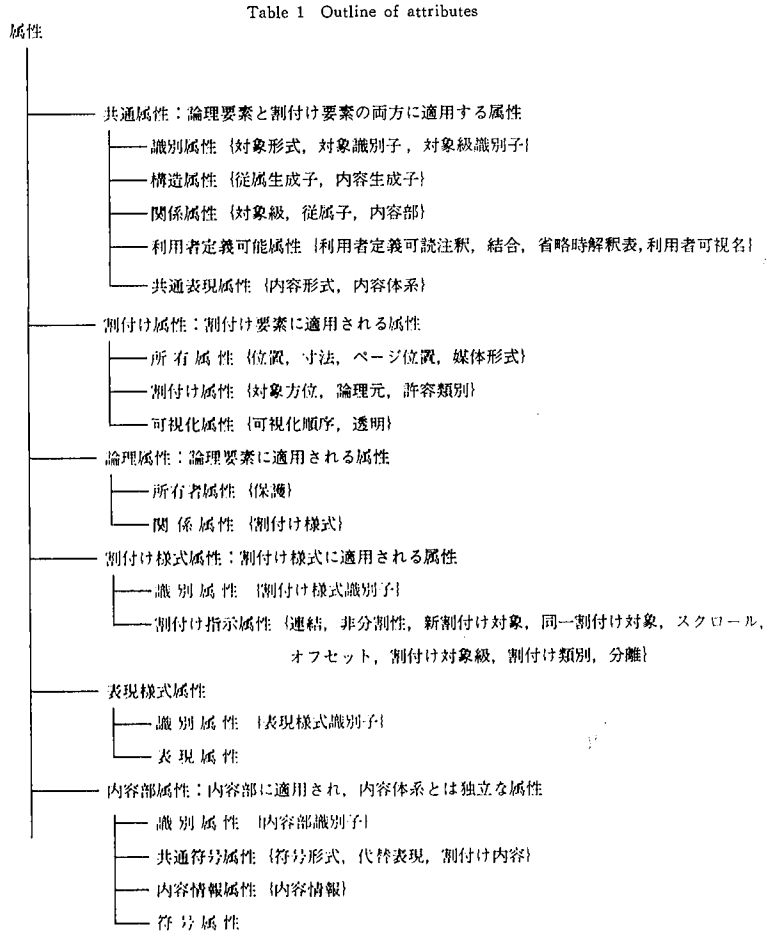


図 6 文書モデル
Fig. 6 Document processing model

表 1 属性一覧表
Table 1 Outline of attributes



だけに適用される「割付け属性」，論理要素だけに適用される「論理属性」，割付け様式に適用される「割付け属性」，表現様式に適用される「表現様式属性」，内容部に適用され，内容体系とは独立な「内容部属性」がある．これらの属性の内訳の名称だけを抜萃すると，表1のとおりである．

たとえば，論理構造を生成した際に定めた対象識別子やその子孫を示す従属子は，割付け処理された後にも同様である性質があるから，共通属性である．また，割付け対象個有の位置や大きさなどの所有属性は，割付けされた対象にだけ効力を有するから，割付け属性になっている．この割付け位置と割付け寸法が割付け対象ごとに与えられるので，マウスと窓機構向きのデータ構造になっている[2]．

4. 交換文書の構成

文書は，書式付き形式，処理可能形式，書式付き処理可能形式の三つの形式の一つで交換できる．

各形式に含まれる情報を表2に示す．

4.1 書式付き形式

表 2 文書形式に含まれる情報
Table 1 Information for document format

文書形式	情報
書式付き形式	文書概要，特定割付け構造，書式付き形式の内容，割付け対象級記述子，表現形式
処理可能形式	文書概要，特定論理構造，内容，論理対象級記述子，割付け対象級記述子，表現形式，割付け形式
書式付き処理可能形式	文書概要，特定論理構造，特定割付け構造，内容，論理対象級記述子，割付け対象級記述子，表現形式，割付け形式

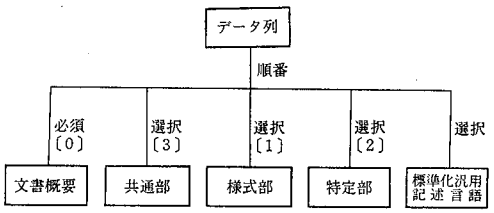
書式付き形式の文書とは，割付け処理された後の文書で，可視化処理さえすれば人間の眼に見える形にできる形式である．

この形式の文書の再編集は，人間がすべてやるより方法がなく，機械処理で自動的に再編集することを意図されていない．日本語文書生成編集機の磁気記録媒体内のデータと類似している．

4.2 処理可能形式

論理構造を有しており，割付け形式などにより機械で何度でも割付け処理ができる．

原稿用紙の内容のような論理構造と内容部があ



注：番号は識別符号を表す．

図 7 データ列の構成
Fig. 7 Structure of data stream

り，それを書籍の体裁に合わせて割付け処理できる形式と考えればわかりやすい．

4.3 書式付き処理可能形式

これは，書式付き形式と処理可能形式の双方を持ち合せた形式で，ここでは一番高水準の形式である．

5. 文書交換様式

メッセージ指向文章交換系 (MOTIS) や記憶媒体との間で授受する封筒内部のデータ列は，順番が定まっており，文書概要・共通部・様式部・特定部・標準化汎用記述言語部の順とする．文書概要以外は，任意選択であり，交換の必要性がなければなくてよい（図7）．

データ列の構成の種類は，データ列組成 A1, A2, Bの3通りが考えられている．その一つのデータ列Aは，文書概要，論理構造，割付け構造，および標準化汎用記述言語部からなる．データ列組成Aは，さらに標準化汎用記述言語部の有無により A1 と A2 に区別される．

データ列Bは，論理構造か割付け構造のいずれか片方しか含有しないデータ列のことを示す．

これらのデータ列は，識別子，長さ記述子，内容の形式で入れ子構造をしている．この識別子は，データ列Bなら "0A"，文書概要なら "80"，などを記述する．長さは，子孫の識別子も含めた8ビットバイト数で符号なしで2進数，または長さバイト数と長さの値で表される．これらの機械向き表現は，アセンブラ言語に類似した変換プログラムや構造化エディタによることになろう．データ列の構文の記述法は，抽象構文記法1（略称 ASN. 1）によることになっている．

6. 適合水準

事務文書体系は，広範囲の事務文書を対象とするために，多段階の適合水準を定めている．それらの水準はテレックス相当→テレテックス相当→文書作成編集機相当→デスクトップ印刷相当→インハウス印刷相当などにわたる．適合水準の分け方は，文書形式・文書概要水準・文書体系水準・内容形式・機能符号集合範囲などで影響を受ける．このうち，文書形式により，文字内容体系内の機能と機能符号集合範囲が段階別に定められている．

たとえば，一番下位水準のテレックス相当では，書式付き形式・テレックス文字集合（英字主体）・機能符号3種（復帰・改行・間隔），文書表題，といったものになる．

高位水準では，一度に字体が10種類も選択できることや，可変文字間隔，ラスタ図形内容体系と文字内容体系との混合使用などもできることから，事務文書のかなりの範囲を対象とできよう．

7. おわりに

次世代の事務文書交換の一端を紹介した．この文書体系と従来の文書体系との違いや特徴を次にまとめる．

1) 事務文書体系は，開放形システム間で事務文書を交換する体系である．テレックス，テレテックス，ファクシミリ，パーソナルコンピュータ，文書生成編集機など異系統間の文書交換を意図している．
2) ファクシミリなどへの割り付け情報を含有する．ドット単位（正確には 1 BMU＝25.4 ミリ/1200）で割り付け位置を交換する．
3) メッセージ指向文章交換系は，当初蓄積交換形式であり，「個人間メッセージ交換サービス」や「メッセージ転送サービス」などから開始される．MHS は，MOTIS の部分集合である．
4) 文書体系は，文書の論理要素と割付け要素に分け，論理要素をある様式で割り付けると，割付け構造になる．

最後に，日頃，有益な助言をいただく ISC/TC 97/SC 18 英・米・仏・独・蘭・加の委員，情報処理学会 SC 18 専門委員会委員，並びに SC 18/WG3･5 合同委員会委員，日本規格協会テキスト交換システム調査研究委員会委員の各位に謝意を表する．

参考文献

[1] (財)日本規格協会情報技術標準化研究センター，システムソフトウェアの標準化に関する調査研究（テキスト交換システム）報告書，1986年3月．
[2] 若鳥陸夫，"マウスと整合性のよい事務文書"，Information, Vol. 5, No. 5～7, 1986.
[3] ISO, *DIS 8613/1, Information Processing-Text and Office Systems-Office Document Architecture (ODA) and Interchange Format-Part 1: General introduction* (第1次案) June 13, 1986.
[4] ISO, *DIS 8613/2, Information Processing-*

Text and Office Systems-Office Document Architecture (ODA) and Interchange Format-Part 2: Document Structures (第1次案), June 1986.

[5] ISO, *DIS 8613/3, Information Processing-Text and Office Systems-Office Document Architecture (ODA) and Interchange Format-Part 3: Document Processing Reference Model* (第1次案), May 1986.

[6] ISO, *DIS 8613/5, Information Processing-Text and Office Systems-Office Document Architecture (ODA) and Interchange Format-Part 5: Office Document Interchange Format (ODIF)*, (第1次案), June 1986.

[7] ISO, *DIS 8613/6* (第1次案), *Information Processing-Text and Office Systems-Office Document Architecture (ODA) and Interchange Format Part 6: Character Content Architectures* (第1次案), April 1986.

(技術研究部)

インテリジェントビルの動向

Current State of Intelligent Building

大嶋重光

S. Oshima

1. はじめに

インテリジェントビルが米国に出現した背景として，情報・通信技術の革新，AT&T 社の分割による通信事業の自由化促進，オフィスの快適性・生産性向上の実現要求，ビルの省エネルギ指向などがあげられる．この他に，地価の安い米国では現在，投資の対象や税金対策としてのビル建設ブームを迎えており，Dallas 市にその典型が見られるようにビル建設ラッシュであること，大都市では都心部のスラム化，交通渋滞の激化，情報・通信インフラストラクチュア建設のデッドロック化が深刻化していることなどがある．そこで，これらの問題解決の一環として，**都心部のオフィスビルの高層化・高機能化と，オフィス地域のさら地への移転構想**があいまって，インテリジェントビル出現をもたらした．

インテリジェントビルの一つの方向は，高層化・高付加価値化によって競争力を強めた豪壮華麗なインテリジェントビルであり，もう一つの方向はテレポート建設の一環としてのインテリジェント化されたオフィスビル群である．

インテリジェントビルという語はくしくも Hartford 市 (Connecticut 州) の中心部に建てられた City Place と，New York Teleport とから別々に発生したものであるが，一般的には都心型の City Place のほうがよく知られている．

ビルの建設に際し，エレベータ，空調システム，エネルギ供給，防犯・防災など，従来個別に制御されていたシステムに，United Technologies Building System 社が情報処理・情報通信システムを統合化し，1983年11月 Hartford 市に City Place の名前で竣工させたビルが最初のインテリジェントビルとされている．

これが引き金となり，インテリジェントビルと銘打ったビルが米国の主要都市で陸続と建設されており，前出の City Place を初めとして AT&T 本社，One Financial Place，LTV Center，Lincoln Plaza など主要なものだけでも 40 棟を数えている．これらは，いずれも都心再開発超高層型のオフィスビル主体であり，その多くはテナント・オフィスビルである．

一方わが国では，先の City Place の完成に触発されて一昨年の後半から昨年にかけて，米国への視察ラッシュが続く一方，東芝本社ビルや，この6月オープンしたばかりでただちに有名になっているアークヒルズなどがインテリジェントビルと命名されており，その数約30棟が数えられている．米国のインテリジェントビルがテナント・オフィスビル指向であるのに対し，わが国では自社ビル主体であることが特徴的と言える．また，わが国では都心型および地域開発型双方を対象としており，とくに後者についてはこれを単にビル単体のインテリジェント化から，ビル間やビル群のインテリジェント化として捕え，これにインテリジェント・コンプレックス，さらにはインテリジェント・シティなどの用語が与えられている．

対象分野がビルやビル・コンプレックス（都市と言ってもよい）の建設であるから，それを支える学術とエンジニアリングの範囲は広大であり，インテリジェントビルの構築に関し部分的言及が散見されるだけである．

したがって本稿では，現在いわゆるインテリジェントビルとして論じられている事柄の実体は何かについて整理するとともに，今後の技術とエンジニアリングの課題について述べてみたい．

2. インテリジェントビルの概要

ビルがインテリジェント化されているという場合，一般にその特徴として次の事柄が指摘されている．

1) 通信ネットワークをビル内に張り巡らし，ディジタル交換機，コンピュータ，多機能電話機，多機能ターミナルなど各種OA機器を導入し，さらには外部ネットワークとの接続による情報のスピーディな検索など，オフィス業務の総合的な支援を行い，オフィスの生産性の向上を図ることができるビルであること．

2) ビルの各所に取り付けられたセンサと各フロアや中央の制御用コンピュータによって，空調，照明，エレベータ，エネルギ供給などの制御を一元的に行い，省エネルギおよび快適で安全なビル環境の向上を図りうるビルであること．

3) ビル内の最適照明，採光，音，空調，レイアウト，さらには休憩室やフィットネス（健康増進）センタの採用などによって，そこで働く人人の健康にも十分配慮が行き届いたビルであること．

このようなインテリジェントビルの特徴からも明らかなように，その構成要素となるシステム系としては，①ビル利用者のための**情報処理・情報通信系システム**と，②ビル設備・機械装置の**制御系システム**，および③現状ではビジネス用オフィスがインテリジェントビル需要の過半を占めていることから，業務の生産性および執務の快適性の観点でより良いオフィス環境を実現するための**オフィス環境系のシステム**が必要となる．

ビルのインテリジェント化（高機能化）が促進されると，ビルのオーナおよびユーザの双方にとって，その機能をより効果的に発揮させるための知識や技術ノウハウが必要となってくる．そこに**インテリジェントビル・サービス**が専業化する余地が生じてくる．米国における“シェアード・テナント・サービス業”の台頭がその典型がある．

わが国ではNTT社がその新規事業展開の一つとして注力しているほか，アークヒルズや大阪に建設中の梅田センタービルにおいて実現されようとしている．

以上のことから，現時点ではインテリジェントビル市場は，これを次のように捕えることができる．

1) 今後急速に拡大することが予測されている**オフィスのインテリジェント化**を担い，インテリジェント化に必要なソリューション・ツール（システムおよびプロダクト）を提供する．

2) **インテリジェント化されたビルの運用・管理（ビル・サービス）**業務を担い，各種サービスを提供する．

本来，供給過剰に陥った米国のオフィスビル市場において，いかにしてテナント獲得戦を有利に展開するかの方策として打ち出されたのがインテリジェントビル戦略であったが，都心部の地価の高騰が依然として進行しており，オフィスビルの供給不足下にあるわが国では，インテリジェントビル・ブームの底流にある機能・サービス・効用について，より地道に発展させようとする兆しが現れている．そして，その動きはシステム（プロダクト）面での技術的統合化，およびインテリジェント・コンプレックス化となって反映されている．

1) システム（プロダクト）の統合化……ビルのインテリジェント化に必要なシステム系のうち，情報処理・情報通信系システム，とくにビル内とビル間の通信系システムが基幹的役割を占めている．このことによってプロダクト面で見る限り，インテリジェントビルは**テレコミュニケーション分野の一部もしくは導入部**に位置付けることができる．テナントビルのように不特定多数のユーザがアクセスする環境下では，より共用性（インターオペラビリティ）の高いシステムの提供の可否がシステム成功のポイントとなる．

2) インテリジェント・コンプレックス化……建設省が発表した「インテリジェント・コンプレックス」構想に代表されるように，すでにインテリジェントビル市場は“ビル内”から“ビル間”や“ビル群”へ，すなわち点から線・面への展開を示しはじめており，地域開発や都市部再開発の一貫として位置付けられている．建設主体も，一部民間のディベロッパ主導のものもあるが，多くは政府や地方自治体の指導・提唱による公共性の強いプロジェクトが続々と出現している．

3. インテリジェントビル・システム

インテリジェントビルでは，次のような諸要件が満足されていなければならない．まず第1に，オフィスで働く人々が快適で働きやすく，かつ生産性の高い仕事ができ，仕事それ自体から満足が得られるようになること．第2には，情報の重要性が今後ますます高まっていくビジネス環境の変化に，企業が先進的かつ柔軟に対応できること．つまりテレコミュニケーションの普及，ニューメディア化の進展，さらにはインテリジェント・コンプレックスにおける情報拠点としての主導的役割を果たしうること．第3に，ビル設備の運営・管理のオートメーション

化が進み，省エネルギ化の増進によるビル管理コストの低減が実現されること．そして，最後にオフィスに働く人々やビル資産のセキュリティが増進されることである．以下では，日本ユニバックの IBE (Intelligent Building Engineering) を例として話を進めてゆく．

このようなインテリジェント化が実現されるためには，次のようなサービス機能が果たされている必要があり，それらは概略次の七つのカテゴリに要約される．

1) 音声・データ・画像など多様なメディアを柔軟かつ低コストで利用できるようにする**通信サービス**
2) データ処理やデータベース・アクセスを行う**情報処理サービス**
3) オフィス業務の生産性を向上させる**統合化 OA サービス**
4) ビル・ファシリティの機能をより有効に発揮させるための**ファシリティ・マネージメント・サービス**
5) オフィスに働く人々や企業のオフィスにまつわる間接業務を代行する**オフィス・サポート・サービス**
6) オフィスに働く人々やビル資産の安全，防災・防犯のための**セキュリティ・サービス**
7) ビルや設備を良好に維持・管理するための**ビル・メンテナンス・サービス**

そしてこれらのサービスを支援するため，ビルにはさまざまな設備・機器，システムが採用される訳であるが，それらは目的と用途に応じて次の五つのロジカル・システムに集約される．

1) 情報処理・統合 OA システム……データ処理の迅速・正確化と意思決定支援
2) 通信システム……情報の収集・配布に際しての距離と時間の制約の克服
3) オフィス環境システム……執務環境と業務の生産性との調和
4) ビル・セキュリティ・システム……人とビル資産の保全
5) ビル・オートメーション・システム……主に省マンパワや省エネルギ

これらのシステムはインプリメンテーションの段階において，まったく統合されていることも，反対にまったく独立して採用されることもまれである．たとえば，ビル内の一部の配線を情報系と通信系とが共用し合っているのが現状である．今後は，従来独立している，とくに情報・通信系とビル制御系間でのビル基幹ケーブル・システムの統合化が技術的課題となろう．これまでビル管理システムとして，扱われていた分野を，それぞれの機能範囲を明確にして独立化させることなども考えられるべきであろう．

4. インテリジェント化のためのシステム・コンポーネント

これまで，インテリジェント化について一般的に述べてきたが，次にもう少し具体的にインテリジェント化のためのシステム・コンポーネントについて紹介したい．先に述べた五つのロジカル・システムは，現実にはさまざまな組み合わせにより実現されている．ここでは，“通信システム”と“情報処理・統合 OA システム”を“情報処理・情報通信システム”とし，また“ビル・セキュリティ・システム”と“ビル・オートメーション・システム”について

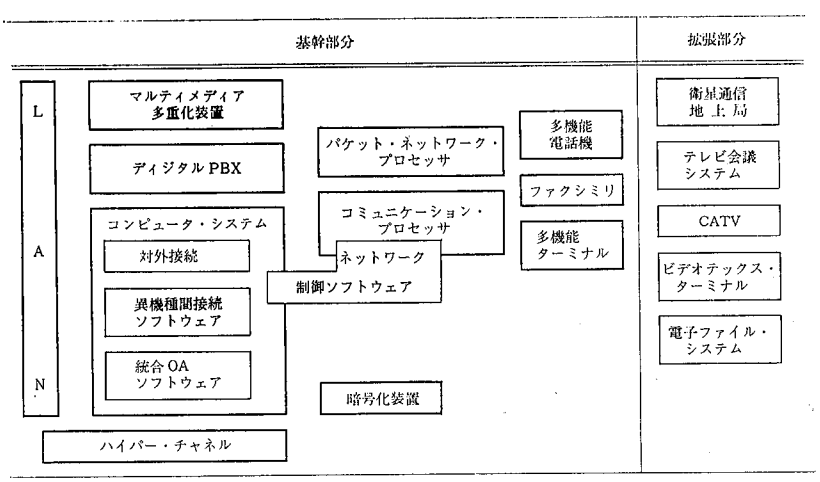


図 1 情報処理／情報通信システム
Fig. 1 Information processing/telecommunication system

は，これを“セキュリティ・省エネルギ・システム”として取り上げることとする．その方が，より現実に即していると思われるからである．また，コンピュータ環境設備と運用もビルのインテリジェント化の一要素であるため，これを“コンピュータ環境システム”として併せて紹介したい．

1) 情報処理・情報通信システム……概略図1にに示したシステム・コンポーネント群が対象とされている．これらは，いずれも本来インテリジェントビルとは独立に発展してきたもので従来から馴染み深いものばかりであるが，インテリジェントビル・ブームを機に，これらをいかなる組み合わせでインテグレートし，ビルに実装するかがコンピュータ・メーカにも問われることになった．

図1のうちの“拡張部分”は，主要なものについてのみ記載してある．

2) オフィス環境システム……オフィスの執務環境を向上させ，あわせてオフィス環境を形成する要素（天井，壁，間仕切り，床，ファニチュアなど）をシステム化（単純化）することによって，オフィスのメンテナビリティを向上させることを目的としている．オフィス環境を構成する要素としては次のものがある．

① オフィス内装
- —天井システム
- —ウォール・パーティション・システム
- —フロア・システム

② オフィス機器・備品
- —オフィス・ファーニチュア・システム
- —オフィス・グッズ（応接セット，スタンド照明，ブラインドなど）

③ 電装設備・機器
- —電気設備システム（無停電照明，床配

表 1 コンピュータ室環境設備・機器
Table 1 Computer environment facilities

設備名	設備・装置・機器・工事
電気設備	定周波定電圧装置，無停電電源装置，自動電圧調整器，専用変圧器，小型無停電装置，受変電設備，配線工事
空調設備	空気調和機，冷水供給装置，冷却塔・ポンプ設備
安全対策設備	電源関連設備，空調関連設備，地震対策設備，地震感知設備，消火設備，防水設備，入退室管理設備
内装設備	防音・防災設備，天井工事，床工事，CDブース
通信設備	電話回線工事，モデム取付工事，光ファイバ・ケーブル，耐雷通信路

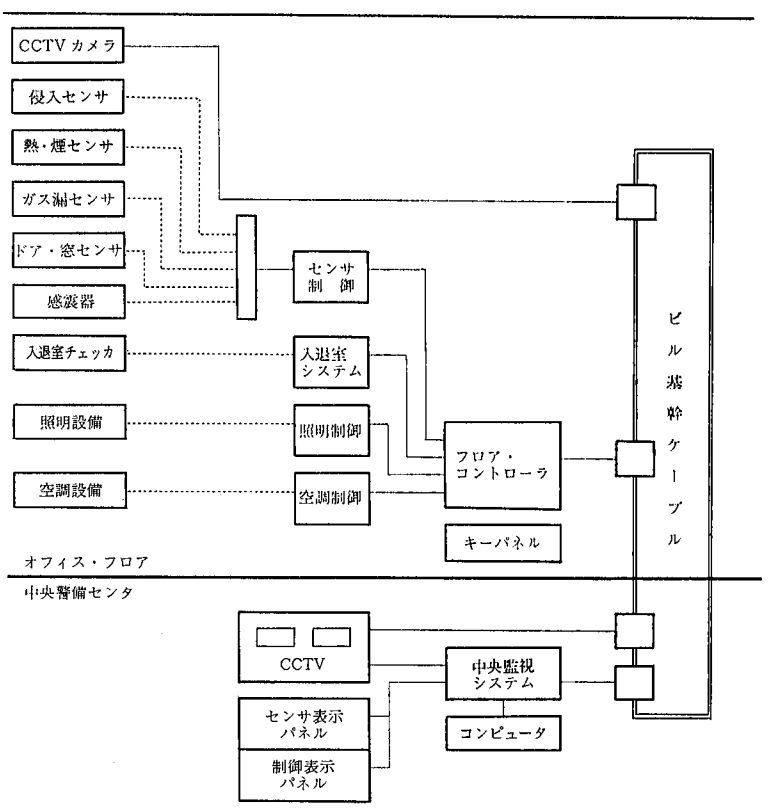


図 2 セキュリティ・省エネルギ・システム
Fig. 2 Security and energy saving system

線，センサなど)
—部分空調システム

3) コンピュータ環境システム……ビルのインテリジェント化の主要な役割を担うコンピュータ・システムの安定稼動，運用・監視の省力化と自動化，およびセキュリティ確保のためのシステムや機械設備の運用・監視の自動化システムおよびコンピュータ室機械設備とがある．

① コンピュータ運用総合自動化システム
—無人運転支援システム
—稼動状況監視システム

② コンピュータ室環境設備
表1に示す設備・機器を主要な対象としている．

4) セキュリティ・省エネルギ・システム……このシステムでは，CCTV モニタ，侵入センサ，ドア・窓開閉センサ，入退室管理システムによる防犯，熱・煙センサ，感震器による防災などオフィス・フロアのセキュリティを主体としている．また，これとあわせて照明設備と空調設備の計画や自動制御による省エネルギの実現を目的とした，中小規模ビル用の汎用ビル管理システムの需要が顕在化している（図2）．

一方，大規模ビル用には，今後ともオーダーメードによる個別システム開発が主体である．

5. インテリジェントビル・サービス

インテリジェントビルの高度な機能を有効に発揮させるため，今後ビルサービスの形態がより専業化することが予想されるが，これらは現実の個々のビルに課される目的や役割によって，さまざまな組み合せが採られることになる．米国においてはこのビル・サービス業が「シェアード・テナント・サービス業」の名の下に登場している．

一般にいわれているインテリジェントビル・サービスを類型別に整理してみると，図3のようにまとめられる．

6. 今後の技術とエンジニアリング上の課題

ビルのインテリジェント化を推進してゆくために，現在，オフィスの OA 化による床構造方式と配線方式の融合，情報通信系伝送路の統合化，ビル制御・セキュリティ系伝送路と情報通信系伝送路との融合，ビル用情報通信機器の統合（たとえばPBXと回線多重化装置），そして OA と OE (Office Environment) との融和など，さまざまな課題が提起されている．

ここでは，近々クローズアップされるであろうと筆者が予想する技術のうち，コンピュータやシステムに携わる者として注視すべきと思われる，①ファシリティ・マネジメント（従来からのいわゆる FM とは趣きを異にするもので，ファシリティ・デザイン・アンド・マネジメントとも呼ばれている），②CG アニメーションによるビジュアル・プレゼンテーション，の2点について若干解説する．

1) ファシリティ・マネジメント……現在，大部分の企業の総務部門などでいわば付帯的に行われている，オフィスを中心としたファシリティの運用管理が，最近注目されはじめている．これはオフィスに関するファシリティ・マネジメントに，より科学的かつ計画的にアプローチし，オフィス環境の良化とコスト・セーブを達成しようとするものである．ビルのインテリジェント化に伴い，OA 機器，通信設備，オフィス機器，エレベータ，照明など，さまざまな設備機器がオフィスに大量に導入されるようになれば，固定資産全体に占める割合が高まってくる．また，経済・産業社会のダイナミックな変貌に対応して，企業組織の変動が激しくなればなるほど，施設（ビルを含めて）の管理をよりシステマティックに行う必要性が高まる．そのため，ファシリティ・マネジメントでは，土地やビルの購入・借用などの不動産管理，ビル・アーキテクチュア・デザイン，保守・清掃・セキュリティなどのビル・オペレーション，オフィス・ファシリティ計画，インテリア・デザインおよびアーキテクチュア，オフィス・ファーニチュア（購入・管理）および通信設備の計画と管理など，広範囲にわたる分野を対象としている．

また，ファシリティ・マネジメントにおいては，全資産，空間，オフィス設備機器が明らかにされ，それらが合理的に配分される必要がある．そのためには，最新かつ正確な現状図と資産明細が保持されていなければならない．このための支援ツールとして CAD システムが有効であり，次のような CAD/FM ソフト分野が想定されている．

- 組織・人事管理
- 施工監理
- スペース計画・管理
- プロジェクト管理
- オフィス機器設備管理
- 財務管理

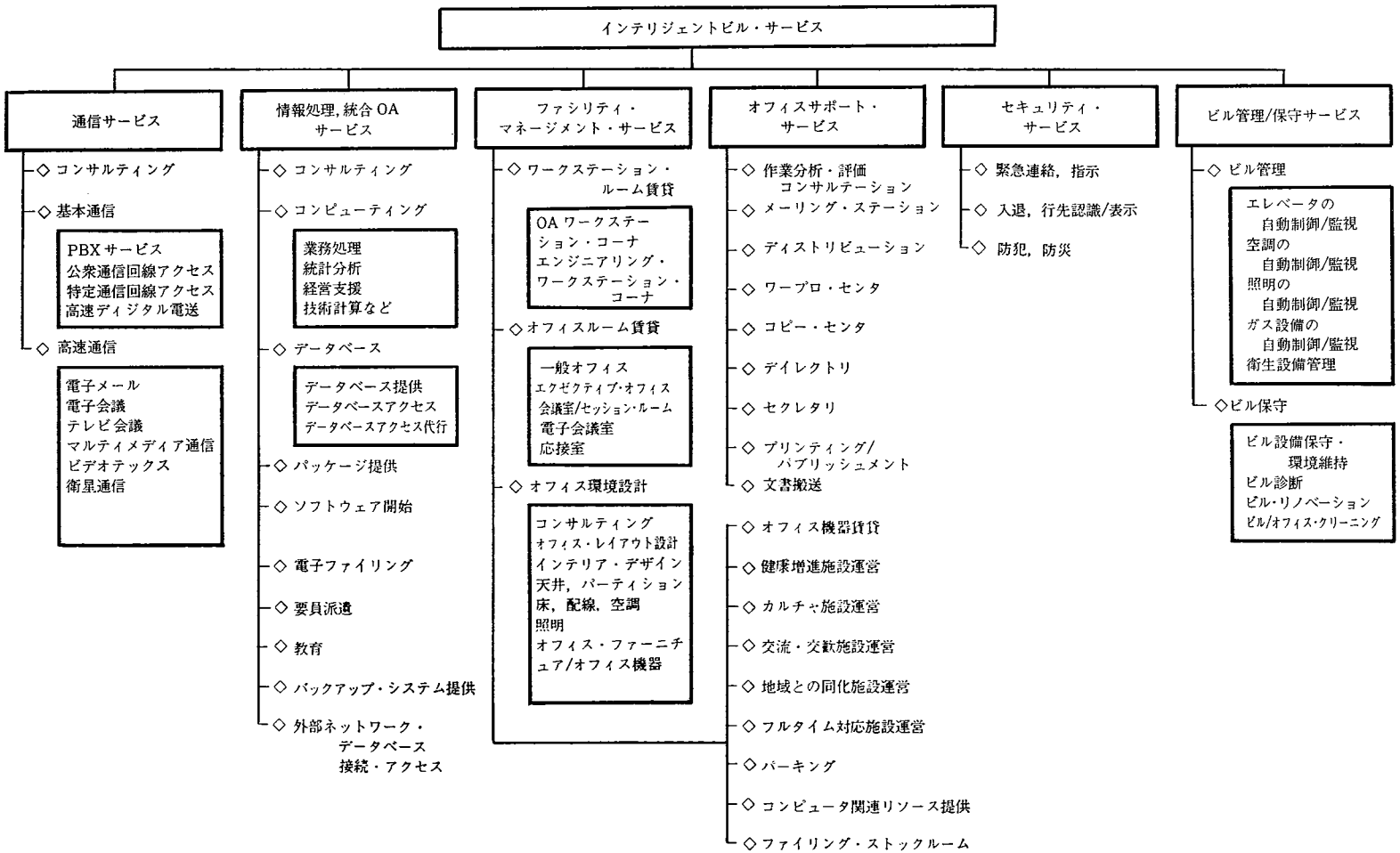


図 3 インテリジェントビル・サービス

Fig. 3 Intelligent building services

- オフィス機器設備バーコード
- 将来予測レポート
- 不動産・リース料管理
- デザインおよび作図

米国では，IFMA (International Facility Management Association) という組織ができており，大企業の FM グループ，建設，通信，家具メーカなどにより，広範囲な活動が展開されている．

2) CG アニメーションによるビジュアル・プレゼンテーション……インテリジェントビル建設の増大，とくに地域開発の促進に伴うインテリジェント・コンプレックスの開発に際しては，専門家を対象とした技術的にも高度な内容のプレゼンテーション，地域住民など素人といってもよい一般の人々へのわかりやすくかみ砕いたプレゼンテーション，行政関係者・議員・関係企業の非専門部門の人々へのプレゼンテーションが必要である．また，計画の諸段階においても力点と内容をそれぞれ異にするプレゼンテーションが，計画遂行上今後益々重要となることが予想される．このように広範囲多岐にわたるプレゼンテーションの方法として，CG アニメーションによるビジュアル・プレゼンテーションが，わが国でも採用されはじめている．

大阪大学工学部笹田助教授の研究室による，日本各地や Paris などの都市を対象とする CG アニメーション (CITIES) や，神戸，横浜，大阪梅田地区などの都市再開発計画への導入事例はよく知られている．このほか企業においても CG アニメーションは使用されており，日建設計の日本電気本社ビル，竹中工務店の後楽園球場エアードームなどが知られている．

CG アニメーションは，単にグラフィックスによる画像の作成だけではなく，音響・映像・シナリオ作りなど，広範にわたる分野にまたがる統合化が必要であり，その制作に際しては感性豊かな人材と多様な器材・資材，さらには多くの資金と時間を必要とする分野である．コンピュータをはじめとする技術進歩が，今後急速に CG アニメーションの適用を容易にし，応用範囲を拡大してゆくものと予想される．

7. おわりに

これまで，日本ユニバックの IBE を中心にインテリジェントビルの現状を述べてきた．日本ユニバックでは，従来から通信システムおよびオフィス環境システムの設計・施工・納入に従事している．このほか，現在オフィスビル，ハイテク・マンション，地域インテリジェント・コンプレックスなどの建設に関連企業・団体などと協力し，積極的に取り組んでいる．インテリジェントビルの長期的目標は，オフィスの生産性と快適性との調和であり，その実現に向って努力してゆきたい．

参考文献

[1] FACILITIES DESIGN & MANAGEMENT, Gralla Publications, Vol. 5, No. 5 May 1986.
[2] Architecture and technology, Series 2, 3, デルファイ研究所, 1986.
[3] 高度情報化時代の OA ビル・OA オフィス設計実務便覧，フジ・テクノシステム，1985.
[4] 浅野元晴　高度情報化時代のビルディング革命，企画センター，1985.

(事業企画部)

山田　真市 著

人工知能のための micro-PROLOG プログラムコレクション

サイエンス社，A５判，252+xi pp. 1986年
2,600円

本書は，著者が早稲田大学理工学部での講義の際に書き留めた Prolog プログラムを集録したものである．Prolog プログラムの実習に役立つように，できるだけ多くの問題が採りあげられており，各問題についてその解決法および使用するプログラミング技法が述べられ，Prolog プログラムとその実行結果が例示されている．なお，Prolog としては，パーソナル・コンピュータ用の micro-PROLOG® が使われている．

本書の前半では，数の計算，リスト処理，集合の処理，論理の処理，ソートなどの基本的な問題を採りあげ，Prolog プログラミングの基礎を学習する．

Prolog のように，新しいプログラミング・スタイルを目指す言語を修得するためには，単に言語の使い方を理解するだけでなく，新しいプログラムの思考法を含めたスタイルの学習が大切である．このためには，新しいスタイルで書かれた基本的なプログラムをできるだけ多く読み，書き，試行してみることが最良の方法であろう．この点で本書は絶好の自習書といえる．

後半では，各種のパズル，エキスパート・システム，数式処理，構文解析などの各分野の典型的な問題を採りあげて応用力を身につける．

そこで使われる解法は理論に十分裏打ちされており，参考文献を与えて読者が理論的事柄の理解を深めるための案内になっている．

以下に，本書の構成および採りあげている問題のいくつかを示す．

０章から２章までは micro-PROLOG の書き方，Prolog の計算機構の仕組み，Prolog と計算機との対話法を述べ，３章以降で必要な基本的事柄を参照ノートになるようにまとめてある．Prolog の処理系は，理論的には J. A. Robinson のレゾリューション論理の証明手続きを強く制限した定理証明系であることを指摘し，それに手続き解釈や問題解決解釈を与えることができることを示している．Prolog の特徴である代入と単一化，非決定性評価などが例をあげて説明されている．

３章では，いくつかの簡単な応用問題を通じこれまでの復習をし，述語や定数の正確な使用が大切であることを学ぶ．関東地方の各県を色分けする四色問題では，課題を隣接関係の記述と互いに相異なる色の公理で表現して，Prolog プログラムが論理仕様となっていることを示す．

４章と５章は数の計算，６章と７章はユティリティ・プログラムで，それぞれ基本的な問題が採りあげられている．数の計算では，最大値，Fibonacci 数，階乗，最大公約数，Eratosthenes のふるい，Diophantus 方程式などの問題を採りあげ，関数の帰納的定義が即プログラムであることを示す．また，効率向上のためのストリーム計算，ループを帰納的定義に変える McCarthy 変換，フィルタなどのソフトウェア技術も紹介されている．組み込み述語を使い慣れた算術記号に置き変えて，プログラムを読みやすくする例も採りあげられている．ユティリティ・プログラムでは，リスト処理，集合の処理，メタ変数の使い方，論理の処理，ソートなどで，頻繁に使われる基本述語を多数集めてモジュール化し，集合処理では，Generate and Test のソフトウェア技法の例をあげ，これが Prolog のバックトラック機能により自然に使えることを示す．８章以降は応用問題を取り扱う．

８章では，ハノイの塔，n-クイーン，水入れのパズルなどの簡単なパズルを採りあげ，Prolog によるトップ・ダウン問題解決や Generate and Test による問題解決の方法を述べる．また，コップの状態と可能な状態推移の記述によって水入れの問題を解く．

９章では人間が覆面算を解く過程をそのまま記述し，数字の非決定性選択と組み合わせることによって解を求め，また数学的に４次の魔方陣を構成する過程を記述して，すべての解を得る．

10章の「考えるプログラム」では，E. Shapiro の MASTER MIND(N=3) と P. H. Winston と B. K. P. Horn の著書「LISP」(Addison-Wesley 社刊，1981．白井・安部共訳，培風館）にある動物識別用のプロダクション・システムを題材にして，論理的思考能力を示すプログラムを作る．プロダクション・システムでは，段階的に３通りの推論エンジンを作って，why や how の問い合わせに答えられるシステムにしてゆく．当然のことながら，Lisp のプログラムに比べてはるかに明解になっており，Prolog がエキスパート・システムの構築に適した

言語であることを示す.

11章では，簡単な因数分解と多項式の微分を例題にして数式処理のプログラムを作る.

12章では，算術式の構文解析とボトム・アップおよびトップ・ダウン還元のプログラムを作る.

この二つの章では，文脈自由文法の構文則とPrologの規則の間の強い類似性を指摘し，Prologの評価法がそのままトップ・ダウン構文解析手続きになることを示している．ボトム・アップ還元には，演算子順位を構文解析によって，スタックをリストで表現して明解なプログラムを作る。

これらのPrologプログラミングを修得することによってPrologが人工知能の分野で有用であることが明らかになるとともに，“プログラムの仕様記述”およびrapid prototypingという，ソフトウェア工学の分野でも役に立つことを知るであろう．多くのプログラマが本書によって，Prologプログラミング・スタイルに習熟されることを望みたい.

（技術研究部　宗像淸治）

コンピュータ関連の辞典

以前，本誌上（第6号，1984年2月）において，同一の表題で辞書・事典類を紹介したことがある．しかし，その後，ハイテク・ブームあるいはマイコン・ブームを反映してか，こと和書に関する限り，『最新』，『…がよくわかる』といった表題を冠した辞典類が多く刊行されてはいるが，初心者はいうに及ばず，かなりの経験者にも理解できるように書かれたものは，きわめて少ないというのが現状である.

本稿では，いわゆる際物（きわもの）を除外し，1981年以降に新刊または改訂された辞典類をリストアップし，その中から数点採りあげて解説を試みることにする．なお，これらのうち，項目番号の右肩に＊印の付いたものは，過年度本誌上で紹介したものである.

1. 洋　　書

[1] M. H. Weik, Communications Standard Dictionary, Van Nostrand, 1981.

[2] C. J. Sippl, Computer Dictionary 3rd ed., Sams, 1980.

[3]* A. Chander (ed), Dictionary of Computers, Penguin, 1984.

[4] Dictionary of Data Communications, Penguin, 1984.

[5] Dictionary of Electronics, Penguin, 1984.

[6] V. Illingworth (ed), Dictionary of Computing 2nd ed., Oxford Univ. Press, 1986.

[7] D. Longley, Dictionary of Information Technology, McMillan, 1982.

[8] C. J. Sippl, Dictionary of Data Communications, MacMillan, 1984.

[9] Dictionary of Electronics, MacMillan, 1984.

[10]* Encyclopedia of Computer Science and Engineering 2nd ed., Van Nostrand, 1983.

[11]* J. Belzer (ed), Encyclopedia of Computer Sciences and Technology, Dekker, 1975～81.

[12] J. L. Steele (ed), Hacker's Dictionary, Harper & Row, 1983.

[13] IEEE Std. Dictionary of Electrical and Electronics Terms, ANSI/IEEE Std. 100-1983.

[14] IEEE Std. Glossary of Software Engineering Terminology, ANSI/IEEE Std. 729-1984.

[15]* C. J. Sippl, Microcomputer Dictionary 2nd ed., Sams, 1982.

[16] L. G. Christie, et al., Microcomputer Terminology, Prentice Hall, 1984.

[17] Webster's New World Dictionary of Computer Terms, S & S, 1983.

[18]* Stan Kelly-Bootle, The Devils DP Dictionary,McGraw-Hill, 1981.

最近，書店の店頭において目に付くのが[6] Dictionary of Computingである．初版が1983年であるので，わずか3年足らずで改訂されたことになる．掲載語数は4,000以上（初版は3,750）で，プログラムの理論，プログラミング言語とその概念，プログラム開発技法をはじめ主要な計算機メーカ，ハードウェア／ソフトウェア製品に至るまで，幅広い分野の用語が採録されている．一般の辞書類のように，定義調の記述ではなく，必要に応じて例題や図表などを挿入し，わかりやすく解説しているところに特徴がある．集合論や符号理論の分野の基礎的な用語についても同じような配慮がなされているが，記述統計学の用語が数多く採録されているのにはいささか疑問を感じる．また，情報科学だけの立場に偏っているわけではなく，マイクロコンピュータをはじめ，最近話題になっている光ディスクの分野や，伝統的な入出力装置やプリンタのリボン，用紙に至るまでの用語が採録されており，情報科学を専攻しようとする学生だけでなく，ユーザ・プログラマにも安心して使える辞書である．なお，収録されている用語は，英国英語の表記が採録されている

ので，不慣れな人には奇異に感じるが，米国の文献では採りあげられることのないソフトウェア製品が収録されているのは非常に有効である．

[13] は，米国内の国内規格や業界規格の中で定義されている用語をすべて収録したもので ANSI/IEEE Std 100 自身が一つの規格である．1977年に初版が刊行されている．今回は1983年版であるが，情報処理用語に関する限り，[14] を併合したほかはまったく変化がない．というのも，ANSI（アメリカ規格協会）の情報処理用語分科会では積極的に規格を制定しないという方針が採択されたためである．

[14] は，IEEE COMPUTER Society のソフトウェア工学技術委員会の援助のもとに発刊されたものである．ISO 用語集および ANSI X3/TR-77 American National Dictionary for Information Processing に対して，ソフトウェア工学の分野の用語が追加されている．一般に，ソフトウェア開発方法論に関する用語は，ソフトウェアのライフサイクルをどのようにとらえるかによって決まる．該書では，ライフサイクルを，concept exploration, requirements, design, implementation, test, installation and checkout, operation and maintenance, retirement の8段階に区分している．なお，前出 [6] においては，system requirements, software requirements, overall design, detailed design, component production, component testing, integration and system testing, release, operation and maintenance である．

[3] は和書 [29] として講談社から，[15] は和書 [33] として共立出版から日本語版が発刊されている．しかし，[3] は最近改訂されたにもかかわらず採録されている用語は古く，[4] と [5] を補完することによって使用に堪えうるものとなる．

[12] は，Stanford, MIT, Carnegie-Mellon といった各大学の研究所においてハッカー達が保守してきたスラング（Jargon file）を集大成したものである．OS のコマンドやパラメタ名の省略形などを日常会話の中で使うと仲間同士と意志の疎通がはかれることがある．該書は，このような文字列の発音の仕方，日常会話における場面での使用方法などを解説したものである．わが国では，ハッカーは，パスワード破りや複写厳禁のソフトウェアをコピーしたりする悪者にされているが，該書ではハッカーの定義なるものが七つあり，いわゆる悪者は最後のエントリに記載されている．ちなみに，前出 [6] においては，ハッカーは「許可を受けずにシステム・ソフトウェアに細工をする人」，となっていて悪者扱いにされているのである．

2. 和　書

[19] コンピュータエンジニアリング用語34,000 英和編，インタープレス，1984.

[20] コンピュータエンジニアリング用語34,000 和英編，インタープレス，1984.

[21] 英和和英 JIS に基づく科学技術用語辞典，全5巻，インタープレス，1979.

[22] 英和和英 科学技術用語に基づく科学技術用語辞典，全9巻，インタープレス，1981.

[23] 両引き術語＝略語集，インタープレス，1984.

[24] 土岐編，英和和英 情報処理用語辞典（改訂版）日本理工出版会，1981.

[25] 英漢計算技術詞衆（第2版），科学出版（北京），1984.

[26] 日英漢電信電子学辞典，人民郵電出版社（北京），1981,

[27] 三島・相磯編，コンピュータ英語活用辞典，オーム社，1984.

[28] 井口編，コンピュータ英和辞典，学研，1983.

[29]* A. Chander 著/坂井訳，コンピュータ用語辞典，講談社，1982.

[30] 日本電信電話会社編，最新データ通信用語辞典，ラテイス（丸善），1984.

[31] ニューコンパクト版 電気電子用語辞典，オーム社，1984.

[32]* 日本ユニバック総合研究所編，共立総合コンピュータ辞典，共立出版，1982.

[33]* C. Sippl 著/岡本他訳，マイクロコンピュータ辞典，共立出版，1984.

[34]* 情報処理学会編，JIS 情報処理用語解説，朝倉書店，1983.

[35] 日経産業新聞編，日経ハイテク辞典（第2版），日本経済新聞社，1985.

[36] 情報・通信新語辞典，日経マグロウヒル社，1986.

[37]* JIS 用語辞典 基本・一般（情報処理用語），日本規格協会，1978.

[38]* JIS 工業用語大辞典，日本規格協会，1982.

[19] から [26] までは，英語と対応日本語（あるいは，その逆）をリストアップした対訳集である．[19] と [20] では，34,000 語という膨大な採録数であるが，一つの用語に対して出典文献ごとに掲載されており，絞り込みがなされていない．このため翻訳者が使用する場合，その選択に困惑することも予想される．

[27], [28] は，辞書というよりは，コンピュータ関係の文献を翻訳するにあたって，その用語がどのように使われているかを，実例を提示して解説したものである．

ここに掲げた以外にも多くの辞書類が発刊されているが，洋書［6］，[10] に見られるように新しい概念をわかりやすく解説したものが見当たらない．和書[32]は，用語編と解説編とからなり，相互に参照することによって理解できる編成となっている．新しい分野の用語も採録されており，まだ新鮮さが損われていない．

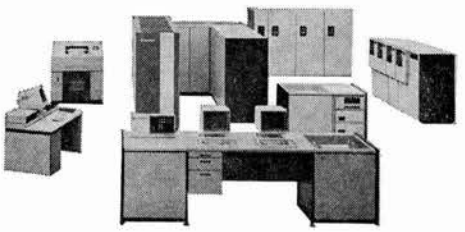
近年，わが国においては，ハードウェア技術の進展には瞠目すべきものがあるが，ソフトウェアの新しい概念や用語などは，依然として「入超」の域を脱しきれていない．昨今，折りにふれてテクニカル・ライティングの貧困が取りざたされるが，その原点である言葉の問題，すなわち辞書の充実にはあまり考慮が払われていない．辞書の編集と刊行は，労多くして報いの少ない仕事である．使いものになる日本語の辞書が少ないのは，外来の概念が身につかないためなのか，それとも陽の当たる場所を見つめるあまり基礎的な事柄に背を向けるという最近の風潮のせいなのだろうか．

（技術情報サービス部　下田宏一）

NEW PRODUCTS

今年発表の製品の中から主なものを選んで紹介します．各製品の詳細についてはマニュアル等をご参照ください．

●1100/90 AD シリーズ



1100/90 AD シリーズは，IP (Instruction Processor) を一つの CPU キャビネット内に2個収納したダイアディック（双頭）・プロセッサ・システムで，従来の 1100/90 に比べて約2倍の CPU 能力を有し，同時にコスト・パフォーマンスを大幅に向上させている．

CPU および主記憶装置をコンパクト化しスペースの削減を図り，さらに電源装置の改良により，消費電力の低減も実現している．

また，1100/90 AD シリーズには，CPU 1台の 1100/91 AD と，同2台の 1100/92 AD の二つのモデルがあり，1100/91 AD は，従来の 1100/91 の約2倍，1100/92 AD は同じく 1100/92 の約2倍の CPU 能力を持っている．

なおソフトウェアは，オペレーティング・システムとして OS 1100 を採用し，従来の 1100 シリーズと完全な互換性がある．

(資料コード：081291004-0)

●科学技術計算用スーパー・コンピュータ ISP システム

ISP (Integrated Scientific Processor) システムは，UNIVAC シリーズ 1100/90 システムの機能分散プロセッサの一つとして開発された科学技術計算専用の高速プロセッサで，追加結合（密結合）により異質型のマルチ・プロセッサ・システムを構成している．つまり，科学技術計算処理装置 (SPU) とその主記憶装置 (SPSU) から構成される ISP が，1100/90 システムに一体化され ISP システムとして提供される．

この結果，ISP システムでは，従来のスタンド・アロン型や疎結合型（チャネルを介して汎用機と結ばれる）のスーパー・コンピュータが持っている問題点，①既設置の汎用コンピュータとのソフトウェア上の互換性，②疎結合型の場合でのホスト・コンピュータとチャネルを介してのやりとりにより生ずるオーバー・ヘッド，③特別な冷却設備や専用スペースが必要，などを解決している．

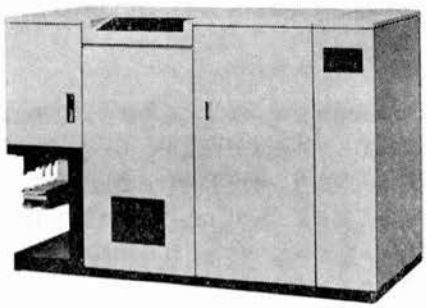
ISP システムは，汎用コンピュータ 1100/90 システムのオペレーティング・システム OS 1100 によって統合的に制御されており，①1100/90 でサポートされるソフトウェアの各種機能を利用できる，②密結合のマルチ・プロセッサ・システムであることから，疎結合のスーパー・コンピュータに見られるチャネルを介してのプログラムやデータのやりとりによるオーバーヘッドはまったくない，③1100/90 に組み込む形で，SPU と SPSU を増設することが可能であり，新たな専用のスペースや特別な冷却装置を必要としない，④汎用コンピュータの操作と同様に 1100/90 のオペレータが操作できる，などの特徴を持っており，従来のスタンド・アロン型や疎結合型のスーパー・コンピュータとは異なり，汎用コンピュータの環境でスーパー・コンピュータの能力を活用できるシステムである．

また，この ISP システムの利用により，科学技術計算のスピードは，従来の 1100/90 での処理に比べ約6～9倍となる．なお，ISP システムは，通常の処理に加えて，ベクター演算とスカラー演算を高速で処理し，最高速度は 266 MFLOPS である．

(資料コード：081291002-0)
(資料コード：481293002-0)

●0490 型日本語印書装置

0490 型日本語印書装置は，日本語処理システム LET'S-J を構成する日本語印書装置の中で最も高速の印書装置で，オンライン・システムとオフライン・システムの2種類が準備されている．



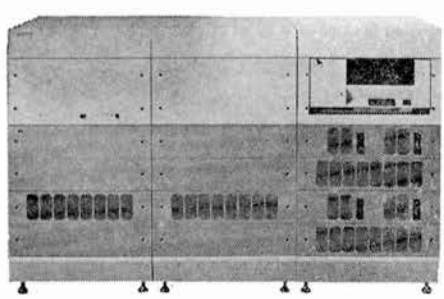
0490 型日本語印書装置は，高速領域の印書装置としては最高の毎分 11,700 行（6行／インチ）の性能を実現している．

0490 型日本語印書装置は，露光部にレーザー・ビームを使用した乾式電子写真方式のノン・インパクト・プリンタで，シャープで鮮明な高印字品質を得ると同時にコスト・パフォーマンスに優れた印書装置である．

さらにライン・プリンタ並みの操作性を実現するために，用紙オート・ローディング機構を採用したほか，トナー補給容器のカートリッジ化を行うなど，操作性の面でも大幅な向上が図られている．

性能面では，①文字サイズは漢字で7ポ，9ポ，12ポ，英数カナで 10 CPI, 12 CPI, 15 CPI 印字が可能，②フォント・メモリはテキスト（7ポ，9ポ，12ポ）およびオーバレイ共に 16K 字まで拡張が可能，③コピー枚数は最大 254 枚まで指定可能，などの特徴を有している．（資料コード：081821032-0)

●シリーズ 7000 モデル 40



シリーズ 7000 モデル 40 は，UNIX* (SYSTEM V. 2) を搭載した 32 ビット・スーパー・ミニコンピュータである．このモデルは，UNIX を高速に走らせる工夫と，メモリ・アクセス・タイムやシステム・パフォーマンスを向上させる工夫によって7 MIPS 以上の高速演算処理を実現している．

シリーズ 7000 モデル 40 は，高速処理など高い性能とともに，CPU パワーを多用する大規模アプリケーションに適したコスト・パフォーマンスの高いスーパー・ミニコンピュータである．

主な利用分野は，高速演算処理を必要とする科学技術計算分野，ソフトウェア開発分野，各種研究開発分野などである．

シリーズ 7000 モデル 40 の特徴は，次のとおりである．①CPU は，32 ビット・アーキテクチャのカスタム・デザインで，インストラクション用，アドレス用およびデータ用にそれぞれ独立したキャッシュ・メモリーを用意し，高速演算処理を実現している．②浮動小数点演算プロセッサを標準装備して，科学計算の処理速度を高めている．③メモリは，4メガ・バイトから最大 32 メガ・バイトまで拡張可能．④入出力系に高速の VERSA BUS** を採用，コンソール・プロセッサをはじめ磁気テープ／ディスク，コミュニケーションなどの制御装置を最大5個まで接続可能で，システム内蔵の高速テープ／ディスク・ドライブおよび最大 240 台までのワークステーションをサポートする．

なお，このシリーズ 7000/40 は，当初構造解析用ソフトウェアなどを搭載したエンジニアリング用システムとして販売されるが，順次専用アプリケーション向きのターンキー・システム（たとえば CAD/CAM を統合した CIM，高度グラフィック処理専用システム）として販売される予定である．

* UNIX は，米国 AT & T 社の Bell 研究所で開発されたオペレーティング・システムで，AT & T 社によってライセンスされている．

** VERSA BUS は，米国 Motrolla 社の登録商標である．

(資料コード：081651010-3)
(資料コード：481655101-0)
(資料コード：481655102-0)
(資料コード：481655301-0)
(資料コード：481655302-0)

●シリーズ 8 AD モデル

このモデルは，従来モデルの基本装備ディスクを大容量化し，かつコンパクト化したもので，従来の同シリーズの低位機・中位機・上位機の各レンジに対して三つのモデルが投入されている．

新モデル3種類は，次の特徴を持っている．

1) システム 100 タイプ AD……基本ディスク容量を従来機の2倍の 40 メガ・バイトとし，最小構成でのワークステーションと端末のサポート台数を大幅に拡大したため，従来上位機でしか利用できなかった大規模アプリケーションの利用が可能となった．
2) システム 200 タイプ AD……オフコン最高速の5インチ 125 メガ・バイトの大容量・高性能小型ディスクを搭載することにより，省スペー

スと省エネルギが実現された.

3) システム400タイプAD……オフコン初の8インチ320メガ・バイトの大容量ディスクを基本装備し，最大3.8ギガ・バイトという汎用機並みの容量まで増設が可能.

3モデルの投入により，シリーズ8は11モデルから成るフル・ラインナップとなった.

さらに，今回強化したネットワーク関連ソフトウェアのうち，シリーズ8間パケット通信強化機能および遠隔管理システム機能は，次のような特徴を持っている.

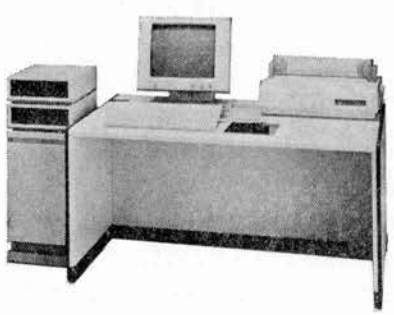
1) シリーズ8間パケット通信機能……これは，①リモート・ワークステーション・ホストおよび端末機能，②双方向同時のリモート・ワークステーション処理機能，③リモート・ファイル・アクセス機能，④プログラム間通信機能，⑤リモート・コンソール機能などをサポートする．なお，これらの機能は，専用回線でも利用できる.

2) 遠隔管理システム機能（DPS Ⅳ機間）……ユーザ内ネットワークにおいて，ホストとなるシリーズ8と端末となるシリーズ8間でのプログラム作成・修正・追加・テストランや異常事態発生時のホスト側からの原因解析・修復オペレーション，端末側からの各種問合せ応答などの高度なネットワーク・システム運用管理機能をサポートする.（資料コード：081751092-3）

（資料コード：081751093-2）

（資料コード：081751094-1）

●Lisp 専用ワークステーション KS-301



KS-301は，オフィス環境で使用できる軽量小型の高速高性能なLisp専用ワークステーションである.

ハードウェアの特徴として，①Lisp言語を高速で処理するため，Lispプロセッサのカーネル部分がマイクロ・プログラム化されており，処理スピードが速い，②メモリーは，最大16メガ・バイトまで拡張可能，③モジュール間のデータ転送を高速に行うために二重バス構造を採用，37.5メガ・バイトの高速性を実現している，④各種入出力装置やKS-301同士を接続するための各種インタフェースを備えている，などがある.

ソフトウェアの特徴として，①マルチ・ウインドゥとマルチ・タスク機能を持つシステム制御ソフトウェアは，Lisp言語の標準となるであろうCommon Lispを採用しており，Zmacsエディタなどの豊富なプログラム開発機能を含んでいる，②LAN（ETHERNET*）経由で接続された他のKS-301とのファイル転送，電子メール，リモート・ログインが可能である，③マウスを利用して容易に絵をかくグラフィックス・ツール・キット，知識をネットワーク形式でディスプレイに表現するツールGRASPER**，論理型言語Prologが用意されている，などがある.

* ETHERNETは，Xerox社の登録商標である.
** GRASPERは，米国Texas Instruments社の登録商標である.

（資料コード：481663001-0）

●エキスパート・システム開発用ツール KEE

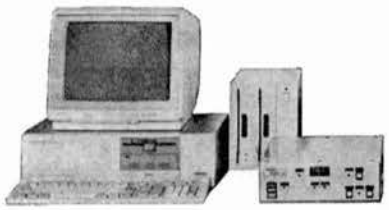
KEE*は，ES（エキスパート・システム）構築用ソフトウェアであり，市場に出ているESシェル（エキスパート・システム開発用ツール）の中で最も高性能な商品の一つである．その特徴は，①フレーム型知識表現とルール型知識表現を組み合わせたハイブリッド型ESシェルである，②フレーム型知識表現は静的な知識（事実）を表現するのに適しており，対象や概念をユニットと呼ばれる構造体で表現し，ユニット間の関係を意味ネットワークで表現する，③ルール型知識表現は動的な知識（規則）を表現するのに適しており，「IF（前提部）THEN（結論部）DO（行動）」というプロダクション・ルールの形で知識を記述できる，④ユニットに動的な知識を付加するためのメソッド機能があり，この機能によってオブジェクト・オリエンテッド・プログラミングが実現できる，⑤推論機能は前向き推論と後向き推論が自由に使用できる，⑥ビジュアルなマン・マシン・インタフェースを提供するアクティブ・イメージ機能によって絵によるメニュ選択やデータ入力が可能，などである.

* KEEは，Intelli Corp.社の登録商標である.

（資料コード：021001002-1）

（資料コード：481663002-1）

●NC 工作機集中制御システム MX-1 モデル 10



MX-1 モデル 10 は，大容量ハード・ディスク (20メガ・バイト，紙テープ約5万メートル分) を標準装置した多機能デスク・ステーション DS 7 をホストに，各種NC工作機をはじめロボット，マシニング・センタなど最大4種のNC機を光ファイバで接続し，集中制御するシステムである．

MX-1 モデル 10 は，NC 機の運用から紙テープ (NC テープ) を追放し NC テープ作成者や NC 工作機オペレータにとって煩わしい紙テープの取り扱いをなくすと同時に，長いプログラムでもNC機側からの1回のデータ・リクエストで連続運転が行えるようにするもので，NC 機の稼動率を大幅に向上させることができる．

また，紙テープ・リーダー・インタフェースしか持たない，ほとんどの旧型 NC 機も接続可能であり，旧型機の能力を大幅に改善できる．

MX-1 モデル 10 の主な特徴は，①現場の工作機側で NC データを容易に呼び出せ，現場の状況に合ったデータ供給が可能，②メーカや型式にかかわらずほとんどのＮＣ機と接続が可能であり，現有設備をそのまま活かせる，③夜間や休日など長時間の連続運転が可能で，NC 工作機の稼動率が向上する，④現場側から MX-1 モデル 10 本体へデータの逆転送ができ，NC 機側からのデータの修正・登録が可能，⑤光ファイバの採用で，ノイズの影響を受けず条件の悪い現場での導入も可能，などである．

(資料コード：081871903-0)
(資料コード：481875901-0)

●ソフトウェア開発総合支援システム TSX 1100

TSX 1100 は，UNIVAC シリーズ 1100 ユーザのために開発されたソフトウェア開発総合支援システムであり，通産省の「シグマ・システム」の先取りとも言えるものである．

TSX 1100 の開発目的は，①ソフトウェアの開発・保守における“生産性の向上”のニーズへの対応，②日本ユニバックのユーザ・システム開発のコンセプト RSDM (Reliable Software Development Method) に基づいた支援ツールの提供，③すでに単体で提供している各種開発ソフトウェア群の統合化，などである．

また，TSX 1100 の特徴は次のとおりである．

1) ユーザ・システム構築におけるソフトウェアの開発・保守情報を，会話型の操作でデータベースに蓄積・管理するための TSX-BASIC (基本部) これは，「シグマ・システム」でソフトウェア・データベース管理システムと言われているものである．
2) システム設計工程を支援するツールとして，データベース処理効率の見積もりを行う TSX-PET．このほか，システム設計段階で定義されるファイル，レコード，画面，帳表などの情報は，データ定義支援ツール TSX-DEFN によってデータベースに登録でき，COBOL 言語で利用できる登録集が自動生成される．
3) プログラムの仕様記述とプログラミングの工程を支援するツールとして使用される超高級言語，かつ仕様記述言語 TSX-PASE．
4) テスト工程を支援するツール TSX-TEST…これは，テキスト・データの作成・印書，モジュール・テスト，実行中の制御の流れやデータ値の監視，プログラム内容の経路分析，出力結果の照合など，テストの工程を総合的に支援するもの．
5) 保守工程を支援するツール TSX-ADMS…これはプログラムの修正履歴情報を管理し，サブルーチン変更に伴う関係プログラムを自動的に更新するなど，開発・保守を円滑に行えるように支援するもの．
6) ソフトウェア開発やユーザのシステム開発全般について，管理者を支援するツール TSX-MANAGE．このツールは，効率良く開発を進めるために，担当者の登録情報，各工程の進行状況，成果物の作成状況などについて，各種レポートを作成し管理者を支援するもの．

これらの特徴を持つ TSX 1100 によって，利用者は蓄積されたデータベースを基に各支援ツール群を利用しながら，システム開発の各工程を効率良く進められ，生産性向上と品質向上を実現できる．

(資料コード：040101001-2)
(資料コード：481205503-0)
(資料コード：481205504-0)
(資料コード：481205505-0)
(資料コード：481205506-0)

ISSN 0289-6257

技　　報

UNIVAC TECHNOLOGY REVIEW

No. 11

発　行　日	昭和61年8月31日
発行人兼編集人	富　田　和　夫
発　行　所	日本ユニバック株式会社 東京都港区赤坂 2-17-51 〒107 TEL (03) 585-4111 (大代表)
頒布価格	1,500 円
印　刷　所	三美印刷株式会社

いよいよ実用化に入ったAI。

私は、米国スペリー社のラリー・L・ウォーカーです。去る5月、スペリー社のパートナーである日本ユニバックは、知識システムとして「KS-301」とソフトウェア「KEE」を発表し、AIの実用化をさらに強力に推進することになりました。

スペリー社では、現在、多数のナレッジ・エンジニアが「ノースウエスト・オリエント航空の座席予約管理システム」をはじめ数多くのアプリケーションをユーザーと共同開発しております。

私たちは、従来の情報システムの世界とAIを融合させ、さらに高次元の問題解決が図れることこそ、実用化を目指したAIであると考えています。そのためには、皆様とスペリー社・日本ユニバックの優れた資質をもつナレッジ・エンジニアとの共同作業が重要なポイントと言えます。

問題の本質を捉え、その解決のためにAIをどう具現化したらよいか……。ユニバックはあなたの問題解決をサポートする、有能なエキスパートです。

Larry L. Walker

米国スペリー社
ナレッジ・システムズ・センター所長

AI is ready to use now.

企業の個性をシステム化する

UNIVAC

日本ユニバック 東京都港区赤坂2-17-51 〒107 ☎03(585)4111
日本ユニバック情報システム
東京都港区赤坂2-17-22赤坂ツインタワー本館 〒107 ☎03(587)8111

AI技術を応用した日本ユニバックの「知識システム」に、知識工学専用ステーション「KS-301」、エキスパート・システムの構築と利用のためのソフトウェア「KEE」が登場。新しい情報処理の世界で、総合的な問題解決のために、力強く応えます。

ユニバック知識システム

KS-301 (UNIVAC Knowledge Station-301) / KEE (Knowledge Engineering Environment)

*KEEは、Intelli Corp社の商標です。