46—S 002

汎用図形処理言語の開発

—ディスプレイ・システムの研究開発—

昭 和 47 年 3 月

JIPDEC

財団法人　日本情報処理開発センター

この事業は、日本自転車振興会の機械工業振興資金による「昭和４６年度情報処理に関する調査・研究補助事業」の一部として実施したものであります。

序

当財団は、情報処理に関する調査および研究開発の一環として情報処理システムにおけるディスプレィ・システムの利用に関する各種ソフトウェアの研究開発を進めておりますが、そのうちこの報告書は汎用図形処理言語の開発成果を報告するものであります。

最近各方面にグラフィック・ディスプレイ装置の利用が普及しつつありますが、ソフトウェアの開発が立ちおくれ、ゆき悩んでいる面も多々うかがわれます。とくに汎用のソフトウェアの不足が大きな問題の一つであり、その基本的な手段として汎用図形処理言語の開発は意義のあるものと思われます。

本報告が各方面に利用され、わが国情報処理産業発展の一助として寄与できるよう念願いたす次第であります。

昭和４７年３月

財団法人　日本情報処理開発センター

会長　難　波　捷　吾

目　　次

ま　え　が　き

１９６３年にＭＩＴの Sutherland が ＳＫＥＴＣＨ ＰＡＤを発表してセンセーションを起し、グラフィック・ディスプレイ利用の画期的な発展が予想されたが、案に相違して今日現在それ程伸びていない。

その主な原因はグラフィック・ディスプレイ装置のコスト高と、汎用ソフトウェアの欠如だと言われている。

コストの問題も非常に重要ではあるが、こゝではさておき汎用ソフトウェアにつき少し考えて見よう。

グラフィックにおける汎用ソフトウェアの汎用のポイントは

Device independent

Interactive operation

Data structure

などが考えられるが、いづれも、あらゆるグラフィック・アプリケーションに共通する汎用的なものが可能であるかどうか、そしてまた汎用化だけが最適の方法であるかどうかについてもまだ多くの議論がある。

これらの事情をふまえて、当財団が取り上げた汎用グラフィック・ソフトウェアは、比較的ベーシックなレベルをねらった。

第１年度は最も基本となるグラフィック・オペレーティング・システム（ＣＧＯＳ）を開発し、第２年度、３年度で汎用グラフィック言語の開発をおこなう。

汎用グラフィック言語は、汎用ソフトウェアの最もオーソドックスなアプローチであり、前述の汎用の三つのポイントおよび既存の共通言語との関連についても充分考慮して設計した。

４６年度はその第２年度目にあたり、この報告書では既存のグラフィック言語の概観と当財団で開発しつつある汎用グラフィック言語、ＵＮＧＬ（ＵＮiversal Graphic Language）の言語仕様を中心に報告する。

第１章　汎用グラフィック言語

第１章　汎用グラフィック言語

第１節　グラフイツク・ソフトウェアとグラフイツク言語

一般にグラフィック言語と云う言葉は、図形処理に関係する広範囲の言語を総称して用いられることが多い。

例えば、その中には数値制御用言語ＡＰＴとか、グラフ処理言語とか、各種の作図言語なども含まれるであろう。ここでは、その範囲を限定して、特にＣＲＴの存在を前提とするインタラクティヴ・グラフィック・システムに於けるプログラミング言語を議論の対象とする。

即ち、単に図形を記述したり、表示したりすることにとどまらず、図形を仲介とする人間と機械の対話に基本を置いた処理形態、即ちマン・マシン・インタラクティブ・システムを実現する様な言語を対象として話を進めてゆくことにする。

始めに、前提とするグラフィック・システムの性格を明らかにする意味で、一般のグラフィック・ソフトウェアの構成とその機能について概説しておこう。

１－１　グラフィック・ソフトウェア

『グラフィック・システムのソフトウェアについて一般的に論ずることはかなりむずかしい。それは現在のグラフィック・システムに於ては、まだ一般的に受け入れられるソフトウェア・システムの概念が、必ずしも固定されていない状態にあるからである。特殊な分野に目的をしぼった実用システムは存在するが、これは他の分野の問題には利用しにくい。グラフィック・システムにはたして普遍的なソフトウェア・システムと云うものが存在し得るかどうかすらが問題にされているのである。』

―情報処理　ＶＯＬ１２№７、コンピュータ・グラフィックス〔２〕大須賀氏論文より引用―

この様に、グラフィック・ソフトウェアの体系に関する一般的な概念は、まだ完全に確立されていないのが現状であり、その一般的な姿に関して論ずることは仲々困難である。

ここでは、比較的現状に沿ったとらえ方として、これを次の３つのレベルでとらえ各々の機能について解説することにする。

① グラフィック・オペレーティング・システム

② グラフィック・システム・プログラム

③ グラフィック・アプリケーション・プログラム

① グラフィック・オペレーティング・システム

グラフィック・オペレーティング・システムはグラフィック・システム全体の運営を管理するコントロール・プログラムであり、その基本的な概念は従来のＯＳと変りはない。具体的な機能を挙げると次の様になるだろう。

- グラフィック・ジョッブのスケージューリングと、その実行の管理
- リソースの割り当て
- グラフィック・コマンド、コンソール・リクエスト等の処理
- マルチプロセッサー・システムでは計算機間コミュニケーション機能等も含まれる。

② グラフィック・システム・プログラム

グラフィック・システム・プログラムは、グラフィック・システムの利用を容易にする様な各種のソフトウェアから構成され、アプリケーション・プログラムから利用される。

その主なものを挙げると次の様になる。

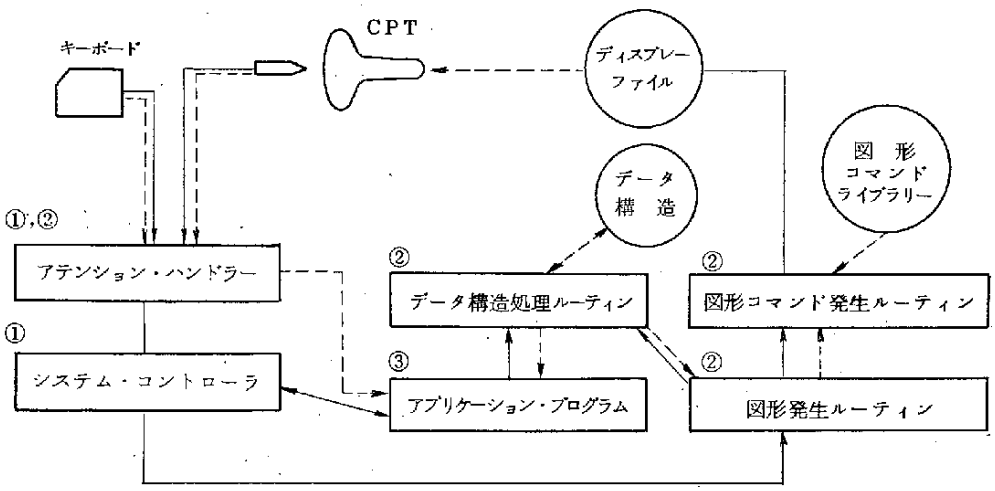
- データ構造を扱うもの
- 割り込み処理を扱うもの
- 図形データからディスプレーコマンドを生成するもの等

③ アプリケーション・プログラム

アプリケーション・プログラムはユーザの問題解析プログラムである。

グラフィック・システムに於ては、利用者が問題を解くためにその都度プログラムを作成、準備することなしに、すでにシステムに準備されている各種のアプリケーション・プログラムを利用する形態が理想的であるとされている。即ち利用者は、計算機に関しては素人であり、アプリケーション・プログラムの開発は別の専門のプログラマに任せられることが多い。したがって他の分野以上に計算機の非専門家にも使い易いプログラムであると共に、それぞれの目的に応じたものが各種必要になる。

これらの３レベルのソフトウェアの関連を〔図１－１〕に示す。



〔図1－1〕

1－2　グラフィック言語の類別

前述した様な、グラフィック・システムを背景とするグラフィック言語には、各種のタイプのものが存在する。

これ等は、一般に次の2つの分類基準から類別されることが多い。

①　言語のレベルに関して

- プロセデュラル・ランゲッジ（手続き向き言語）― 汎用言語
- プロブレム・オリエンティド・ランゲッジ（問題向き言語）― 専用言語

プロセデュラル・ランゲッジは、FORTRAN等のコンパイラー言語レベルのものにあたるもので、問題の処理手続きを一つ一つ記述してゆく形態をとる。汎用グラフィック言語という言葉を使うこともある。

プロブレム・オリエンティド・ランゲッジとは従来のコンパイラー言語の呼び名であるが、ここでは別の意味で使用している。即ち、対象とする問題に密着した表現形式で問題を直接記述出来る様な専用言語を指している。

多くは手続きの記述という形ではなく、必要なパラメータの指定あるいはそれに近い簡略な形をとるものが多い。

②　記述方式に関して

- WRITTEN　FORM　（記述式）

○ POINTING FORM （指示式）

記述式言語は、従来のプログラミング言語の様に文字列でステートメントを記述する形式を指す。

指示式言語は、ライトペン、ファンクション・キーの操作を通じて、ステートメントを指定する形式のものである。①、②の分類のうち、こゝでは①の方式にしたがって話を進めよう。

a 汎 用 言 語

前述したプロセデュラル・ランゲッジにあたるもので、グラフィック・システム・ソフトウェアの機能をより高いレベルで利用することを可能にする様なランゲッジ・ファシリティーであり、多くのアプリケーション・システムを実現する上での共通したプログラミング・エイドとなるものである。

言語レベルからは、FORTRAN、ALGOL等のコンパイラー言語のレベルに属し、グラフィック・データの扱い、データ・ストラクチャ操作、割り込み処理等の機能を備えていることが前提とされる。

4節以下では専ら汎用言語を中心に話を進めてゆく。

b 専 用 言 語

前述のプロブレム・オリエンティド・ランゲッジにあたるものである。問題に直結した表現で対象とする系を記述することができる。しかしあくまでも言語であるから、個々のアプリケーションに密着したものではなく、比較的広範囲の応用分野の問題を取り扱うことが出来る。一般に、この言語によるシステムは、記述された系を計算機内モデルとして表現する機能も備えて居る。又、各種のアルゴリズムに基く、解析用のモジュールも、システム内に含んでいて、ユーザがアルゴリズムの選択をすることにより、対象とする系を解析することが出来る様な形態をとっているものが多い。

この例として、連続系を対象として扱う、BIOMOD（RANDで開発された）、INSIGHT（カリフォルニア大学）、ブロック・ダイヤグラムとして表現される様な系を扱うDESIGNPAD（IBM）、OLBA（東大宇宙航空研）、その他POGO（RAND）、PGS（IBM）等が挙げられる。このうちのDESIGNPADを次の第3節でとりあげその詳細を紹介する。

第2節　専用グラフイツク言語

グラフィック・アプリケーション・システムを編成するための行き方として、汎用言語によるアプローチは必ずしもこの分野の一般的な方向とはなっていない。むしろ、実用性と云う観点からは、汎用言語の開発に熱意を注ぐこと自体、余り意味がないとする見方もある。

即ち、グラフィック・システム・ソフトウェアはベーシックな形で残して置き、その中で効率の良いアプリケーション・システムなり専用言語を実現してゆくのがグラフィック・システムの本来の姿だと考える分である。

専用言語の詳細についてふれることは本書の目的から少しはずれることになるかもしれないが、上で述べたような事情を考慮すると仲々見逃せない存在と思われる。ここではその一例としてDESIGNPAD（A computer graphics system for block diagram problems byL.A.Belady,M.W.Blasgen,C.J.Evangelisti and R.D.Tennison IBM SYST J №2 1971）について簡単にふれておくことにする。

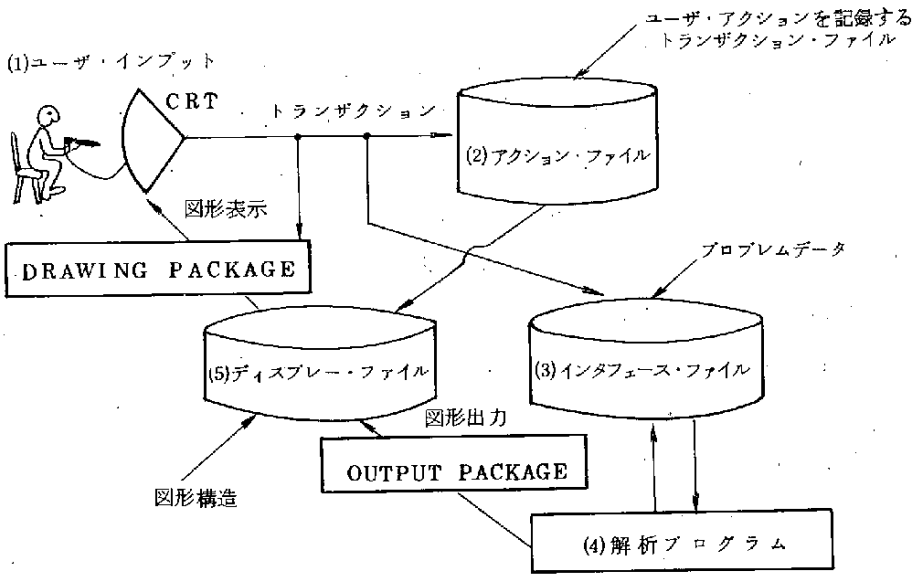
2－1　DESIGNPAD（IBM）

概　　要

従来のグラフィック・アプリケーション・プログラムは、応用分野ごとに独自のシステムを構成するという行き方をとっていたため、他の応用分野との互換性を欠き、本質的に非常に類似した問題であるにもかかわらず、別の応用分野のものになると扱う事が出来ない場合が多かった。ここで紹介するDESIGNPADシステムは、こうした事情を考慮し、対象とする系をレーベルド・ブロック・ダイヤグラムに帰着させることにより、より広範囲の問題を統一的に扱う事を試みている。DESIGNPADを使用するユーザは、システムの提供するコマンド・ランゲージを通じて、CRT画面上に、インタラクティヴにブロック・ダイヤグラムの作成・修正を行う事が出来る。又システムはモデリングの過程を図形表示すると同時に、プロブレム・モデルを内部的に作成し、ユーザの指示に従って解析プログラムを起動し、これを解析し結果を画面に表示する事が出来る。

この時、解析プログラムは、ユーザが作成したプログラムを使用する事も出来るし、システムの提供するGPSS、ECAP、CSMP、LISP等を使用する事も可能になっている。

〔図2－1〕は、DESIGNPADの構成を示したものである。



以下にその動作概要を簡単に記す。

(1) ユーザ・インプット：ユーザはライトペン、キーボードを使用して、ＣＲＴ画面から問題とする系をインプットする。入力形式は、アプリケーション・インディペンデントな形、即ちレーベルド・ブロック・ダイヤグラムの形で行なう。

(2) アクション・ファイル：ユーザのインタラクションのヒストリーは、トランザクションと云う形で、中央のアクション・ファイルに送られ保存される。アクション・ファイルは、画面から入力されたすべての情報（図形情報、トポロジカル情報等）を含むマスタ・ファイルとして重要な役割を果す。

(3) インタフェース・ファイル：ユーザ・インプットから作成される問題用データ構造である。定義されたブロック・ダイヤグラムのモデルが作成される。

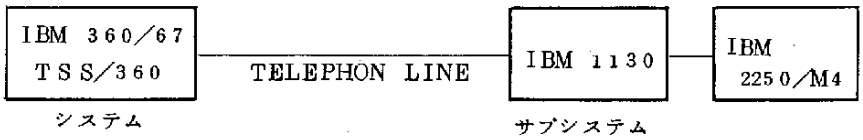
(4) 解析プログラム：インタフェース・ファイルを対象として、問題解析を行なう。対象とするアプリケーションが変わってもこの部分だけを取り変えればよい。

(5) ディスプレー・ファイル：表示用図形データ・ファイルである。ユーザ・インプット又はアクション・ファイルから作成され、ユーザに作成中のモデルを表示する。又解析プログラムの出力からも作成され、結果の表示をする。

この様にＤＥＳＩＧＮＰＡＤ に於ては、対象とする問題が変わってもそれが基本的にレ

ベルド・ブロックダイヤグラムとして表現される系であれば、ユーザは単に、解析プログラム又は解析プログラムとプロブレム・データ・ストラクチャのインタフェースだけを考えればよく、従来の形式（アプリケーションごとにシステム再編成する）に較べれば、その労力は大幅に軽減されたことになるだろう。

ハードウェア構成



〔図2－2〕 DESIGNPADのハードウェア構成

〔図2－2〕はDESIGNPADのハードウェア構成を示したものである。2台の処理装置の間でロードを分担する2プロセッサー・システムを採用している。

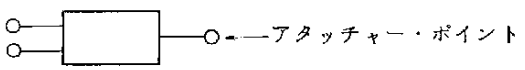
両プロセッサーは2,000ボー回線で結ばれ、サテライト側は主として、リアルタイムが要求されるユーザとのインタラクションを処理し、ホスト側はデータ・ベースの提供と作成されたモデルの解析を行なう。

レーベルド　ブロック　ダイヤグラム

問題解析を行なうためには、ユーザは対象とする系をブロック・ダイヤグラムとして表現しなくてはならない。

はじめに、ブロック・ダイヤグラムを構成する基本的な要素の概念について述べておこう。DESIGNPADでは、ブロック・ダイヤグラムの構成要素として次の4つの基本的なブロックを扱う。各々は、後述するプロブレム・データ・ストラクチャーに於ける1エンティーに対応する。

(1) レーベルド・ブロック――モデルの素子を表現するための最も基本的なブロックである。他の要素と接続するためのアタッチャー・ポイントを備え、ブロックの形状とアタッチャー・ポイントの数はユーザが任意に定義出来るようになっている。〔図2－3〕参照



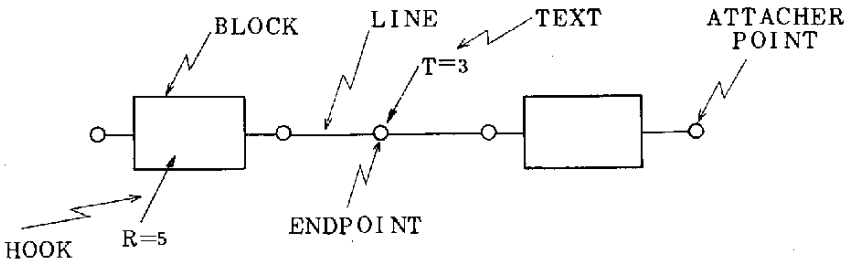
〔2－3図〕

(2) テキスト⟶キャラクター・ストリングの集合で、ブロックを修飾するために用いる事も出来るし又テキスト自身をブロック的に使用する事も可能である。

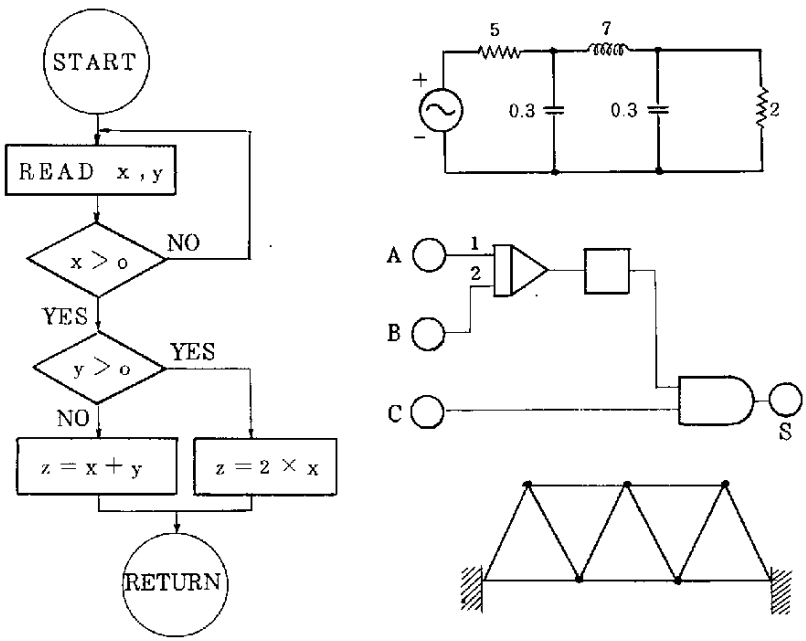
(3) フック⟶テキストを他の要素と関連づけるために使用する。

(4) ライン⟶2つの端点を持つ線分で、ブロック同志を結合するために使用する。（ブロック間の関連表現のために使用する。）

〔図2－4〕は、これ等の関係を図示したものである。又〔図3－5〕はレーベルド・ブロック・ダイヤグラムとして表現される様な系の一例を示したものである。これをどの様に表現するかはユーザの任意である。即ちライン、テキスト、ブロック、フックを系のどの要素に対応させて用いてもよい。



〔図2－4〕 ブロック、テキスト、アタッチャポイント、ライン、エンドポイント、フックの例



〔図2－5〕 レーベルド、ブロック、ダイヤグラムの例

ブロック・ダイヤグラム作成のための便宜

(1) シ ー ト

ユーザはＣＲＴ上にブロック・ダイヤグラムを作成してゆくが、それはシートと呼ばれる仮想の媒体に記録される。１シートは６０ｆｔ×６０ｆｔの大きさを持ち、システムは複数枚のシートを準備している。各シートはユーザが、つける識別名を持ち、識別名による検索が可能である。又１ブロックを１シートに対応させることにより、シート間に階層的な関係を持たせることも出来る。これ等の関係（ブロックとシートの関係）はライブラリー・シートと呼ばれる特定のシートに記録される。（〔図２－６〕参照）

(2) Ｖｉｅｗｐｏｒｔ

ＣＲＴ画面は、最大４つのＶｉｅｗｐｏｒｔに分割して使用することが可能である。即ち同時に４つのシートを写し出す事ができる。又Ｖｉｅｗｐｏｒｔの大きさはユーザが自由に決めることができる。通常のデザイン・プロセスに於ては、次の様にＣＲＴを４つのＶｉｅｗｐｏｒｔに分けて使用するのが便利である。

① モデリング・シートの表示 ―→ 設計中のブロック・ダイヤグラムを表示する。

② ブロック・メニューの表示 ―→ ユーザが定義したブロックはやはり１つのシートに収容されている。設計は、これを引用しながら、能率よく進めることが出来る。

③ コマンド・メニューの表示 ―→ モデリング・サブシステムに対する指示を与えるライトキーを表示し随時システム・オペレーションをとることが出来るようにして置く。

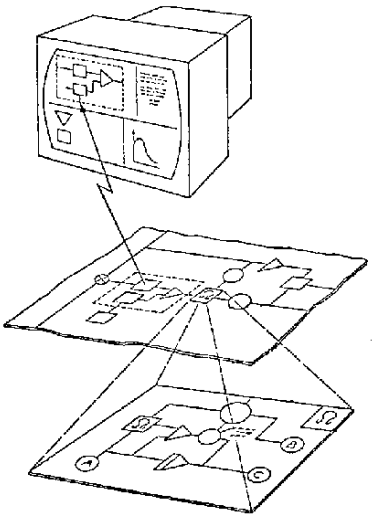
④ アウトプットの表示 ―→ 解析後の結果を表示する。（図２－６参照）

(3) ＷＩＮＤＯＷＩＮＧ

シートの大きさに対して、これを表示するＶｉｅｗｐｏｒｔが小さいので、シートの必要な部分だけをとり出して表示する操作が必要となる。（ＷＩＮＤＯＷＩＮＧ）

ＤＥＳＩＧＮＰＡＤでは、ディスプレー・ファイルに、セル構造を採用することにより、これを効果的に行なう事が出来るようになっている。（セル構造については後述する）

ユーザはＷＩＮＤＯＷＩＮＧコマンドを使用することによりＶｉｅｗｐｏｒｔに表示されたシートを連続的に移動させ、シートの任意の位置を覗たり、アクセスしたりすることが可能である。



〔図２－６〕

コ　マ　ン　ド

ユーザはコマンドを使用して、ブロックを定義したり、ダイヤグラムを作成したり、すでに作成ずみのシートを表示してみたり等、システムの機能を十分に利用する事が出来る様になっている。

どの様にして、実際のユーザ・アクションがとられるかを知る意味で、以下にコマンドの一覧表をかかげておく。

コマンド一覧表

Start－up

Cold start	システムを初期化しライブラリ・シート、ライト・キー・シートと２つのブランク・シートを表示する。
Warm start	以前の状態から再開始する。

Control

Viewport size	(Viewportのサイズを決めるためのライト・キーをピックし)、ライトペンでViewportの境界を指定する。
Display sheet	指定されたライブラリ・シート上のシートを、指定されたViewport に表示する。
Window	Viewportを指定し、ライト・キーをピックすることにより、ウインドイング、コントロールがスクリーンに表示される。ライトペンでコントロールを指定することによりViewport内でシートを連続的に移動させることが出来る。
Clear sheet	指定されたシートをクリアーする。
Analyze	アプリケーション・プログラムの名前を指定すると、現在モデリングViewportに写っているブロック・ダイヤグラムを対象に解析を始める。
Display／blank hook	全てのフックを表示／消去する。 Modeling Viewport内のブロックを指定し、そのブロックにつながるline あるいはブロックに沿って、ライトペンでブロックを動かすことができる。line は必要に応じて、引き伸ばしたり、縮めたりできる。

Drawing

Draw line	Modeling Viewport内で任意の点を指定し、そこからラバー・バンドラインを使って自由にラインを書くことが出来る。
Copy block	reference viewport 内のブロックを指定し、modeling viewport内の任意の位置に置くことができる。

Erase line/block	モデリング Viewport 内のライン又はブロックを指定して消す。

Text editing

Begin text	modeling viewportの任意の位置にテキストを作成出来る。
Edit text	通常の text-editing 機能が用意されている。
Delete text	テキストを指定してそれを消去する。
Relate text	テキストをライン、ブロック、アタッチャ・ポイントのどれかにフックで結びつける。
Erase hook	フックを指定してそれを消去する。

Block definition

Specify shape	ライト・キーをピックすることによりshape specificationモードに入る。ブロックを定義するための画面が表示される。
Draw	このモードで、ブロックの形状が定義される。又ライト・キーを使ってアタッチャ・ポイントが定義される。
Label block	キャラクタ・ストリングでブロックにレーベルをつける。
Specify function	形状の定義が終ってから、ライト・キーを指示するとモデリング・フェーズに移す事が出来る。 この状態で、そのブロックの詳細を別のシートに定義する事が出来る。（ブロックとシートの対応）
Specification complete	ライト・キーによりこのフェイズを完了させる。

Sign off

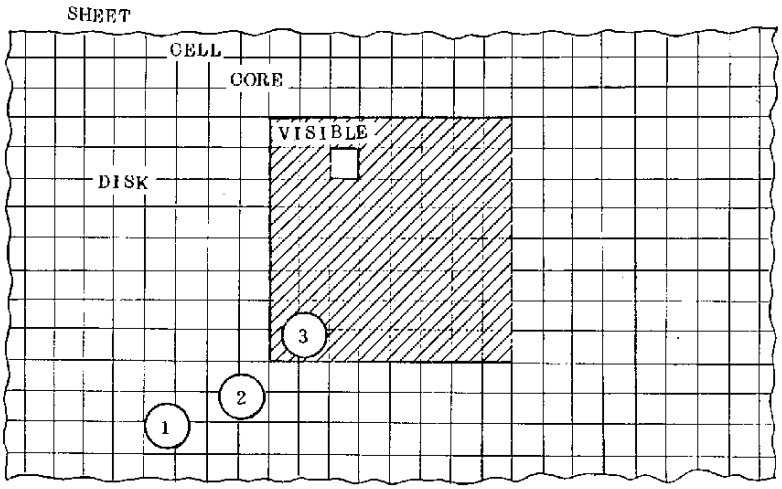
Save	ライト・キーにより、定義された全てのブロックと、モデルを、ユーザの名前でセイブしておく。
Quit	クイットをかける。

〔図２－７〕はＤＥＳＩＧＮＰＡＤのソフトウェア構成を示したものである。

ＤＲＡＷＩＮＧ　ＰＡＣＫＡＧＥは、シートの図形的情報の管理をつかさどる。即ち、ＣＲＴ画面からの図形入力を受け取りこれをディスプレー・ファイルに記録したり、又、表示要求の出たシートをアクション・ファイルから再生し表示したりする。

シートの図形情報は、ディスプレー・コマンドオーダーの形でディスプレー・ファイルに保存される。ディスプレー・ファイルは、主記憶、２次記憶にまたがって作成されるが容量に限度があるため、その時点で必要なものだけが蓄積されている。又ウィンドイングを効果的に行なうため、セラー構造と云う特別な構造を備えている。

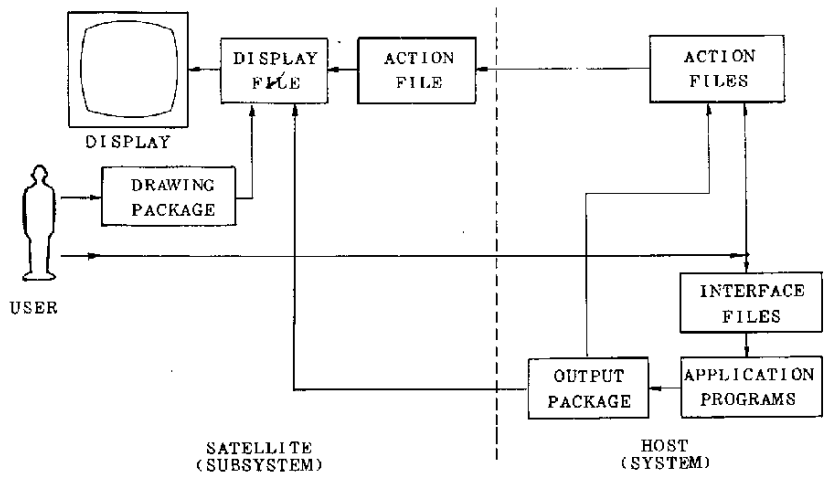
＊セラー構造→シート上の図形を量子化しセル単位でこれを扱う方法である。即ちシートを〔図２－８〕の様に1.5インチ四方の格子に分割し各セル（ます目）に１つのディスプレー・サブルーティンを対応させる。各ディスプレー・サブルーティンはシート上の位置情報にちなんだ名前がつけられて管理されているから、シート上の特定の範囲が指定されると、ど



(2)—(3) USER CONTROL

(1)—(2) ANTICIPATED BY SYSTEM

〔図２－８〕



〔図２－７〕　ソフトウェア構成図

のディスプレー・サブルーティン群が表示の対象となるかがすぐ判るようになっている。

アクションファイルは、ユーザ・アクションの記録であり、シート上のブロック・ダイヤグラムの情報は、図形情報、トポロジカル情報も含めてすべてこの上に記録されている。

例えば、あるシートの表示要求が出た時、これがディスプレー・ファイル上にすでに存在しなければDrawing package はアクション・ファイルからこれを再現することが出来る。

〔図２－９〕は、ユーザ・アクションによって発生するトランザクションの形式を示したものである。トランザクションはホストに向けて転送されアクション・ファイル、インタフェース・ファイルのインプットとなる。

インタフェース・ファイルはサテライトから送られてくるトランザクションにより作成されるプロブレム・データ・ストラクチャーであり、解析プログラムへのインプットとなるものである。

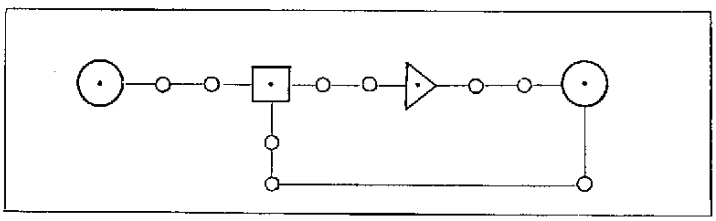
構造化の方法は、基本的にはＬＥＡＰと同じ形式で、ブロック・ダイヤグラムを構成するブロック、テキスト、フック、ラインをそれぞれ１アイテムとし相互の関係を関連３つ組と云う形で表現してゆく。又解析プログラムへの便宜をはかる意味で、システムはあらかじめ定められた形式で構造化を進めている。（例えば、Ｓｅｔ　ｏｆ　Ｌｉｎｅ という形式で、ダイヤグラムに属するすべてのラインが直ぐに得られるように構造化している）

例えば〔図２－１０〕のブロック・ダイヤグラムが定義されると、インタフェース・ファイル

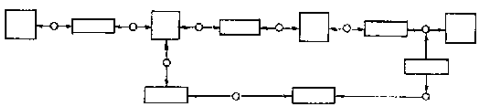
TRANSACTION	COMMENTS
OM SHEETA	THE FOLLOWING TRANSACTIONS ARE ON THE SHEET WIIOSE BLOCK IS ATAON LIBRARY SHEET
COPV a . B	COPY AT LOCATION ON TME ABOVE SHEET AN INSTANCE OF TME BLOCK THAT IS AT ON THE LIBRARY SHEET
DRAW b . d	DRAW A LINE FROM b TO d
COPY d . C	
ORAW e . l	
COPY g . D	
DRAW h 。 i	
COPY i . E	
DRAW h . I	
DRAW l . m	
DRAW m . i	

UPPER CASE LETTERS ARE COORDINATES ON LIBRARY SHEET
LOWER CASE ARE COORDINATES ON THE MODELING SHEET

〔図２－９〕

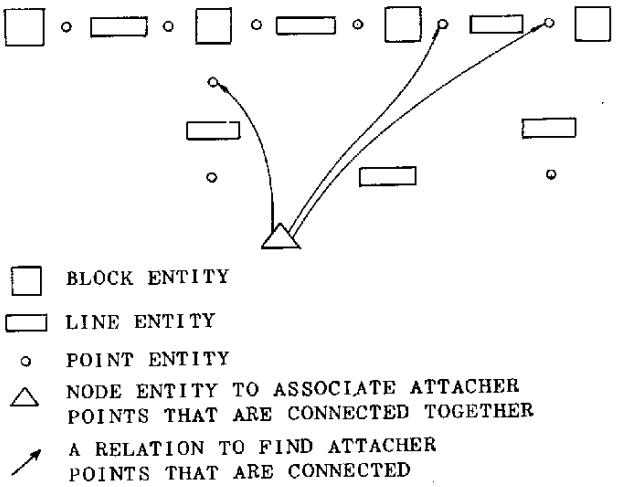


〔図２－１０〕　ブロック・ダイヤグラムの例

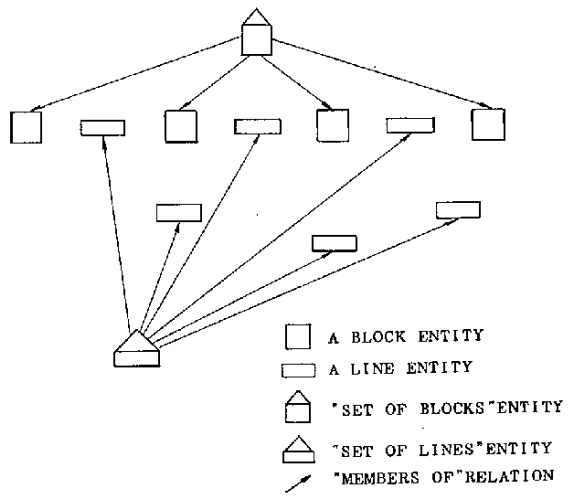


BLOCK ENTITY
LINE ENTITY
POINT ENTITY
RELATION RELATING A POINT WITH BLOCKS AND LINES

〔図２－１１〕　トポロジカルモデル



〔図２－１２〕



〔図２－１３〕

に於ては〔図２－１１〕の形式でこのトポロジカル・モデルが作成される。〔図２－１２〕、〔図２－１３〕は、特にユーザが定義したものでは無いが、構造の管理上都合のよいように、システムがつけ加える関連である。

〔図２－１２〕は、３点がラインだけで結ばれていることを示している。(例えば電気回路の場合は等電位点にあたる)又〔図２－１３〕は、モデルの中のすべてのライン・エンティティー

ブロック・エンティティー を知る事が出来るようにこれを管理するＳＥＴ of line エンティティーSet of Block エンティティーを設けて、これにすべてのライン、ブロックを関連づけている様子を示している。

最後に、Out put package は、解析結果を表示するための便宜をはかる。結果はｘ－ｙグラフ、バーグラフ、２変数関数の投影図等に変換され、ＳＨＥＥＴに格納される。

第3節　汎用グラフイック言語としての機能

汎用グラフィック言語は、1節でその位置づけがなされたように、グラフィック・システムの提供する多くの機能を、より高いレベルから効果的に利用できるようにするプログラミング・サポートとしての位置を持つものである。

この節では、汎用グラフィック言語として要求される基本的な機能として、①グラフィック・データ処理機能　②割り込み処理機能、③データ・ストラクチャ操作機能、④非グラフィカルな機能の4つを考え、各々の機能についての説明と、それが一般の言語に於て、どの様な扱いをされてきたかについての概観を与える。

3－1　グラフィック・データ処理機能

ハイレベルのグラフィック言語では、基本的な図形要素の概念、点、線から、さらに進んで、より高度な図形の概念（例えば構造物、回路図）を表現する形式が必要とされる。以後、この様な図形の概念を図形イメージと呼ぶ事にする。

プログラミング言語に於て図形イメージを自由に表現するためには、具体的に次の様な機能を考慮しなくてはならない。

① 図形イメージを表現するためのグラフィック・データの導入

② 図形イメージの操作機能　図形イメージに回転、移動、拡大、縮小、ウインドーイング等の操作を加える事が出来る様な機能

③ 図形イメージのExpression　いくつかのイメージに適当な操作、例えば②の操作を加えたものを一つの図形イメージとして表現出来るような表現形式。

④ 図形イメージを表示する機能　具体的には表示データの作成、グラフィック入出力、ディスプレー・ファイルに対するアクセス等の処理をする機能

以下では、図形イメージの内部表現形式と云う点から言語を2つのタイプ、ディスプレー・プロセデュア形式とディスプレー・データ・ストラクチャー形式に分け、各々に属する言語について、①～④の機能がどの様に扱われているかを見てゆく事にする。

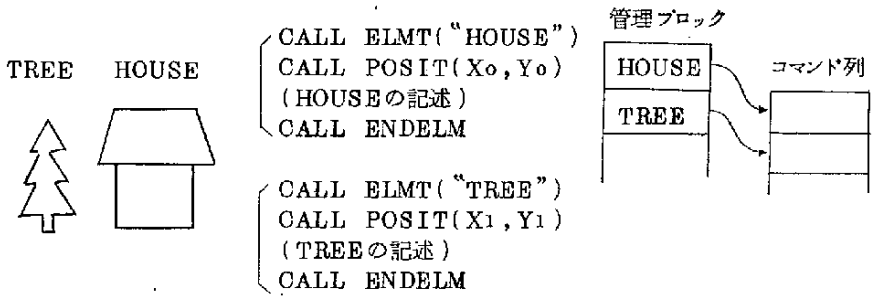
(1) ディスプレー・プロセデュア形式

図形イメージを内部的にプログラムのつながりとして表現する方法である。即ち図形イメージは、表示データを生成するプロセデュアとして定義され、図形イメージの階層関係はプロセデュア間のリンクとして表現される。従って、表示データは、出力命令によってこれ等の一部が実行されることによって作成されることになる。

この種の扱いをしている言語として低レベルのものではＧＳＰ、進んだものでＤＩＡＬ、ＭＥＴＡＶＩＳＵがあげられる。

◦ Ｇ Ｓ Ｐ

ＧＳＰではあらかじめ、ＶＥＣＴＯＲ，ＰＯＳＩＴ 等の基本的な図形データを編集するためのモジュールや、論理的なまとまりを指定するためのモジュール、ＥＬＭＴ、ＥＮＤＥＬＭが準備されて居り、これ等のモジュールをＣＡＬＬ形式で呼び出すことにより図形記述を行う事になる。〔図３－１〕参照、この場合図形イメージは一次元的なプログラムの流れの中に表現されていると見る事が出来るだろう。図形イメージの構造化、操作等の機能は、このレベルのものには見る事は出来ない。



〔図３－１〕

◦ ＤＩＡＬ

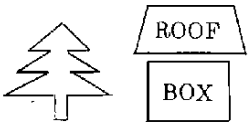
ＤＩＡＬでは、図形イメージを表現するものをディスプレー・プロセデュアと呼んでいる。

ディスプレー・プロセデュアは基本的には次に示す４つの図形出力ステートメントによって定義される。又、これを階層的に記述する事も出来る。即ち一つのディスプレー・プロセデュアから、既に定義されている他のディスプレー・プロセデュアを引用して、図形記述の便宜をはかる事が出来る。

ＭＯＶＥ　Ｘ，Ｙ	相対的な位置ずけ
ＭＯＶＥ　ＴＯ　Ｘ，Ｙ	絶対的な位置ずけ
ＬＩＮＥ　Ｘ，Ｙ	直線記述（相対）
ＬＩＮＥ　ＴＯ　Ｘ，Ｙ	直線記述（絶対）
ＤＩＳＰＬＡＹ “　～　”	ＴＥＸＴの記述

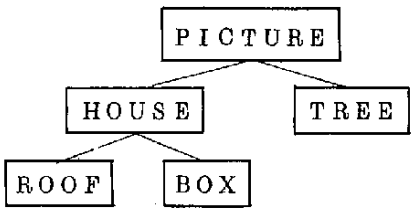
例えば〔図３－２〕で示される例はＤＩＡＬでは次の様に表現される。又〔図３－３〕はこれらのステートメントにより作成されるプロセデュアの階層関係を示したものである。

（例）




```
PICTURE ←"HOUSE AT X1,Y1, TREE AT X2,Y2;"
HOUSE   ←"ROOF AT X3,Y3, BOX AT X4,Y4;"
TREE    ←"MOVE TO X1,Y1, LINE Xn,Xn …
          …TREEの記述;"
ROOF    ←"ROOFの記述…;"
BOX     ←"BOXの記述…;"
```

〔図３－２〕



〔図３－３〕

この中の任意のディスプレー・プロセデュアを指定して表示命令が実行されると、関連したプロセデュアが次々と実行され表示データが作成されてゆくことになる。

ここでＤＩＡＬに於ける図形の扱いを図形イメージの操作性という点から整理すると次の様になる。

① ディスプレー・プロセデュアの構造はプログラム作成時にスタティックに決定される。

② ディスプレー・プロセデュアを引用する時その位置情報、スケール情報を与える事が出来る。

③ ディスプレー・プロセデュアにアーギュメントを与える事が出来る。

④ ディスプレー・プロセデュア内に判断文、繰返し文等のステートメントを書く事を認めている。

①は実行時に動的に図形構造を変える事の出来ない事を意味する。これは構造化の方法から来る大きな問題である。

③、④は、この問題（イメージの動的な操作性）に対してある程度その機能を補ってくれるだろう。

②は、定義した図形を引用する時にどの程度までに、それに操作を加えられるかと云う事で、サイズや位置情報は自由に変えることが出来る事を示している。

○ METAVISU

METAVISUに於ける図形イメージの扱いもDIALによく似ている。METAVISUでは、図形イメージを表現するものをグラフィック・プロセデュアと呼び、PL/Iのプロセデュアに似た形でこれを表現する。グラフィック・プロセデュアの定義の仕方も、DIALとほとんど変わらない、即ち基本的な図形出力ステートメントと、他のグラフィック・プロセデュアの呼び出しと云う形で一つのグラフィック・プロセデュアが定義される。次の例は〔図3－1〕の例をMETAVISUのグラフィック・プロセデュアにより表現したものである。

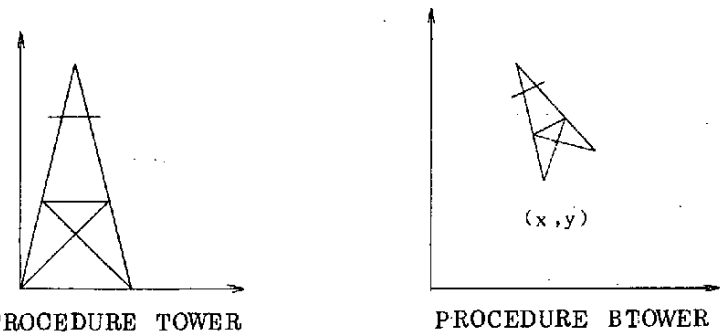
```
(例)  PICTURE;PROCEDURE;
        CALL  TREE  AT  X2 , Y2
        CALL  HOUSE  AT  X1 , Y1
      END;
      HOUSE:PROCEDURE;
        CALL  ROOF  AT  X3 , Y3
        CALL  BOX   AT  X4 , Y4
      END;
      TREE:PROCEURE;
        LINE  TO  X5 , Y5 TO X6 , Y6……;
      END;
      ROOF:PROCEDURE;
        LINE  TO  ……  ;
      END;
      BOX:PROCEDURE;
        LINE  TO  ……  ;
      END;
```

図形イメージの操作性と云う点に関しては、DIALよりやや機能が拡張されている。DIALでは、ディスプレー・プロセデュアに対して、移動、スケーリングの操作しか加える事が出来なかったが、METAVISUではさらに、回転、マスキング等の特殊な操作を加える事が出来る様になっている。

例えば、次の例は定義されているグラフィック・プロセデュアTOWERを使って、これを

1／2のスケール・ファイターでスケーリングし、これを30° 回転したものを（x、y）の位置に移した図形をBTOWERとして定義する過程を示したものである。

（例）



〔図3－5〕

```
BTOWER：PROC；
R＝3.141592／6.0
CALL TOWER AT x、y SCALE 0.5、0.5 ROT R
END；
```

(2) ディスプレー・データ・ストラクチャー形式

これは、より一般的な方法、即ち図形イメージを内部的にデータ・ストラクチャーとして表現する方法である。図形記述ステートメントは、実際にはデータ・ストラクチャーの記述であり、実行時に内部にデータ・ストラクチャーが作成される。即ち、図形イメージの実体はデータ・ストラクチャーとして在る事になる。

実際の表示データは、表示命令が出された時にデータ・ストラクチャーを対象として作成される。（データ・ストラクチャーをそのままコマンド・イメージで作っている例もある）

この方法は、インタプリティヴに表示データが作成されると云う点で、処理速度に問題を残すが、図形イメージの操作性、図形構造の動的な操作性に関しては、ディスプレー・プロセデュア形式に較べてはるかに自由度の高いものになる。多くの言語はこの種の扱いをしているが、記述のレベルからこれを2分すると、次の様になる。

① 図形イメージの構造を明示的に記述する形式の言語

（例） GRINⅡ，AIDS

② 図形イメージを高度な表現形式で与える言語（図形イメージの構造は、このエクスプレッションの中で暗に定義されている）

（例）　GRAF　GPL／I

以下に各々についての扱いを解説する。

○　GRIN Ⅱ

GRINⅡで扱う図形構造は、リーフ、ノード、ブランチと呼ばれる3つの基本的な要素から構成され、各々は次の情報を蓄えている。

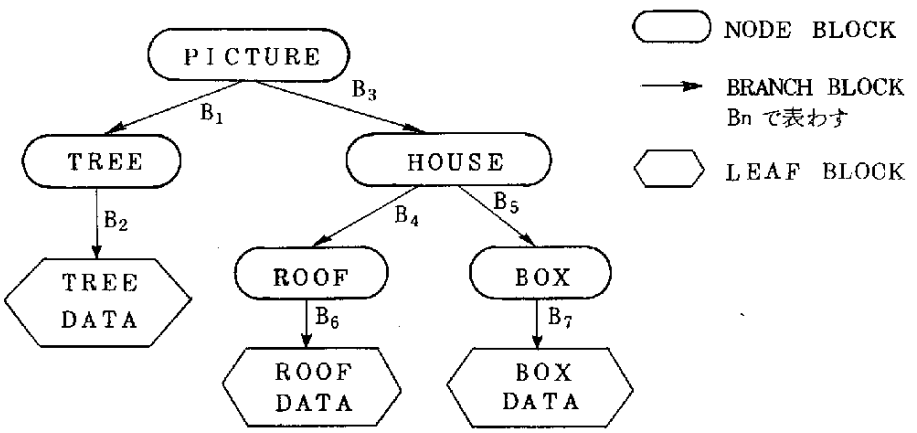
リーフ・ブロック……実際の図形データを蓄える。

ブランチ・ブロック……図形変換情報（位置情報、スケール情報）、表示のための情報（輝度等）を蓄える。

ノード・ブロック……識別名を持ち、構造化された図形を表わす。

ここで、図形イメージに対応するものは、リーフ、ノードであり、ブランチは、図形イメージの相互の関係とそれに加えられる操作を表現する要素と考えられる。（移動と、スケーリングしか許していない）

〔図3－6〕は、〔図3－1〕の例をGRINⅡのデータ構造として表現したものである。



〔図3－6〕

図形記述は、次に示す様なステートメントを用いて行なわれるが、図形記述をストラクチャー記述と云う形で行うため大変表現しにくいものになっている。

LEAF …… リーフ・ブロックの定義

NODE …… ノード・ブロックの定義

BRANCH …… ブランチ・ブロックの定義とノード間の結合

BRANCH …… ブランチ・ブロックの定義とノード間の結合

DETACH …… ブランチを取り除く

ATTACH …… ブランチをノードに付加する。

又表示は、対象とするノードを、ディスプレー・ノードと呼ばれる特定のノードにブランチで結合すると云う形で行なわれる。以下に、〔図3－1〕をGRINⅡで表現したプログラム例を示す。

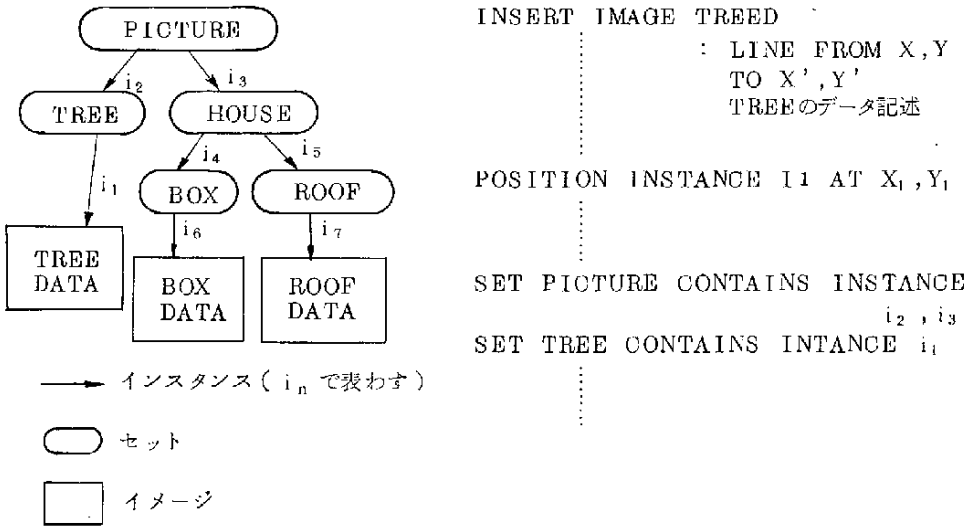
```
LEAF      TREED                    (TREEデータの記述)
VECTOR    (X1, Y1……)
LEAF      ROOFD                    (ROOFデータの記述)
VECTOR    (X2, Y2……)
      ……
(ROOFデータの記述)
      ……
(HOUSEデータの記述)
      ……
NODE      TREE《B2,,,TREED》       (リーフ、ブランチ、ノードの結合)
      ……
NODE      PICTURE                  (ノードの定義)
BRANCH    BI,,,,PICTURE            (ノード、ブランチの結合)
      ……
      ……
```

○ AIDS

AIDSも、GRINⅡとほぼ同じ構造を採用している。

GRINⅡでノード、ブランチ、リーフと呼ばれたものはAIDSではそれぞれ、セット、インスタンス、イメージと呼ばれる。即ち、セット、イメージが図形イメージを表現し、インスタンスは図形イメージの相互の関係と、それに加えられる操作を表わす要素として働く。(操作はやはり、移動とスケーリングしか出来ない)

〔図3－7〕は〔図3－1〕をAIDSの構造として表現したものである。又次のプログラム例は〔図3－7〕の構造をAIDSによって記述したものである。この過程は次の様に要約される。



〔図３－７〕

① イメージの定義 …… 実際の図形データを定義する。
（Insert IMAGE）

② インスタンスの定義 … イメージ又はセットを引用する時は，必要な位置情報を与えておく （Position INSTANCE）

③ インスタンスとイメージ又はインスタンスとセットの結合をはかる。
（INSTANCE Defines IMAGE）

④ セットの構造化（ＳＥＴ Contains）

この過程からも分る様にＡＩＤＳも、図形の記述性という点に関しては、かなり内部構造を意識する必要があり、余り記述性が良いとは云えない。

ＧＲＩＮⅡ，ＡＩＤＳがあえてこの様なストラクチャー記述の形式を採用した背景として次の様な事情が考えられる。

ＧＲＩＮⅡ、ＡＩＤＳでは図形データ構造の中に、プロブレム・モデルを編成する機能を備えている。即ちＧＲＩＮⅡではデータ・ブロック、ＡＩＤＳではレーベルブロックと呼ばれるプロブレム・データを扱うブロックがあり、各々は図形データ構造の中に組み込む事が出来る。この様な形でストラクチャーが編成されると、図形データ構造操作はそのまま問題向構造操作につながることになるから、問題向けデータ構造にアクセスする手段を残すためには、図形構造を記述する手段として，データ・ストラクチャー操作の機能をそのまま含めて置かな

くてはならない。

○　GPL／I

GPL／Iでは、プロブレム・データ構造と、図形データ構造を分離して扱っている。

又図形データ構造の在り方については、「ユーザからは見えないものでなくてはならない」という立場をとっている。即ち、ユーザには高度な図形の表現形式を与え、これを評価して内部構造を作り出すのはシステムの仕事であると考える分である。

GPL／Iに於て図形イメージを表現するグラフィック・データはイメージと呼ばれる。イメージはさらに基本的な図形データ、ベクトルによって、その実体（実データ）が定義される。その他イメージの実体を定義するグラフィック・データとしていくつかのグラフィック・ファンクション　TEXT、ARC等が用意されているが、余り細部にはふれない。

ベクトルは位置ベクトルの概念によくあてはまる。即ち、2つ又は3つのコンポネントで定義され、各々は2D又は3Dの位置情報を表わす。

例えば（1，5）、（6，9）を結ぶ線分はイメージLとして次の様に定義される。

L＝（1X＋5Y）|||（6x＋9y）（X、Yはベクトルの成分を表わし|||は結合を表わすオペレータである）。又イメージ、ベクトル等のグラフィック・データの間で定義されるグラフィック・オペレータとして次の様なものが用意されている。これは図形イメージに操作を加えるオペレータと考えられる。

グラフィック・オペレータの例

+>　イメージ間の併合

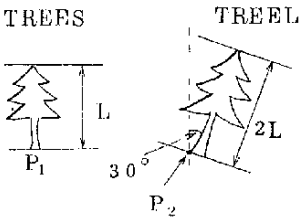
|||　イメージの結合

ⓐ　位置付け

<>　スケーリング

*>　座標軸のまわりの回転

例えば、〔図3－8〕で示されるTREESがイメージとして定義されている時TREELはこれを用いて次の様に表現される。



TREEL=《TREES<>(2X+2Y)*>(-30Z)ⓐP2

〔図3－8〕

この例にみられる様なグラフィック・データとグラフィック・オペレータで構成される図形イメージの表現形式をＧＰＬ／Ｉではイメージ・エクスプレッションと呼んでいる。イメージ・エクスプレッションの別の例として、次のプログラムを見てみよう。これは〔図３－１〕の図形構成をＧＰＬ／Ｉのイメージ・エクスプレッションで表わしたものである。

```
HOUSE   =ROOF  @  P1 +>  BOX @ P2
PICTURE =TREE  @  P3 +>  HOUSE @ P4
```

（Ｐ１ ～ Ｐ４はベクトルで各イメージの位置情報を与えている）

これ等の例からも分るように、ＧＰＬ／Ｉでは、図形の内部構造を全く意識することなく、意図する図形を高度な表現形式で表わす事が出来る。

ＧＰＬ／Ｉでは、この他にも色々便利な機能を備えている。例えばイメージには各種のアトリビュートを与える事が出来る。これは、イメージ・アトリビュートと呼ばれ、スタティクに定義されるものとダイナミックに変更出来るものがある。イメージ・アトリビュートは、今まで述べてきたイメージ・エクスプレッションがイメージの幾何学的性格を決定するのに対し、イメージの他の図形的特質（例えば、明るさとか色）を定義するのに使用される。（例えばＤＡＳＨＥＤ属性を持ったイメージを表示すると、破線で表示される）又イメージに適用されるＦｕｎｃｔｉｏｎも各種ありイメージに対して種々の操作が加えられる様になっている。

３－２　割り込み処理機能

割り込み処理は、グラフィックスに於けるインタラクティヴ・プロセスを実現するための本質的な機能である。

これは具体的には、ファンクションキー、ライトペン等の各種の割り込み要因に対して、処理プログラムの動作をどう対応ずけるかの方法の問題と云える。

以下では、各言語システムに於ける割り込み処理の方法について次の２つの観点からこれを見てゆく。

(1)　割り込みの処理形式

(2)　割り込み制御の方法

(1)　割り込みの処理形式

一般に、割り込みの取り扱い方に関しては、非同期処理と同期処理の２つの処理形式がある。

前者は、割り込みが入ると直ちにその処理に移る様な形態を指す。又後者の扱い方には、次の２つの処理形式がある。

① 割込みが入った時に、直接ユーザ・プログラムには割り込みをかけずに、システムがこれをキューイングしておく、そして必要な時点でユーザ・プログラムがこれを取り出す。

② プログラムが割り込み待ちの状態にある時のみ割り込み処理プログラムに制御を渡す。

本来インタラクティヴ・プロセスとは、結果を見てアクションを起すという形で進められてゆく性格のものであるからその処理形式としては同期処理を考えておけば十分の場合が多い。

しかし、ある種の状態を想定した時、例えばメインプログラムである処理を統行中にループに入ってしまったとか、あるいはある処理に時間がかかっていつまでたっても応答が返らないと云う様な事態が発生した時、一担この状態を脱して別の処理をしたいと云うような要求が起るかもしれない。この種の処理は非同期処理系でなくては実現出来ない。

しかし、一般に非同期処理を扱う系では、そこで使用されるプログラム・リソースにリエントラント性が要求されることから、既存の言語をベースとするグラフィック言語に於ては、この種の処理を実現する事は仲々困難になっている。又これ以前の問題として、非同期処理が可能かどうかは、前提とするグラフィック・オペレーティング・システムの性格によって決定ずけられてしまっている様な場合もある。(注1)

(注1) GRAFはEXPRESS GPSを前提にして作られているが、EXPRSS GPSでは割り込みをキューイングする形式をとっている。

(2) 割り込み制御の方法

前の項では、割り込みが発生した時のシステム動作の性格について述べた。

ここでは、システムに対して割り込み制御のための情報をどう云う形式で指示するか又それが内部的にどの様に扱われるかについて見てみる。

ここで割り込み制御のための情報とは具体的には次の様な内容を含んでいる。

① 割り込み抑制情報

各種の割り込み要因に対して、その割り込みを禁止するか、許容するか(受けつけるか受けつけないか)と云う指示。

② 割り込み要因とその処理プログラムとの対応関係をどう云う形で指示するかと云う事。

(注) ②については、システムは全然関知しない(即ちユーザが割り込み要因を調べそれに応じた処理をする)と云う例もある。

a　内部的な扱い

割り込み制御過程の表現は各言語ごとに色々な表現形式が試みられているが、その内部的な扱いに関してはほとんどのものが共通した方法をとっている。

〔図３－９〕、〔図３－１０〕はその基本的な扱いを図示したものである。以下に簡単に説明を加える。

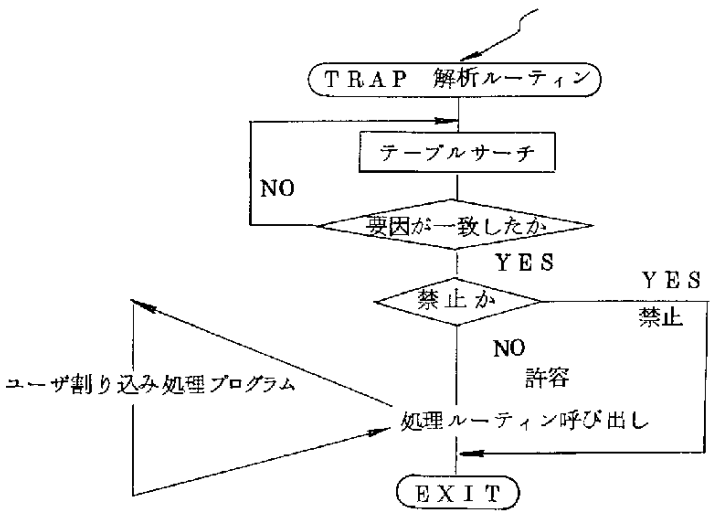
〔図３－９〕：ユーザから登録された、各種の割り込み制御情報は、要因ごとに整理され図の割り込み制御表に記録される。

〔図３－１０〕は、割り込みが発生した時の解析動作を示すものである。システムは作成した制御表をもとにその要因に対してどの様な動作をとるかを決定する。

割り込み制御テーブル

割り込み要因	禁止許容FAG	処理ルーティンアドレス
ライトペン（ＬＰＥＮ）	1	処理プロセス１のアドレス
ファンクションキー（ＦＫＥＹ）	1	処理プロセス２のアドレス
メッセージ（ＭＥＳＳＧ）	0	処理プロセス３のアドレス

〔図３－９〕



〔図３－１０〕

b　割り込み処理過程の表現

プログラム上、割り込み処理過程をどう表現するかと云う問題は結局、どういう形式で割り込み制御情報を登録するかと云う問題又同期処理の場合はキューイングされた割り込みの取り出し又は割り込み待ちの状態をどのような形式で表現するかという問題に帰着する。

この扱いを表現のレベルから分類すると次の様になる。

①　割り込み制御表を持た無いもの。

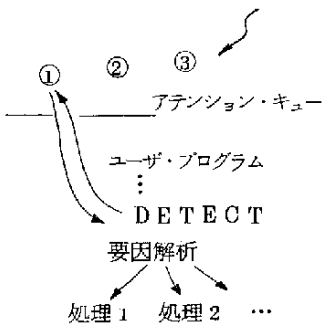
②　割り込み制御表を明示的に作成するもの。

③　割り込み制御表を意識させずにこれをプログラム・ステートと云う概念で扱ってゆくもの。

ここで、①は同期処理を扱う系でしか実現できないが②、③はどちらの系でも実現可能である。以下にこの①、②、③の各々につき簡単に例をあげて説明する。

①　割り込み制御表を持たないもの。

この例として、ＧＲＡＦを見る事が出来る。ＧＲＡＦでは、ユーザ・プログラムからシステムに与える指示は割り込みの抑制情報（禁止、許容）だけである。システムはこれに従ってアテンション・キューを作成してゆく。即ち、割り込みが発生しても、直接ユーザ・プログラムに割り込みをかけると云う事はしないで、これをアテンション・キューに登録しておく。（〔図３－１１〕参照）



〔図３－１１〕

ユーザ・プログラムからはＤＥＴＥＣＴ等の命令を実行する事により、アテンション・キューから割り込み情報を取り出す事が出来る様になっている。割り込み情報は、ユーザ・プログラムにより解析されそれに応じた処理がとられる。

②　割り込み制御表を明示的に作成するもの

ＧＳＰ、ＧＰＬ／Ｉ等では割り込み要因と処理ルーティンの対応をシステムに登録するための命令が用意されており、ユーザはこれによって割り込み制御表を作成する。

◦　ＧＰＬ／Ｉ

ＧＰＬ／Ｉでは、ＰＬ／ＩのＯＮ文を拡張して、これを表現している。

即ち　ＯＮ　要因　ＢＥＧＩＮ；処理ルーティン　ＥＮＤの形式で割り込み要因と処理ル

ーティンの関係を登録する。

例えば、〔図３－９〕の制御表は次の一連のＯＮ文を実行することにより作成される。

（例）

ＯＮ　ＬＰＥＮ　ＢＥＧＩＮ；処理プロセス１　ＥＮＤ；

ＯＮ　ＦＫＥＹ　ＢＥＧＩＮ；処理プロセス２　ＥＮＤ；

ＯＮ　ＭＥＳＳＧ　ＢＥＧＩＮ；処理プロセス３　ＥＮＤ；

又、割り込みの禁止、許容はラベル接頭語によって制御している。例えば、次に示される手続きブロックに制御が移った時このブロック内ではライトペンとファンクションキー割り込みが可能である。

（例）

（ＬＰＥＮ　ＦＫＥＹ）：ＥＸＡＭＰＬＥ：ＰＲＯＣＥＤＵＲＥ　ＯＰＴＩＯＮＳ（ＭＡＩＮ）

⋮

又割り込みはキューイングされ、ＴＡＫＥ文又はＷＡＩＴ文によってこれを取り出す方法が採用されている。

前者はアテンションがなければ、そのままＴＡＫＥ文以下の処理が続行されるのに対し後者はアテンションが入るまでプログラムを待ち状態に置く。

○　ＧＳＰ

ＧＳＰも、サブルーティン・コールの形式で、割り込み要因と処理ルーティンの関係を登録する形式のものが多い。

詳しくはＧＳＰの項を参照されたい。

（例）

ＣＡＬＬ　ＳＥＴＵＰ（ＬＰＥＮ，ＲＯＵＴＩＮＥ１，ＦＫＥＹ，ＲＯＵＴＩＮＥ２，ＭＥＳＳＧ，ＲＯＵＴＩＮＥ３）

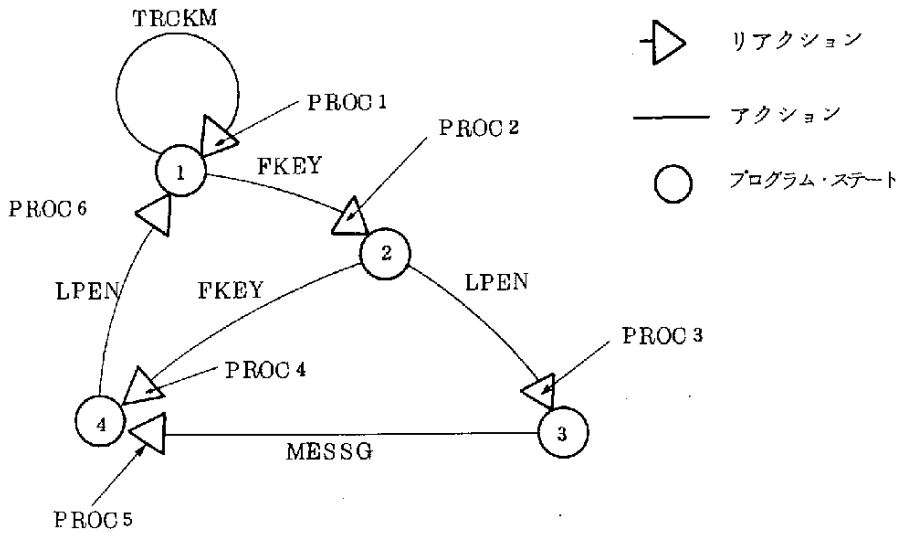
③　割り込み制御表を意識させずにこれをプログラム・ステートという概念で扱ってゆくもの。

ユーザに対しては直接、割り込み制御表を意識させず、割り込み制御表によって支配されるプログラムの状態にプログラム・ステートと云う概念を与え、プログラム・ステートをアクション、リアクションという観念で定義させようとする試みである。これはＷ.Ｍ.ＮＥＷＭＡＮ等の考えに基くもので、基本的にはインタラクティヴ・システムを有限の状態を持つオートマトンと考えアクション、リアクションをそれぞれオートマトンへの入出力として扱おうとする考え方である。

ＤＩＡＬ、ＡＩＤＳ、ＭＥＴＡＶＩＳＵ等はこの様な扱いをしている。一般に、プログラ

ム・ステートの遷移を表現するために、〔図３－１２〕の様なステート・ダイヤグラムが使用される。

次に示すプログラムは、〔図３－１２〕のステート・ダイヤグラムをＤＩＡＬによって表現した例である。



〔図３－１２〕

```
DURING 1    DO   BEGIN ;
          ON TRCKM DO BEGIN PROC1 ENTER 1 END ;
          ON FKEY DO BEGIN PROC2 ENTER 2 END ;
                END ;
DURING 2    DO   BEGIN ;
          ON LPEN DO BEGIN PROC3 ENTER 3 END ;
          ON FKEY DO BEGIN PROC4 ENTER 4 END ;
                END ;
DURING 3    DO   BEGIN ;
          ON MESSG DO BEGIN PROC5 ENTER 4 END ;
                END ;
DURING 4    DO   BEGIN ;
          ON LPEN DO BEGIN PROC6 ENTER 1 END ;
                END ;
```

3－3　データ・ストラクチャー操作機能

グラフィックに於けるデータ構造の研究は、主として図形データの構造化と問題のモデリングと云う２つの面からの要請によって進められてきた。

グラフィック言語に於て、この２つの構造、即ち図形構造と問題用構造をどう扱うかと云う点に関しては、一般に次の様な扱い方が望ましいとされている。

- 図形構造と問題用構造は分離して扱う。
- 図形構造はシステムに内在するユーザからは見えない構造として扱う。
- 問題用データ構造は、どんな問題でも表現し得るような汎用性の高いものでかつ、それを扱う言語は優れた表現形式を備えていなくてはならない。

(1)では、この様な傾向が出てきたいきさつを知る意味で、モデリングが具体的にどのように進められるかを解説し、現在までに試みられたいくつかのアプローチの例を紹介する。

(2)では、現在までに開発されてきた各種のデータ・ストラクチャーについてそれ等を一つの流れの中で見る事により、相互の共通点、相異点を明らかにしてゆくことにする。

(1)　構造化の方法

a　構造化の立場

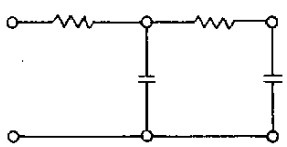
我々が実際に、ＣＲＴに図形を表示し、それに何等かのインタラクションを加えながら、問題を解析してゆく過程を考える時、その対象とする系を表現するためには、次の３つの立場から、構造化の要求が起きてくる。

①　図形イメージの構造化

②　図形の論理的なグルーピング

③　問題のモデリング

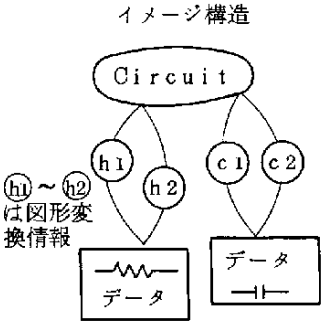
例えば、例として次の回路図を対象に回路解析を行なう場合を考えてみよう。



〔図３－１３〕

対象とする系をＣＲＴに表示するためには、まず図形データの定義から始めなくてはならない。この時、回路図を一つ一つ独立に記述していったのでは大変非能率であるから、通常はその中の基本的なパターンをいくつか定義しておき、それに適当な操作（回転、移動等）を加えながら組み立ててゆくと云う方法をとる。この様な形式で図形データを扱うためにはこれが適当に内部構造化されていなくてはならないだろう。①はこの意味での構造

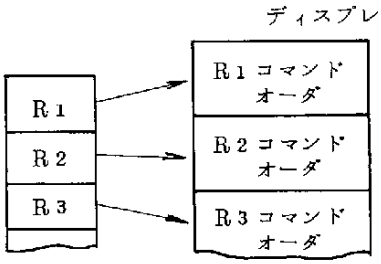
化の必要性を云っている。つまり、図形の記述性、操作性の要求から来る図形の内部表現形式と云い替える事が出来る。これは一般にクロス・コネクションを持つ木構造として表現され、イメージ構造等と呼ばれる。〔図３－１４〕は〔図３－１３〕をイメージ構造として表現した一例である。



〔図３－１４〕

次に、定義された図形データを表示する段階を考えて見る。例えば〔図３－１４〕を表示して〔図３－１３〕の回路図が表示されたとする。この時、回路図の一部を指定して、それに何等かの操作を加えようとすれば、指摘された部分が回路図のどの部分であるかを計算機に知らせるための何等かの方法を講じておかなくてはならない。一般にライトペン等を備えたＣＲＴでは、図形が指摘された時、計算機に知らされるのは指摘された部分に対応するディスプレー・ファイル上のコマンドのアドレス又はあらかじめその図形要素をあらわすコマンドの集合につけられたフラグの番号である。

いずれにしろ、計算機にどの図形要素が指摘されたかを教えるためには図形要素とそれを表わすコマンドの集合との間に対応関係がつけられていなくてはならない。この対応表はコリレーションリスト等と呼ばれ〔図３－１５〕の様な構造をとる。



〔図３－１５〕

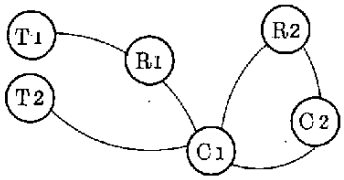
ここで、計算機が〔図３－１４〕を対象にして〔図３－１５〕の形式の表示データを作成する事を考えてみよう。〔図３－１４〕の構造をトレースすれば確かに〔図３－１３〕の回路図を構成する図形的情報は得られるだろう。しかし、その中の図形要素が個々独立なものか又はいくつか集って１つの図形要素を構成しているのかと云う点については何も知る事は出来ない。（例えばＲ１，Ｃ１が個々独立なものか、Ｒ１とＣ１で一単位になるのか、それとも回路全体が一単位なのかは、このままの構造では分らない）つまり〔図３－１４〕の構造

では不十分でありさらにその中に論理的なまとまりを指示するような構造を組み込まなくてはならない。これが②の意味するところである。

次にこの様にして、図形データが定義され、表示された回路を対象に回路解析を行なう場合を考えて見る。これまでの過程で満たされた事は回路図が画面に表示され、その中の一つの回路素子に働きかけるとそれが回路のどの部分かを認識出来るという段階までである。回路が一つの物理系として、どの様に構成されているかと云う情報については、まだ知らされていない。

ここで、さらに問題解析のためのもう一つの構造が必要となる。例えば〔図３－１６〕の回路のトポロジカル・モデルがその例である。③の問題モデリングとはこうした立場からの構造化の要請である。

回路のトポロジカル・モデル



〔図３－１６〕

b　構造の扱い

ここでは先に述べた３つの構造即ち、イメージ構造、論理構造、問題構造が実際にどの様に扱われているかを見てゆこう。

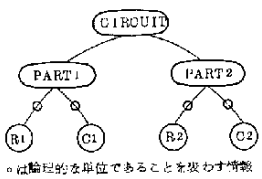
○　論理構造の扱いについて

論理構造の扱いは、それ程難しい問題ではない。この扱いについては、次の２つの扱い方が一般的である。

①　イメージ構造の中に組み込む。

これはイメージ構造の中に、さらに論理構造も表現する方法である。ＧＰＬ／Ｉ等がこの扱いをしている。ＧＰＬ／Ｉではイメージ構造はイメージ・エクスプレッションを通して定義される。この時、論理的な単位は表示されたイメージを構成する、最下位のイメージと云う風に約束がされている。例えば〔図３－１４〕が次の様なイメージ・エクスプレッションで与えられた時、これを表現するイメージの内部構造は〔図３－１６〕の様になり、約束に従って

PART1＝R1＋C1
PART2＝R2＋C2
CIRCIT=PART1+PART2



〔図３－１６〕

〔図３－１６〕の最下位のイメージＲ１、Ｒ２、Ｃ１、Ｃ２が各々論理的な識別対象となる。

一方これをＰＡＲＴ１、ＰＡＲＴ２を単位にして識別する必要があるかも知れない。この時ＰＡＫＴ１、ＰＡＫＴ２を最下位のイメージになるように意識しながらイメー

ジ・エクスプレッションを書くのでは、図形の記述法に制約を加える事になるだろう。GPL／Iでは、特にこの目的のためにインスタンス・ファンクションと呼ばれるイメージ・ファンクションを設けて、イメージ構造に細工を加える事が出来る様になっている。〔図3－17〕はインスタンス・ファンクションにより論理的な構造を変えたものである。

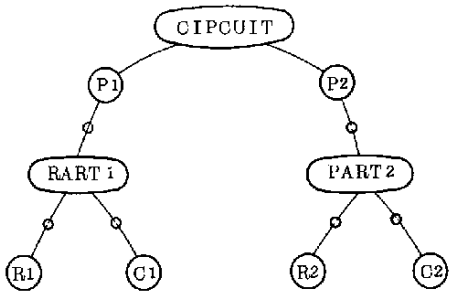
PART1 = R1 + C1

PART2 = R2 + C2

P1 = INSTANCE(PART1)

P2 = INSTANCE(PART2)

CIRCUIT = P1 + P2



〔図3－17〕

② イメージを出力する時に指定する方法

これは、出力ステートメントで指定された単位を1つの論理的単位として扱う方法である。

DIAL、METAVISU、GRAF等がこの様な扱い方をしている。例えば、〔図3－16〕の様な形でイメージ構造が組まれている時CIRCUITを指定して、出力するとCIRCUITが一つの論理的な単位として扱われる。又R1、R2、C1、C2を各々一つの論理的な識別対象とするためには、各々を指定して4回に分けて出力すればよい。

（例）

- CIRCUITが論理的な単位として扱われる。

```
CALL  CIRCUIT  AS  IDC
```

- R1、R2、C1、C2が各々一つの論理的な単位として扱われる。

```
CALL  R1  AS  IDR1
CALL  R2  AS  IDR2
CALL  C1  AS  IDC1
CALL  C2  AS  IDC2
```

- 問題構造の扱いについて

これまでの考察でイメージ構造、論理構造の扱いにはそれほど大きな問題がない事が分った。以後はこれらをまとめて図形構造と呼ぶ。

次は問題構造の扱い方について考えてみよう。問題構造と図形構造の扱い方については次

の3つのアプローチが可能である。

① 問題構造と図形構造を同一のデータ構造の中に表現する。一つの構造の中に、問題用の情報と図形情報を両方持たせようとするアプローチである。この例としてＧＲＩＮⅡ、ＡＩＤＳがある。

例えば、ＧＲＩＮⅡに於ては、データ・ストラクチャーを構成する基本的な要素として、リーフ・ブロック、ノード・ブロック、ブランチ・ブロック、データ・ブロックと呼ばれる4つのブロックが用意されていて、ユーザはあらかじめ決められた規則（例えばデータ・ブロックに問題用データ、リーフ・ブロックには、図形データという風に）に従って図形データ、問題用データを適宜ブロックに格納し、適当な構造化ステートメントを用いてこれを構造化する（お互いの関連をつける）と云う形式をとっている。

この様にして編成されたデータ・ストラクチャーはその表現に無駄がなく、又図形情報によるインタラクションが同時に問題用モデルに反映されるというメリットを持つと考えられる。しかし、構造化は、問題モデルの表現を中心にして進められなくてはならないから、事実上図形を扱うための便宜的な構造化即ちイメージの構造化はあきらめなくてはならない。（図形の記述性が悪くなると云う事）図形情報はあたかも問題情報の一部としての扱いを受ける分である。又図形情報、問題情報をデータ・ストラクチャーに編成する作業が困難になることは云うまでもないだろう。

② 問題構造、図形構造を分離し、各々のデータ・ストラクチャーを別々に備える。

図形データ構造は1つの専用データ・ストラクチャーに受け持たせ、これはユーザからは見えない存在とする。（つまりユーザがハイレベルの図形表現をすることによりシステムがこれを内部構造化して扱う）

一方問題用データ構造に対しては汎用性のあるデータ・ストラクチャーを備え、記述性の良い構造化ステートメントを使う事によりユーザが自由に対象とする問題を表現出来るような便宜をはかる方法である。現在、このアプローチは最も実際的であるとされている。この典型的な例としてＭＥＴＡＶＩＳＵを見る事が出来るだろう。その他問題用データ・セットを言語に含んでいないがＧＰＬ／Ｉ、ＤＩＡＬ、ＧＲＡＦも方向としては同じである。

③ 図形構造、問題構造は分離して扱い、問題構造の方はシステムに作らせる。

ディスプレー・ファイルは図形の位置情報の集りであるから、これをトレースすれば、図形要素間のつながりがある程度分るはずである。この操作をシステムにやらせようという分である。例えば、ＤＩＭと呼ばれるアプリケーション・システムではこの様な方法を採用してい

る。しかし現実には、図形要素の位置情報が与えられても、要素間のトポロジカルな関係を把握するのは非常に困難な場合が多い。（たとえば非常に接近した2点は、つながっているのか、離れているのか判断に苦しむ）又ディスプレー・オーダをトレースする作業は、インタラクションがあるたびにくり返さなくてはならない（それが問題モデルにどう影響するかを調べなくてはならない）から大変非効率である。この様に考えると、一般の系を対象にした時、この様な機構が実現される可能性は非常にうすいと考えてよいだろう。

(2) データ・ストラクチャーの発展過程

(1)では、主として対象とする系をデータ構造に編成する上での問題についてふれてきた。

この項では、これまでに開発されたデータ・ストラクチャーを中心に、データ・ストラクチャー自体の問題点について考えてみる。

a 専用データ構造と汎用データ構造

より基本的な問題から話を進めると、システム・ファシリティーとしてのデータ・ストラクチャーの在り方に関して従来2つの観点があった。

1つは、個々のアプリケーション毎に、オプティマイズされたデータ・ストラクチャーを容易に編成する事が出来るような環境を作り出すのがシステム・サポートとしての使命であり、ユーザは各アプケーション毎にそれに適した専用データ・ストラクチャーを編成すべきだとする見方である。つまりシステムは、データ・ストラクチャーを任意に構成出来る様な基本的な機能（例えばフリー・ストレージ管理）のみを提供する分である。AED、DSPL等はこのアプローチの例であると云われている。もう1つは、問題に依存しない謂わゆる汎用のデータ・ストラクチャーの開発を実現すべきだとする方向である。

これまでに開発されてきた多くのデータ・ストラクチャー・パッケージはこのアプローチと見る事が出来るだろう。

この両者のアプローチの是非に関しては、前者では効率の良いシステムが実現出来るがアプリケーション毎にデータ・ストラクチャー・パッケージを作成する作業が課せられると云うこと、又後者にあっては、一般に汎用性と処理効率の相反する性格から、系のオプティマイズが困難になること、等の事情があり、どちらのアプローチが本来の姿であると云う確定した見方はない。

当面、グラフィック言語が対象とするデータ・ストラクチャーは後者であるから、今後の話は後者に限って進めてゆく。

ここでデータ・ストラクチャーの発展と題したが、これはあくまで汎用化の方向と云う意味で

の発展であって、個々のデータ・ストラクチャーのどちらが優れているという意味は含まれていない。又汎用性に関して、具体的に次の様な機能を考えておけばよいだろう。

① 関連の表現性に関する一般性

② 扱うデータ量に関する一般性（２次記憶への拡張性）

b 用語の概念

データ・ストラクチャーに於ける構成要素の呼び名は各システムごとに独自につけられて居り、同一概念を表わす用語が他のシステムでは別の名称で呼ばれる事が多い。まぎらわしさを避けるためにここでは普偏的な呼び名、アイテム、カテゴリ、アソシェーションを次の様に定義しておく。

アイテム：構造化の対象となる要素、独自のデータの他、カテゴリー、アソシェーションを持つ。

カテゴリ：アイテムのカテゴリーを表わす部分

アソシェーション：他のアイテムのカテゴリーに属することを表現する部分

又、用語の対応表を次にかかげておく

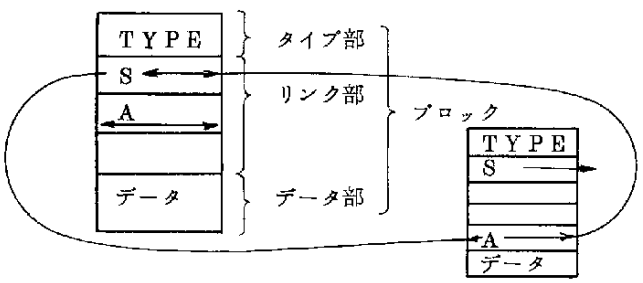
用語の対応表

要　素	SKETCHPAD	CORAL	APL	ASP
アイテム	ブロック	ブロック	エンティティ	エレメント
カテゴリ	リング・スタート(ヘン)	スタート・エレメント	サブセット	リングスタート
アソシェーション	リング・タイ(チキン)	エレメント	アソシエーティブセット	マソシェータ

c RING STRUCTURE

グラフィックスに於けるデータ・ストラクチャーの発展過程に於て、その典型的な存在として、リング・ストラクチャーを見ることができる。

今までに考案されてきた各種のデータ構造も多くは（SKETCHPAD，CORAL，APL，ASP）その基本をリング構造に置いている。



〔図３－１８〕
SKETCH PADに於ける
RING STRUCTURE

〔図３－１８〕は、その初期に現われたリング構造の例で、その最も単純化されたフォームを表わしている。ここでリング構造に於ける構造化の様式についてまとめると、それは次の様に要約されるだろう。

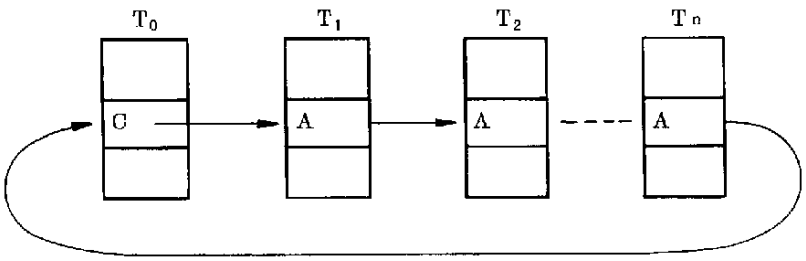
① 構造化の対象となる要素 アイテム にブロックを対応させる。

② アイテムは、いくつかの カテゴリー を持ちこれは図のリング部の一部（Ｓと記した所）によって表現され、リング・スタート等と呼ばれる。

③ アイテムは又、他からの アソシエーション （他のアイテムからの参照、厳密には他のアイテムのカテゴリーに属すること）を表現するため、やはりリンク部の一部（Ａと記した所）をこれにあてる。これはリング・タイ等と呼ばれる。

④ その他、アイテムはデータ部、属性部（タイプ部）を保有し、そのアイテム独自のデータを蓄える。

⑤ アイテム間の関係は、一つのアイテムＩのカテゴリーＣを出発点として、他のアイテム（ブロックのアソシエーションを表現するリンク部）I_1 ……I_n をつらぬき再びもとのカテゴリーにもどってくるリングとして表現される。（〔図３－１９〕参照）

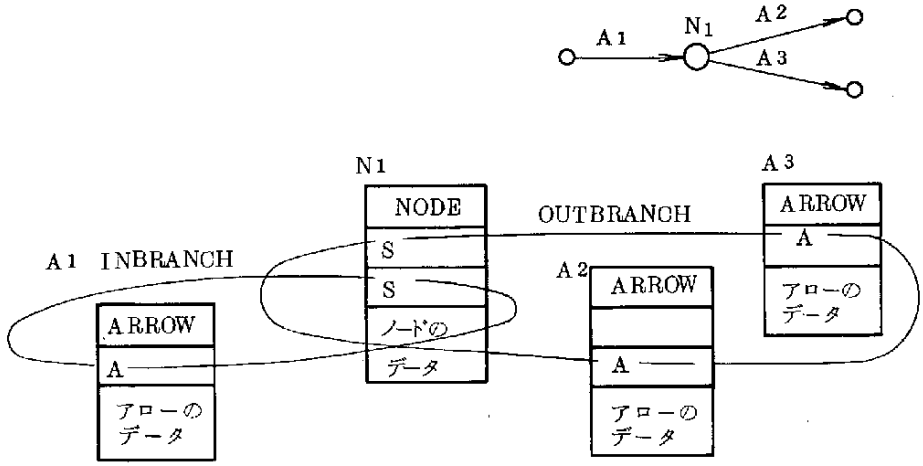


〔図３－１９〕

この時アイテム I_1 ……I_n は、アイテム I_0 の（カテゴリＣに関する）メンバーであると云われる。

⑥ リング・リストは一般に両方向リストとして表現され、メンバーの挿入、削除、検索が効率よく出来る様になっている。

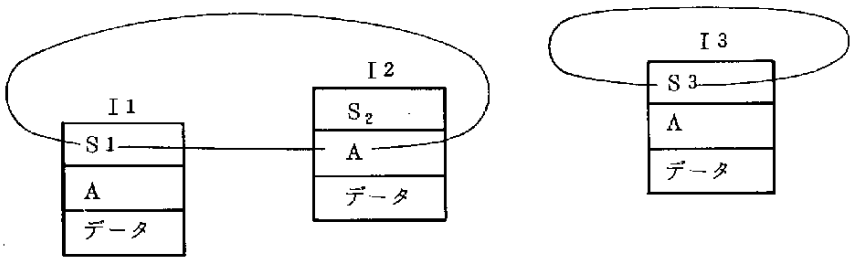
〔図３－２０〕はネット・ワークの一部をこの構造化様式に従って表現したものである。ノード、アローを各々１つのアイテムと考え、ノードはＩＮＢＲＡＮＣＨ、ＯＵＴＢＲＡＮＣＨと云う２つのカテゴリーを持つ。アローはアソシエーションのみを持つ。



〔図３－２０〕

この様な構造化様式に従ってデータ・ストラクチャーが編成される時、一般に次の様な問題がおきてくる。

① 一つは動的なカテゴリー、アソシェーションの追加に際してリンク部がそれを表現しきれなくなる可能性があると云うことである。例えば、〔図３－２１〕に於てアイテムＩ１、Ｉ２の間に図の様な関連が定義されていたとする。この時アイテムＩ３とＩ２の間に新たな関連（Ｉ２はＩ３のＳ３に属する）を定義する必要が生じたとすると、Ｉ２は新たにこれを



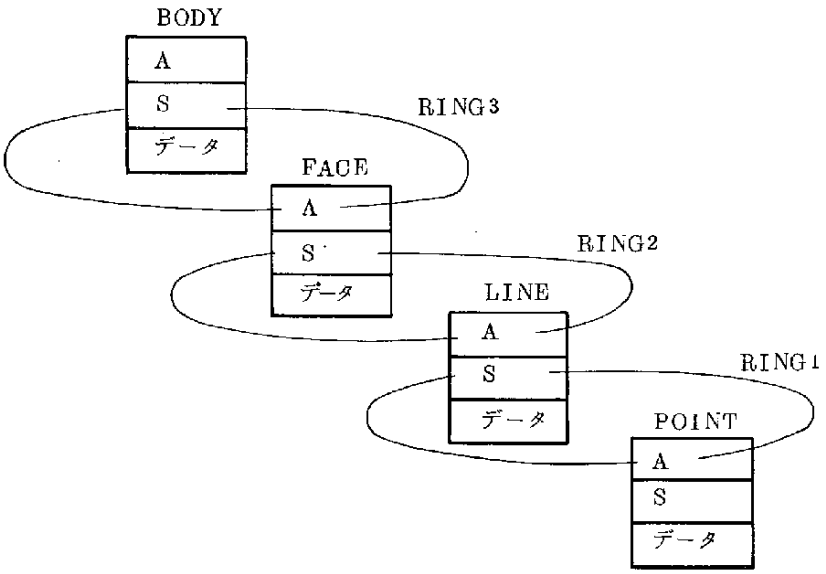
〔図３－２１〕

表現するためのアソシェーション用の領域を必要とする。この時リンク部に空の領域が無かったらどうなるであろうか。結局新たにＩ２を表現するブロックを作り直さざるを得ないだろう。

② さらにもう一つの問題は、あるリング・リスト上のアイテムが自己の属するリングの性格を知るためには一つ一つリングをたどってそのリング・スタートを見出さなければならないと

云う問題である。これは、リング構造の本来の性格であるには違いないが、特にアイテム間に複雑な階層関係が定義されている時には、その上位のアイテムを知るために必要とするポインタ操作は大変な手間になる。例えば〔図３－２２〕の様な例を想定してみよう。

〔図３－２２〕は、３次元物体の構造をリング・ストラクチャーで表現したものである。（リング上のメンバーは勿論複数個ある。ここでは便宜上単数で表現している）



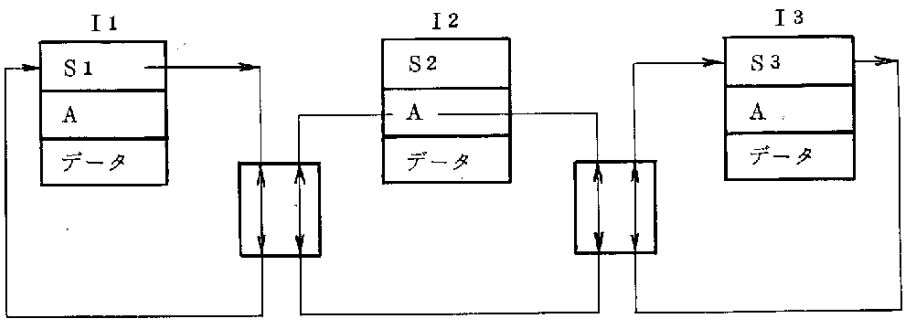
〔図３－２２〕

ここで、アイテムＰＯＩＮＴがどのＢＯＤＹに属するかと云う事を知る必要が生じたとすると、これを知るためにはＲＩＮＧ１から始めて順次ＲＩＮＧ２……ＲＩＮＧ３を巡回し上位のアイテムを見出していかなくてはならないだろう。

ＳＴＥＣＨＰＡＤ→ＡＳＰに到るリング・ストラクチャーの研究は主としてこの２点をその中心的な課題として扱ってきている。以下にその過程について簡単にふれておこう。

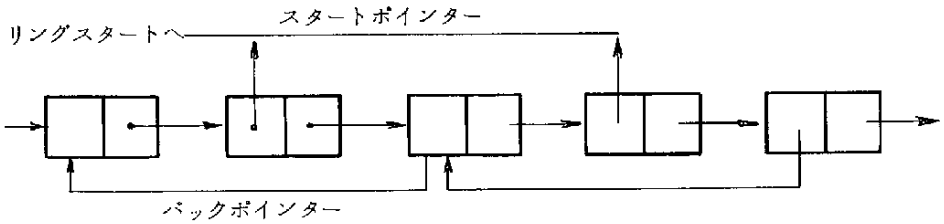
①　ＣＯＲＡＬ

ＣＯＲＡＬでは、アソシエーションの動的な追加に対して、ＮＵＢと呼ばれるコネクター・ブロックを設けることにより、これを可能にしている。（しかしカテゴリー（リング・スタート）の動的な追加と云う点に関しては、考慮を加えていない）例えば〔図３－２１〕に於ける問題はＮＵＢを使用することにより〔図３－２３〕の様に表現することが出来る。



〔図3－23〕

又②の問題に答えるために、即ち、リング・リストの任意の位置から、直接そのリングの始点を知る事が出来る様にするために、リング・リストを〔図3－24〕の様に（バック・ポインター、スタート・ポインターを交互に使用する）構成している。

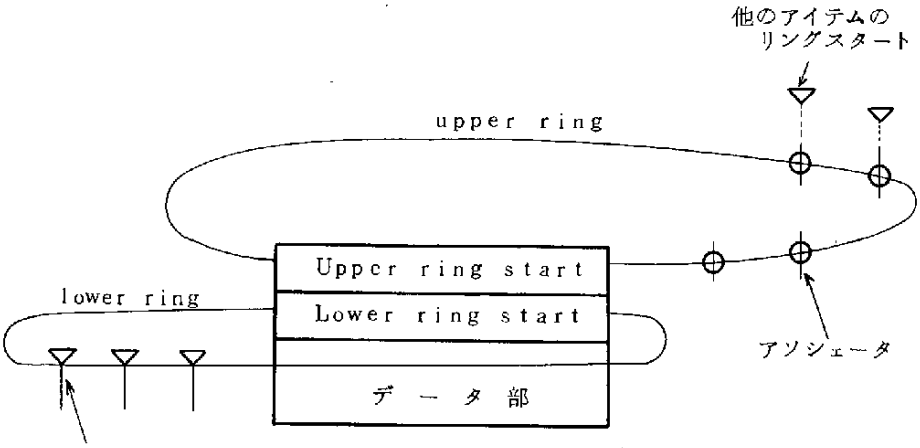


〔図3－24〕

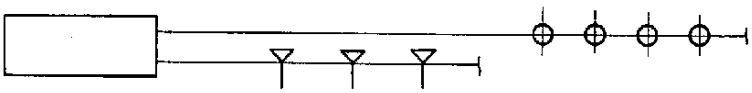
② ASP

ASPではカテゴリー、アソシェーションをそれ自身リスト構造（リングを益す）にすることにより両者の動的な生成を可能にしている。

ASPに於ては、アイテムは複数個のリンク部を持つ代わりにLOWER RING、UPPER RINGと呼ばれる2つのリング・スタートを持つ。そして各々はそこを始点とするリングを構成しその上に任意個のリング・スタート（カテゴリー）とアソシェータ（アソシェーション）を持つ事が出来る様になっている。これ等の一般的な関係は〔図3－25〕に示す。又〔図3－26〕はこれをASPノーテーションで表現したものである。

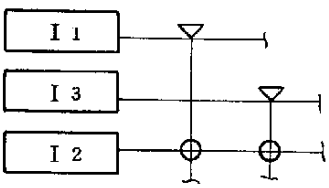


〔図３－２５〕



〔図３－２６〕〔ＡＳＰノーティションによる記述〕

例えば〔図３－２１〕に於ける問題はＡＳＰでは次の様に表現する事が出来る。



〔図３－２７〕

又アソシェータはアソシェーティヴ・リング、アッパー・リングの他にリング・スタートへのポインターを格納する領域を持ち、リングをたどらなくても直接リング・スタートに到達出来るようにして検索効率を上げている。

この様にして、リング・ストラクチャーに於ける関連の表現性の問題、即ちその動的な拡張性と云う点に関しては一応の解決をみた。

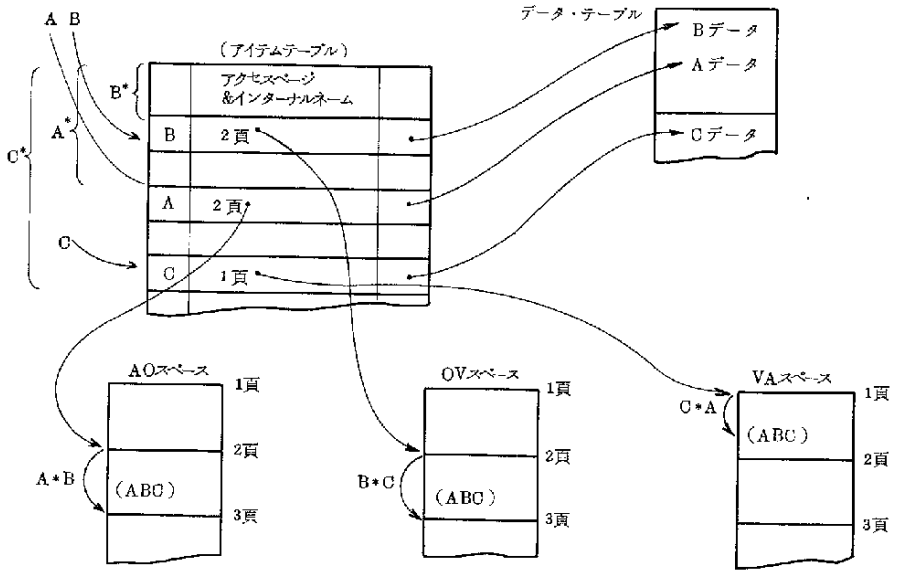
しかし、さらに汎用性としてもう一つの要請、即ちストレージの２次記憶への拡張性と云う点に関して、そのリスト・オリエンティドな性格から大きないきづまりを見るに到った。

d　ＨＵＳＨ構造

先に述べたリング・ストラクチャーがその構造化の原理をリスト構造に置くのに対し、これは全く別の行き方、即ち、ハッシュ技法をその構造化の原理とする。

ここでは、構造化に際して、アイテム、カテゴリー、アソシエーションと云う構造化要素を区別する概念は全く無い。単に、構造化の対象としてアイテムのみを考える。そして他のアイテムとの関連は、３つのアイテムを単位として表現する。即ちこれをオーダード・トリプルの形で記憶することにより、関連を保持する。例えばA、B、Cの親子関係（CはBの子供、BはAの子供）を表現するには、新たにＣＨＩＬＤと云うアイテムを設け、CHILD・A＝B、ＣＨＩＬＤ・B＝Cと表現しても良いし、単にA・B＝Cと云う三つ組で記憶し、その順序性に意味を与えても良い。

〔図３－２８〕は、A・B・Cの３つのアイテムの間に定義された関係＜A・B＝C＞がどの様に内部表現されるかを示したものである。アイテム・テーブルは、アイテム名称と、データ・ポインター、インターナル・ネーム、アクセス・ページ（それをアクセス・ワードとする時の格納頁）の対応表である。アイテム・テーブルに於けるアイテムの格納場所自身もそのアイテムをハッシングすることにより決定される。



注　A＊はAにハッシング操作を与えること
　　A＊BはA，Bでダブルハッシングを行なうこと。

〔図３－２８〕　連想記憶の機構

次に、＜ＡＢＣ＞のオーダード・トリプルの格納であるが、これは各項をアクセス・ワードとして、図のＡ－Ｏ、Ｏ－Ｖ、Ｖ－Ａの３領域に対して重複して記憶される。

Ａ－Ｏ、Ｏ－Ｖ、Ｖ－Ａスペースは各々固定長サイズでページ化されている。Ａ－Ｏは第１項を、Ｏ－Ｖ、Ｖ－Ａは、それぞれ２項、３項をアクセス・ワードとした時、そのトリプルが格納される領域で、どの頁に格納されるかは、アクセス・ワード（インターナル・ネーム）によって決定される。

Ｏ－Ｖ領域を例にとれば、アイテム・テーブルよりＢのアクセス・ページを知り、Ｂ＊Ｃのダブルハッシングを行なうことにより、その頁内に於けるトリプル格納位置を決定する。アクセス・ワードは一頁に一つとする方法と複数個置く方法があるがスペース効率から、通常は後者がとられている。この時、同じ頁内で、同一アクセス・ワードを有するトリプルは、一つのリストにつながれて居り、１アイテムだけによる検索を可能にしている。又トリプル格納に際しては、次の２つの例外ケースがある。

① コンフリクト …… ダブル・ハッシングの結果、異なるトリプルが同じ頁内位置にマッピングされた時で、これは通常リストを使って、例外処理を行う。コンフリクトが頻繁におこるとサーチ効率を落すことになるからなるべくおこさないような機構を考えなくてはならない。

② マルチプル・ヒット …… ２項を共有する２つ以上のトリプルがある時、当然の結果として同一アドレスにマッピングされる。これもやはりリストを用いて例外処理を行なう。

この様に＜ＡＢＣ＞の三つ組に対して編成される記憶機構は、一般に次の形式の検索を可能にする。（ＳＡＦと呼ばれる）

Ｆ１	Ａ（Ｂ）＝Ｃ
Ｆ２	Ａ（Ｂ）＝？
Ｆ３	Ａ（？）＝Ｃ
Ｆ４	Ａ（？）＝？
Ｆ５	？（Ｂ）＝Ｃ
Ｆ６	？（？）＝Ｃ
Ｆ７	？（Ｂ）＝？
Ｆ８	？（？）＝？

（Ｆ１は三つ組の存在の有無に対する質問、Ｆ８は連想記憶の全ダンプを意味する）

これは、＜ＡＢＣ＞の三つ組で定義された関係のすべてに対する、あらゆる角度からの検索が可能であることを意味し、その関連表現が検索に対していかに優れた性格を持つかを示している。又アクセス・ワードから、アクセス・ページを決定する方式は、ページング・エンヴァイロンメントに於て、一回の二次記憶アクセスでその検索が可能であることを示し、リング・ストラクチャーに於て問題になった多頁にわたるリスト検索の問題は、この機構では完全に問題外になっている。

この種の構造を採用しているデータ・ストラクチャー・パッケージとして、代表的なものにLEAP、ＴＲＡＭＰ、ＥＤＳＰ（日本）等がある。ＬＥＡＰでは、連想三つ組みの表現の他に、セット（アィテムの集合）の表現と、それに関する簡単な処理機能を備えている。又3つ組を一般のOrdered-n-tuple にまで拡張して、関係を定義し、それに集合演算が出来るような方式も研究されている（D.L.CHILDS）

以上に述べてきたように、この種の構造は、関連の表現力、検索能力、２次記憶スペースへの拡張性等に於てはかなり優れた力を持つが内部的にはまだ検討されるべき問題をいくつか残している。その主なものを挙げると次のようになるだろう。

① 頁内に於けるコンフリクトを少なくするにはどう対処すれば良いか。

② 頁内に、トリプルを格納し切れなくなった時、即ち、ページ・オーバーフローが起きた時にはどう対処するか又は、起さないようにするにはどのような管理アルゴリズムが必要か。

③ ＴＲＩＰＬＥを生成したり、消去したりする時、３領域にわたって、２次記憶をアクセスするのは非効率である。これをうまく処理するアルゴリズムは無いか。（これは、その時点で、生成、消去を行なはず生成、消去、コマンド・リストとして主記憶に記憶しておき、検索が実行される時、ページ・アクセスを起しこれに先だって修正を行なう等の方法が考えられている。）

以上データ・ストラクチャーの発展過程を主として汎用性（関連の表現力、２次記憶への拡張性）という立場から眺めてきた。

一般に云えることであるが、汎用化の方向はその内部構造を複雑化し、スペース、処理効率を低下せしめるのが常である。こうした観点からすると、各データ・ストラクチャーの比較に於て、いずれが優れているかと云う事は一概に云えない。

例えば、ある種の問題に対しては、最も単純なＲＩＮＧ構造で十分な表現が出来るかもしれない。この時あえて複雑な構造を採用すれば、領域の浪費を招き処理効率を落し、かえっ

て好ましくない結果を生むことになるだろう。

以下に、表現力、処理効率、必要スペース領という点から与えた優劣の関係をかかげこの項のむすびとする。

① 表 現 力

劣 → 優

$$\mathrm{SKETCHPAD} \rightarrow \begin{matrix}\mathrm{CORAL}\\ \mathrm{APL}\end{matrix} \rightarrow \mathrm{ASP} \rightarrow \mathrm{LEAP}$$

② 処 理 効 率

対象とする系をデータ・ストラクチャに編成する方法が一定的に定まらないから比較は困難である。又2次記憶も含めた時の振舞いも考えると評価はますます困難になるだろう。主記憶装置内で動作する時に、状況を限ればおよそ次の様になることが予想される。

劣 → 優

$$\mathrm{ASP} \rightarrow \begin{matrix}\mathrm{CORAL}\\ \mathrm{APL}\end{matrix} \rightarrow \mathrm{SKETCHPAD} \rightarrow \mathrm{LEAP}$$

③ 必要スペース量

$$\mathrm{SKETCHPAD} \rightarrow \mathrm{CORAL}\cdot\underset{(20)}{\mathrm{APL}} \rightarrow \underset{(42)}{\mathrm{ASP}} \rightarrow \mathrm{LEAP}\ \underset{*(96)}{(64)}$$

（注） （ ）内の数値は、必要スペース量の比率を示す。

＊はAリング、Oリング、Vリングがある時の値を示す。

第4節　言　語　各　論

この節では、これまでに開発されてきた汎用グラフィック言語について、その詳細を伝える。一般に、これまでのグラフィック言語は、ストラクチャー・セットをその中に含めていないものが多い。従って、データ・ストラクチャ・パッケージについては別に後半に、まとめて紹介しておいた。

4－1　GSP（Graphic Subroutine Package）

概　要

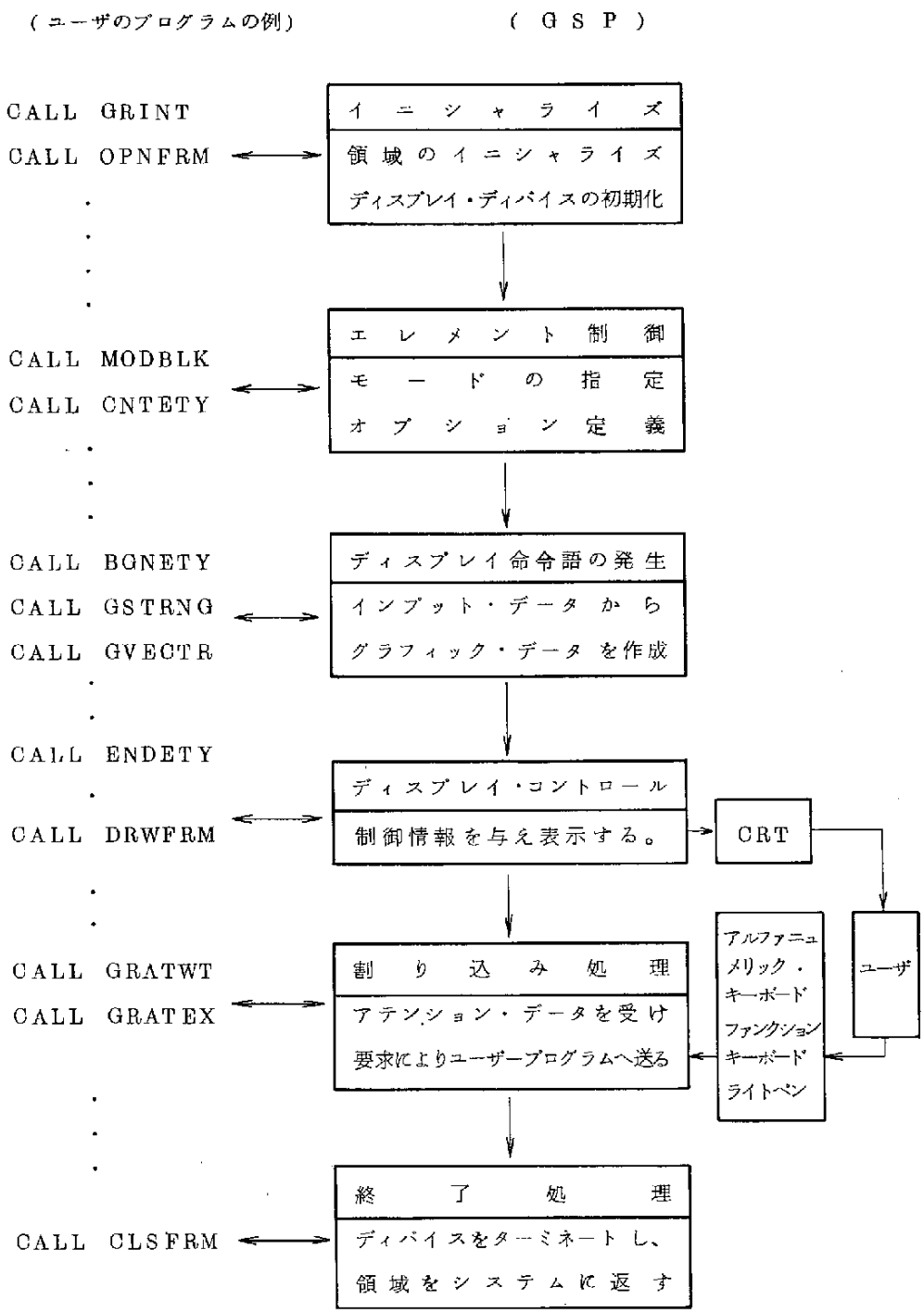
ＧＳＰは既存のコンパイラ言語からサブルーチン・コールの形で手軽に利用できること、又グラフィック言語としての基本的な機能をほぼ満してくれるということから、多くのグラフィック・システムにおいて最も実用に供されてきた。

しかし、図形データの扱いという点に関しては、まだまだ低レベル過ぎる感があり、特に図形の構造化、図形操作に関する機能的不足は大きい。又アーギュメント・リストによる表現は煩雑であり、この面でのユーザの負担も大きい。

ＧＳＰサブルーチンの機能

ＧＳＰでは、複雑な表示データを編集するためのモジュールや割り込み処理に関する手続きを代行するモジュール等があらかじめサブルーティン・パッケージの形で準備されていて、ユーザはこれらのモジュールをＦＯＲＴＲＡＮ等の既存の言語から呼び出して利用する事ができるようになっている。

一般にＧＳＰのサブルーティンは次のように分類され、使用される。


（ユーザのプログラムの例）
（ＧＳＰ）
CALL GRINT
CALL OPNFRM
イニシャライズ
領域のイニシャライズ
ディスプレイ・ディバイスの初期化
CALL MODBLK
CALL CNTETY
エレメント制御
モードの指定
オプション定義
CALL BGNETY
CALL GSTRNG
CALL GVECTR
CALL ENDETY
ディスプレイ命令語の発生
インプット・データからグラフィック・データを作成
CALL DRWFRM
ディスプレイ・コントロール
制御情報を与え表示する。
CRT
CALL GRATWT
CALL GRATEX
割り込み処理
アテンション・データを受け要求によりユーザープログラムへ送る
アルファニュメリック・キーボード
ファンクションキーボード
ライトペン
ユーザ
CALL CLSFRM
終了処理
ディバイスをターミネートし、領域をシステムに返す

(1) イニシャライズ

使用するバッファのイニシャライズ及びディバイスの結合を行う。

(サブルーチン例)

o 指定するディスプレイ装置を使用可能にする。

o 使用領域のイニシャライズ

o そ の 他

(2) エレメント・コントロール

ＧＳＰでは、図形要素(点、線、文字など)を記述することにより、ディスプレイ・オーダが作成される。これらの図形要素は、図形処理を行うための識別対象となる論理的な図形単位にグルーピングされる。(ここではこの論理的な図形単位をエンティティと呼ぶ)エンティティは、制御情報が与えられる単位であると同時に、ライトペン・ピックをした時の識別単位でもある。

次に示す例では、BGNETY と ENDETYにより囲まれた図形要素がグルーピングされエンティティとして定義される。

〔例〕

エンティティ ID1

```
CALL BGNETY(ID1)
   CALL GPOSIT (Xo,Yo……)
   CALL GVECTR (X1,Y1……)
   CALL GSTRNG (X2,Y2,TITLE…)
    ⋮
CALL ENDETY
```

エンティティ ID2

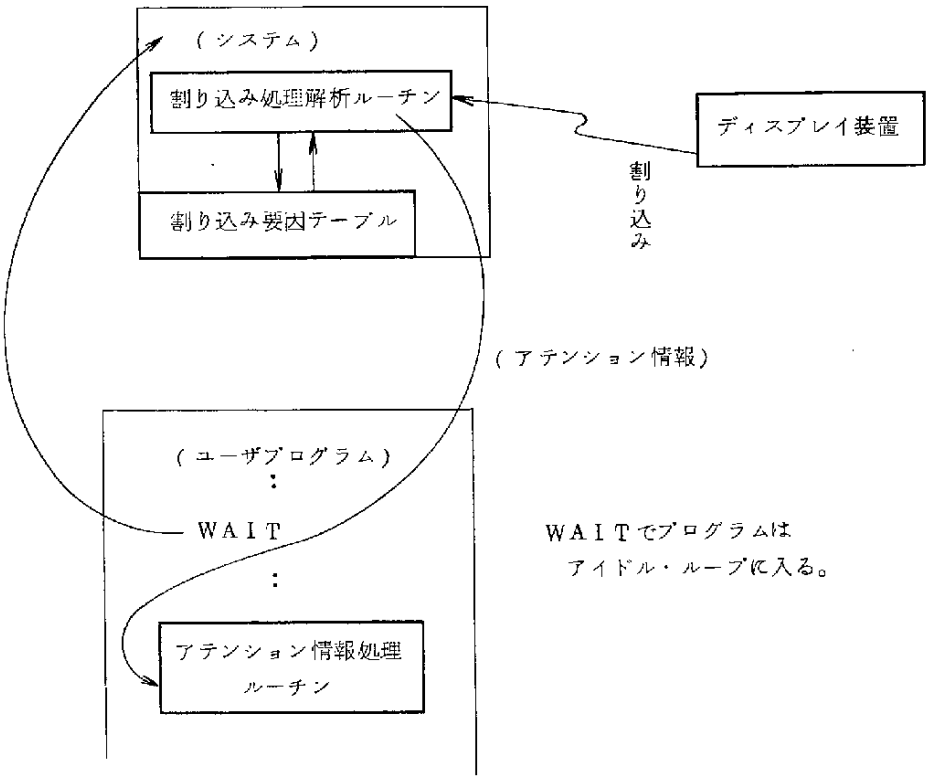
```
CALL BGNETY (ID2)
   CALL GSTRNG (X4,Y4, EXAMPLE……)
    ⋮
CALL ENDETY
    ⋮
```

(3) アテンション・ハンドリング

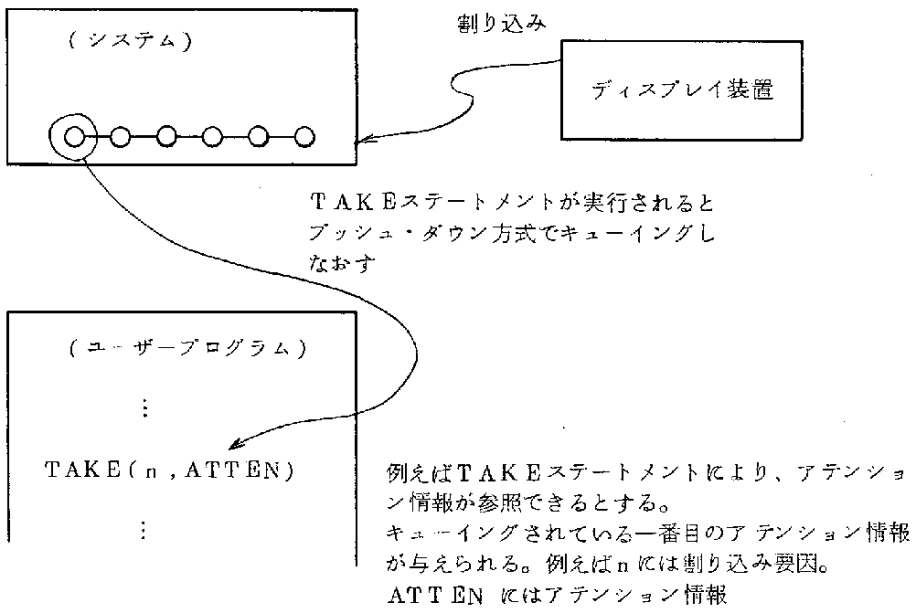
ＣＲＴに表示された図形をインタラクティブに処理するためのルーチンである。

オペレータはライトペンやファンクション・キーの操作により、実行中のプログラムに割り込みをかける。そのアテンションの扱いに対して、ＧＳＰでは次のような方法がよくとられている。

① ユーザが前もって、受け付け可能な割り込み要因（ｅｘ・ライトペン、ファンクションキー……）と、そのアテンション情報を処理するルーチンを対応づけておく。アテンションが起こると制御は割り込み処理解析ルーチンにわたされ、その割り込みが受け付け可能であるかをしらべる。もし受け付け可能であれば、対応のとられたルーチンへ制御がわたされる。もし受け付け不可能であれば、そのアテンションはキャンセルされ、プログラムは復帰される。なお、この場合の割り込みはプログラムが割り込み待ちの状態にある時のみ受け付けられる。



② アテンションが起こると、次々にキューイングされ、アテンション情報だけがスタックされる。そしてユーザ・プログラムの中でそのアテンション情報を参照するためのステートメントが用意されていて、キューイングされているアテンション情報を処理できる。



GSPではこれらの割り込み処理のどちらかが、使用されているケースが多い。しかし両者の処理を併用し、受け付け可能として登録されていないアテンションに対してはキューイングする形をとっているものもみられる。

（サブルーチン例）

①の場合

- プログラムがアテンションを受け付ける割り込み要因を指定する。
- 割り込み要因とそのアテンションのための処理ルーチンの対応をとる。
- プログラムがアイドル・ループに入るためのモジュール

②の場合

- キューイングされているアテンション情報を取りだす。

IBM 1130/2250 の GSPによるプログラム例を次に示す。

〔例〕

最初に一つの正方形と「SQUARE」という文字をディスプレイし、次にもう一つの正方形をだし、それを消すプログラム例を次に示す。

```
DIMENSION  ICA(50),GCA(21),X(4),Y(4)
DIMENSION  X2(4),Y2(4),TEXT(2)

DATA  X / 450.0, 650.0, 650.0, 450.0 /
DATA  Y / 650.0, 650.0, 450.0, 450.0 /
DATA  X2/ 300.0, 300.0, 200.0, 200.0 /
DATA  Y2/ 200.0, 300.0, 300.0, 200.0 /
DATA  TEXT/ SQUA , RE  /

CALL GSPIN (0,0,IRET,ICUM)
CALL ICAIN (1,ICA(1),ICA(50),0)
CALL GCAIN (GCA)
CALL BELMT (2,4)
CALL MVPOS (GCA,450.0,450.0,0)
CALL PLINE (GCA,X,Y,4)
CALL MVPOS (GCA,525.0,675.0,0)
CALL PTEXT (GCA,TEXT,6,1,1)
CALL EELMT (2)
CALL EXEC  (1,2,0)
PAUSE 1111
CALL XELMT (2,5)
CALL BELMT (3,1)
CALL MVPOS (GCA,200.0,200.0,0)
CALL PLINE (GCA,X2,Y2,4)
CALL EELMT (2)
PAUSE 2222
```

```
CALL DELME  (3)
PAUSE 3333
CALL GSPTM
STOP
END
```

4－2　GRAF

概　要

GRAFは、FORTRANをベース言語として作成されたグラフィック言語で、特にグラフィック・データの扱いに関する機能に配慮が為されている。又表現形式も、ステートメント形式をとりGSPのCALL形式と較べればはるかに記述性のよいものになっている。機能の大きさからは、GSPとハイレベル・グラフィック言語の中間に位置するものと考えればよいだろう。

GRAFを最も特徴づける点は、ディスプレー・ヴァリアブルと呼ばれるグラフィック・データの導入である。

ディスプレー・ヴァリアブルは、図形要素を表現するデータ・タイプで、従来の数式表現に於ける変数的な扱いを受ける。又定数に相当するものとしては、ディスプレー・ファンクションと呼ばれるグラフィック・ファンクションが設けられている。即ちディスプレー・ヴァリアブルで表現される図形データの具体的な内容は、ディスプレー・ファンクションで定義されることになる。

この他に、図形データを操作するオペレータとして＋，＝が許められ、グラフィック・データ表現は、これらの演算子とグラフィック・ヴァリアブル、グラフィック・ファンクションを用いたディスプレー・エクスプレッションとして表現される。割り込み処理に関しては、それほど機能的拡充は見られず、アテンションキューを調べ、割り込みが発生していれば、その内容をそっくりユーザに知らせるというごく単純な方法が採用されている。

開　発

IBM　ロスアンジェルス科学センター　A. HURWITZ, J.P. CITRON

開発された時期は1966～1967年頃とみられる。

使用システム

エクステンディドOS／360の下で動作する。CRT装置はIBM2250モデルIを使用している。

言語タイプ

OS／360　FORTRAN Ⅳをベース言語とし、プリプロセス方式をとっている。

発表論文

1967 SJCC GRAPHICAL ADDITIONS TO FORTRAN

機　　能

(1)　数式処理機能

ＦＯＲＴＲＡＮ に同じ

(2)　データ・ストラクチャー・ハンドリング

特に機能的拡張は無い。ＦＯＲＴＲＡＮ に同じ

(3)　図形記述、表示機能

o　ディスプレー・ヴァリアナル

拡張されたデータ・タイプとしてディスプレー・ヴァリアブルがある。ディスプレー・ヴァリアブルは、図形のまとまりを表現する基本的なグラフィック・データである。
又表示の単位でもある。図形表現は、後で述べるようにディスプレー・ヴァリアブル、ディスプレー・ファンクション、図形演算子（＋）のエクスプレッションとして与えられる。
ディスプレー・ヴァリアブルの構成規則はＦＯＲＴＲＡＮの変数構成規則に従うが、変数と区別する意味であらかじめ宣言しておかなくてはならない。又配列型のディスプレー・ヴァリアブルも認められている。

（例）　ＤＩＳＰＬＡＹ　ＲＩＶＥＲ，ＶＡＬＬＥＹ

ＲＩＶＥＲ，ＶＡＬＬＥＹはそれぞれディスプレー・ヴァリアブルである。

o　グラフィック・オペレータ

演算子，＋，＝を拡張し次の意味に使用する。

＋　：ディスプレー・ヴァリアブル又はディスプレー・ファンクション の間で定義される時、図形の併合（ ｉｎｃｌｕｓｉｏｎ ）を意味する。

＝　：右辺のディスプレー・エクスプレッションを左辺のディスプレー・ヴァリアブルで表現する。

（例）〔図２－１〕に於てＭＡＮ，ＤＯＧをそれぞれディスプレー・ヴァリアブルとすると、〔図２－１〕全体はディスプレー・ヴァリアブルＳＣＥＮＥ１として次の様に表現される。

ＳＣＥＮＥ １

ＭＡＮ　ＤＯＧ

ＳＣＥＮＥ１ ＝ ＭＡＮ＋ＤＯＧ

〔図２－１〕

o　ディスプレー・ファンクション

ディスプレー・ヴァリアブルのアクチュアルな図形データ（座標値、テキスト等）を定義するためにいくつかのディスプレー・ファンクションが用意されている。

POINT（x，y）　：　（x，y）の位置に点をプロットする。

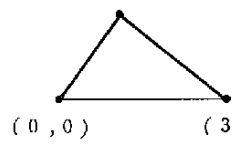
LINE　（x，y）　：　（x，y）まで直線を引く

PLACE（x，y）　：　書き出し点を（x，y）に置く

CHAR（STRING，length，mode）　：　キャラクター・ストリングを書く

PRINT（n，変数リスト）：　nはフォーマット番号で、フォーマットに従って変数の表示データは定義される。

（例）



〔図2－2〕

〔図2－2〕は三角形と、TRIANGLEという文字列を表わしている。FIGUREをディスプレー・ヴァリアブルとして〔図2－2〕を表現すると次の様になる。

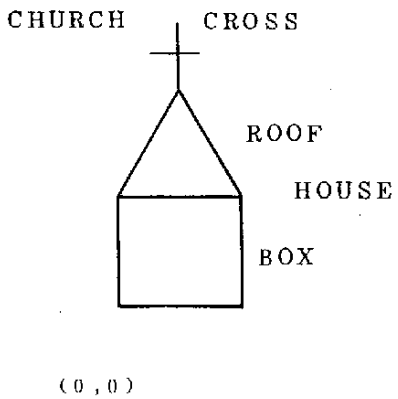
FIGURE＝PLACE(0，0)＋LINE(1，2)＋LINE(3，0)
＋LINE(0，0)＋PLACE(3，2)＋CHAR
(TRIANGLE，8，0)

o　ディスプレー・エクスプレッション

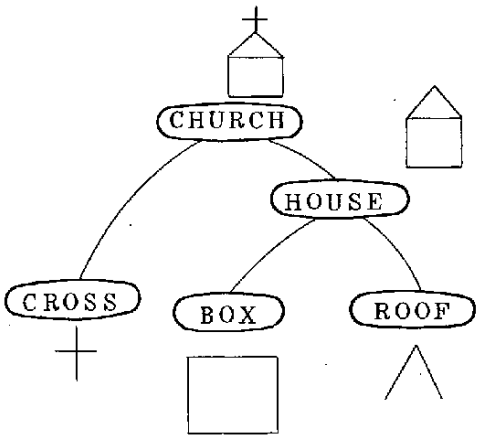
図形表現は、ディスプレー・ヴァリアブル、ディスプレー・ファンクション、及び図形オペレータ＋、＝を用いたディスプレー・エクスプレッションとして表現される。図形の階層化はディスプレー・エクスプレッションの中で為される分である。

（例）〔図2－3〕は、ディスプレー・ヴァリアブル、ROOF，BOX，CROSS，HOUSE，CHURCHから構成される図形を表わしている。又〔図2－4〕はこれらの図形要素間の階層関係を説明したものである。

これは、次の様なディスプレー・エクスプレッションとして表現される。



〔図2－3〕



〔図2－4〕

```
DISPLAY  CROSS,BOX,ROOF,HOUSE,CHURCH  (DVの宣言)
CROSS=  PLACE(0,5)+LINE(2,5)+PLACE(1,6)+LINE(1,4)
ROOF   =  PLACE(0,2)+LINE(1,4)+LINE(2,2)
BOX    =  PLACE(0,0)+LINE(0,2)+LINE(2,2)+LINE(2,0)
          +LINE(0,0)
HOUSE  =  BOX   ROOF
CHURCH=  CROSS+HOUSE
```

o　ユーザの定義するディスプレー・ファンクション

ディスプレー・ファンクションはユーザが定義する事も出来る。

宣言の方法は、FORTRANの外部関数定義の形式によく似ている。以下に例を示す。

(例)　BOXは長方形を表わす図形ファンクションとして次のように定義される。

```
DISPLAY  FUNCTION  BOX(X,Y,U,V)
BOX=  PLACE(X,Y)+LINE(U,Y)+LINE(U,V)
        +LINE(X,V)+LINE(X,Y)
RETURN
END
```

o　グラフィック入出力機能

定義されたディスプレー・ヴァリアブルを表示したり、又表示中のインスタンス(後述する)を消去したりするためのグラフィック入出力ステートメントがいくつか用意されている。表示は先

とっている。

GRAFもこの形式をそのまま受けついでいる。

即ち、ユーザは、あらかじめテーブル（大きさ5の配列）を準備しておき、テーブル・アドレスを知らせて、DETAIN又はDETECT等のファンクションを実行すると、その時のアテンション・キューの内容が、テーブルに通知される様になっている。又アテンションが有ったか無かったか、又あったとすればどういう種類のものかはファンクションの値として知らされる。

この様にユーザはプログラム中でいずれかのファンクションを実行することにより、アテンションの有無とその種類を知ることが出来る。又、さらに詳しい情報はテーブルの中味を参照することにより得る事が出来る。

DETECT（ATQ）　　ATQはテーブル・アドレス

（例）　DIMENSION　ATQ（5）

J＝DETECT（ATQ）

この結果J＝0ならばそれまでに割り込みが発生していなかったことを示す。又J≠0ならば割り込みが発生していた事を示す。この時Jの値とATQの内容は次の様な意味を持つ。

ファンクションの値	意　味
T＝0	アテンションの発生が無かった
T＝−1	ライトペン・ディテクトが発生した
	|ATQ(1)|＝ライトペン・ディテクト されたインスタンスの番号
	ATQ(2)＝ライトペン・ディテクト された時のx座標
	ATQ(3)＝ライトペン・ディテクト された時のy座標
T＝1	ファンクションキーによる割り込みが発生した
	ATQ(4)＝キー番号
	ATQ(5)＝オーバーレイコード
T＝2，3，4	それぞれ（ENDKEY，CANCELKEY，END OF ORDERED SEQUENCE KEY）システムリザーブドのファンクションキー割り込みの発生を意味する。
	ATQには何も通知されない。

LPNAME　ファンクション

インスタンスが、インターナル・ナンバーで管理されていることは前にも述べた。LPNAME

はインスタンスのインターナル・ナンバから、それに対応するディスプレーヴァリアブルを知るためのファンクションである。

（例）　次の例は、ディスプレー・ヴァリアブルS，T，U，Wを表示し、そのどれかがライトペンで指示された時に、S，T，U，Wのどれが指されたかを知るまでの過程を示している。

```
DIMENSION  INQ(5)  ……テーブルの確保
DISPLAY  S,T,U,W  ……ディスプレー変数の定義
IB=PLOT(S,T,U,W)……ディスプレー変数の表示
J=DETAIN(INQ)  ……アテンションキューの吟味
```

でもしJ＝－1ならばLP割り込みが発生したことを示すから、この後で

K＝LPNAME(INQ(1),S,T,U,W)とすると K＝1～4に対してそれぞれインスタンス S，T，U，Wに対応するからKの値からディテクトされたディスプレー・ヴァリアブルを知る事が出来る。

4－3　GRINⅡ

概　要

GRINⅡは、Bell telephone で計画された大規模なオンライン・グラフィカル・サービス・システムの一環として開発されたもので、直接にはGRAPHICⅡ（インタライティヴ・グラフィック・システム）の下で利用されることを目的としている。

言語自体は低レベルのものであるが、図形データの扱いに関しては次の様な特徴を持っている。

o　ディヴァイス・インディペンデントな標準形式として扱われる。

o　図形データ構造の中に問題用データ構造も表現出来る。

始めに、GRPHICⅡを含めたトータル・システムとしてのオンライン・グラフィック・システムの概要を紹介しておこう。これについては１９６７．FJCCの発表論文の中で次の様にその構想が述べられている。

「１９６４年に完成したGRAPHICⅠの経験を基に、高速なハード・コピーから高度なインタラクティヴ・グラフィック・ジョブ・プロセシングにわたる広範囲な遠隔グラフィカル・サービスをはかるシステムを開発中である。全体は次の５つのシステムから構成され、GE－６４５をセントラル・プロセッサーとしてMULTICSの下に動作する。

① STARE（Short Turn Around Record）

端末からの指示で高速なハード・コピーを行う。

（Xerox LDX printer 使用）

② GLANCE（Graphical Accessory on line Console）

端末からコントロールされるプログラムの出力をCRTに表示する。

（CRTにインタラクションはとれない）

③ GRAPHICⅡ

端末に於けるインタラクティヴ・グラフィカル・ジョブを可能にする。

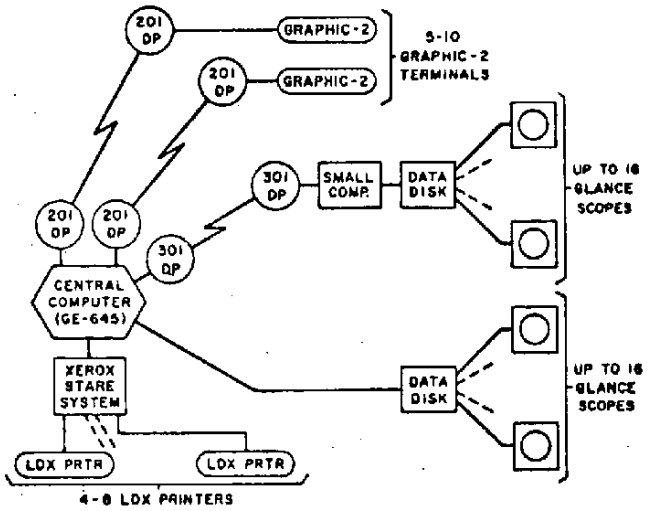
④ マイクロ・フィルム・ハード・コピー

⑤ ドラフティング

①，⑤は標準の商用システムを使用している。

〔図３－１〕は全体の構成を図示したものである。

特徴は、これ等の５つのシステムのデータ通信を管理するGRAFCOMというシステムがあり、そこで扱われる図形データの形式は各ディヴァイスに依らない標準形式を備えてい



〔図3－1〕

ることである。これはプログラム間での図形情報の交換が容易なことを、又中央のファイル・システムに於ては標準図形データファイルとして一括管理出来ること等の長所がある。又標準形式を備えたデータを各ディヴァスに応じた出力フォーマットに変換するため、各ディヴァイス別のトランスレータ・モジュールが存在しこれがその役割を果す。

開　発

Bell telephone Laboratory　CARL CHRISTENSEN, E.N.PINSON

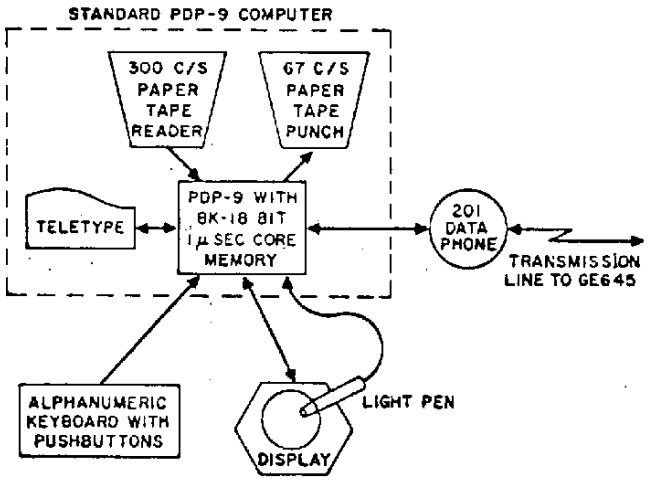
１９６７年にその構想を発表しているが、正確なところは分らない、１９６８より後か？

使用システム

ＧＥ－６４５，ＰＤＰ９から構成されるマルチプロセッサー・システム。

ＭＵＬＴＩＣＳの下で動作する。グラフィック・オペレーティング・システムはGRAPHIC Ⅱと呼ばれる。

〔図3－2〕にそのハードウェア構成を示す。



〔図3－2〕

言語タイプ

マクロ・ステートメントの形式をとる。GRINⅡコンパイラーにより翻訳される。

発表論文

1967 FJCC MULTI-function graphics for a large Computer System

機　　能

(1) 非グラフィカルな機能

はっきりしないが、四則演算程度のものは備えているかもしれない。

(2) データ・ストラクチャ・ハンドリング

図形構造を表現する基本要素として、リーフ・ブロック，ノード・ブロック，ブランチ・ブロックがあるが、この他に非グラフィカル・データを蓄えるデータ・ブロックがある。データ・ブロックは、ブランチ・ブロックに結合出来る様になっているから、図形構造の中に問題用データを含ませる事が出来る。

ストラクチャー操作に関しては、図形構造の操作機能をそのまま利用する形になる。

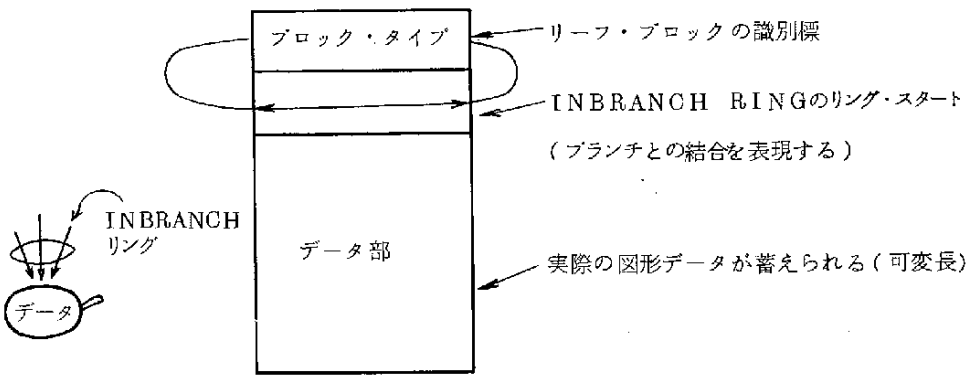
(3) 図形記述、表示機能

図形を構造化して扱うが、図形表現は図形データ構造の表現という形式で与える。

o グラフィック・データ

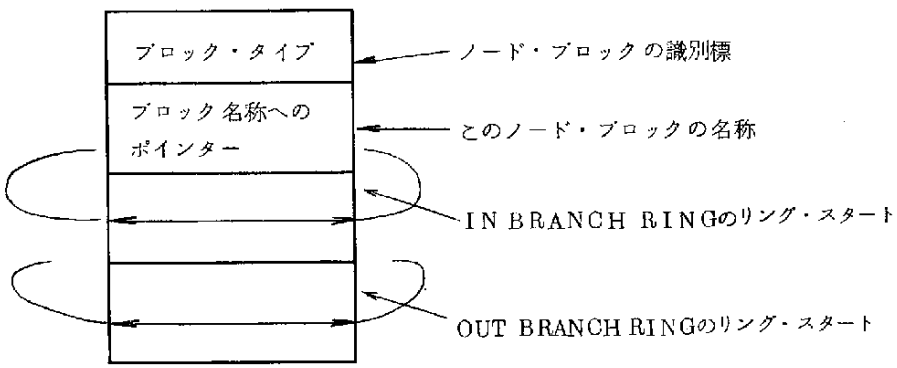
図形データを表現するための3つの基本的なブロックがある。

① リーフ・ブロック　：　実際の図形データを蓄えるブロックで、構造化の対象となる最も下位の要素（アトム）である。（〔図3－3〕参照）

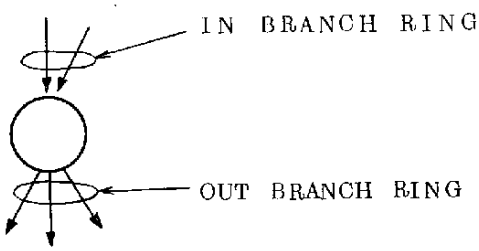


〔図3－3〕

② ノード・ブロック　：　構造化された図形を表現する。

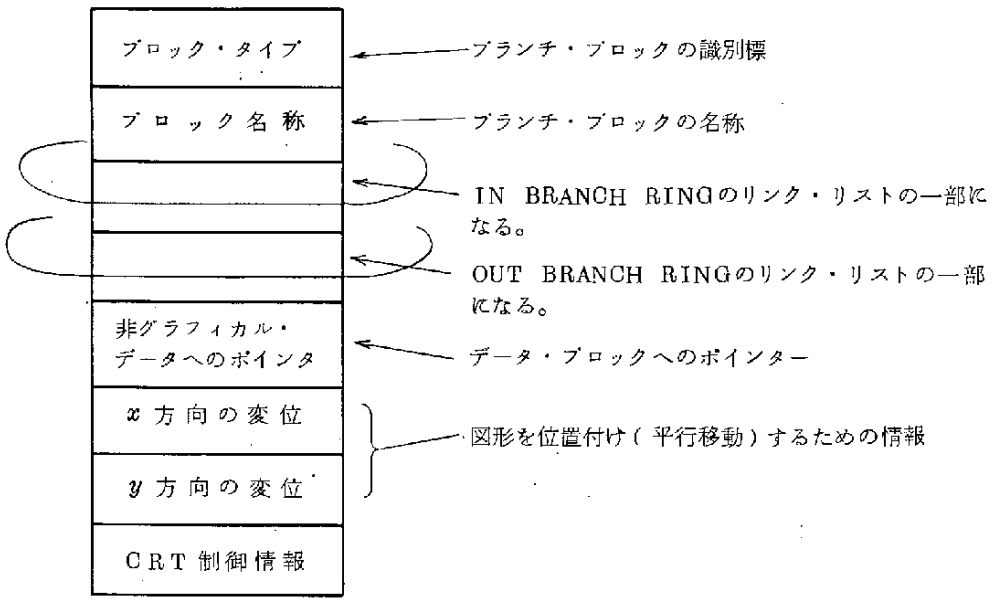


〔図3－4〕

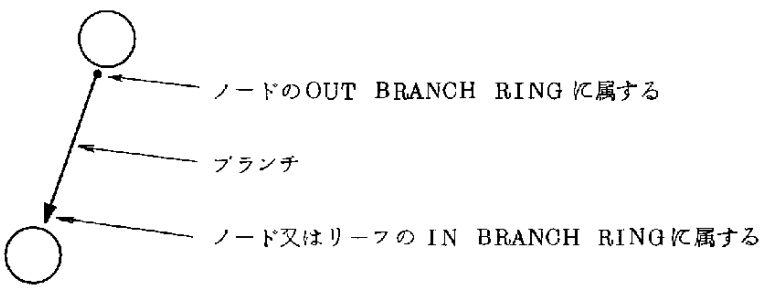


③ ブランチ・ブロック

位置情報、表示される時のＣＲＴ制御情報等を蓄え、リーフ、ノード間の関連を表現する。又データ・ブロックへのポインタも蓄えている。

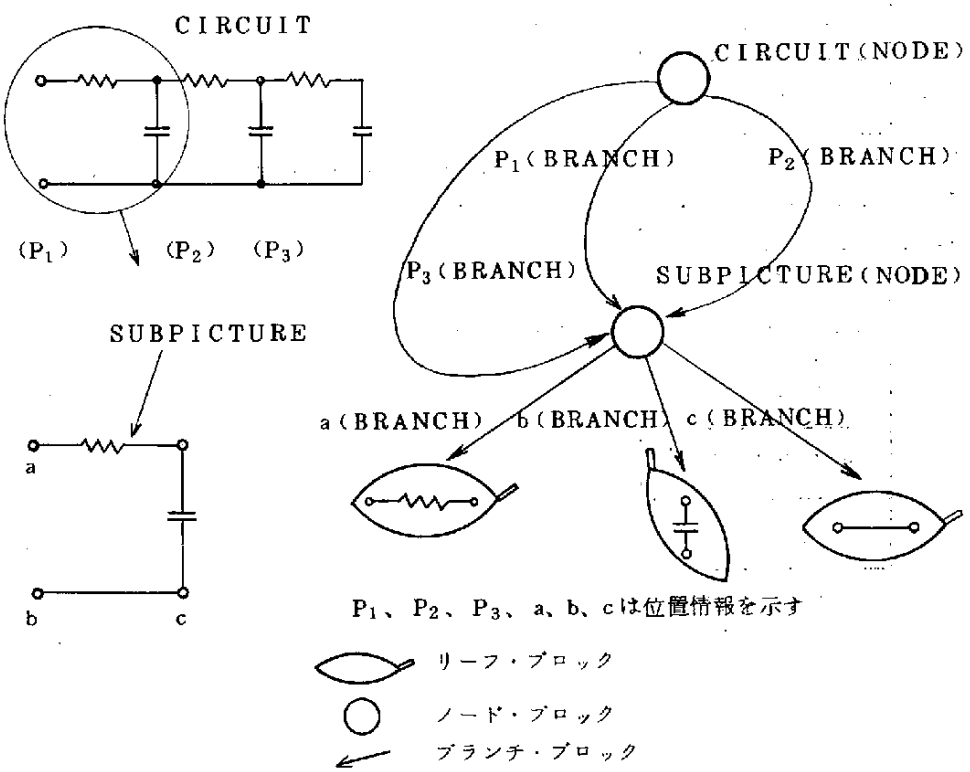


〔図3－5〕



〔図3－6〕は回路図CIRCUIT を GRINⅡ のデータ構造で表現したものである。

〔図3－6〕 データ構造概念図



o 図形データの定義

図形データは、リーフ・ブロックに図形情報を格納するという形式で定義される。

リーフ・ブロックに格納する図形情報を定義するために次の2つのステートメントが用意されている。

① TEXT(A,B,C……)

文字列ABCを定義する。

② VECTOR(X1,Y1,X2,Y2……Xn,Yn)

(X1,Y1),(X2,Y2)…(Xn,Yn)の点列からなる曲線(折線)を定義する。

これ等の情報をリーフ・ブロックに格納するには次のLEAFステートメントが使用される。

LEAF POINTER

新たにリーフ・ブロックを創成し、そのアドレスをＰＯＩＮＴＥＲに知らせる。

一つのＬＥＡＦステートメントから次のＬＥＡＦステートメントの間で定義された図形情報が、最初のリーフ・ブロックの内容となる。

（例）　ＬＥＡＦ　Ｐ１

ＴＥＸＴ　（Ａ，Ｐ，Ｑ）

ＬＥＡＦ　Ｐ２

ＶＥＣＴＯＲ（１，１，２，２）

ＬＥＡＦ　Ｐ３

⋮

Ｐ１のアドレスで示されるリーフ・ブロックの内容は文字列ＡＰＱである。又Ｐ２で示されるリーフ・ブロックの内容は（１，１）、（２，２）を結ぶ線分である。

o　図形の構造化ステートメント

ノード、ブランチ、リーフを構造化するステートメントには次の様なものがある。

① ＮＯＤＥ　ＢＬＫＰＴＲ（（Ｂ，ＤＸ，ＤＹ，Ｐ，Ｔ，ＬＢ，Ｄ），……）

ＢＬＫＰＴＲ　：　ノード・ブロックを創成しＢＬＫＰＴＲにそのアドレスを知らせる。

Ｂ，Ｔ　　：　Ｂはこのノードから出るブランチを定義し、Ｔはそのブランチの行き先のノード又はリーフを指定する。

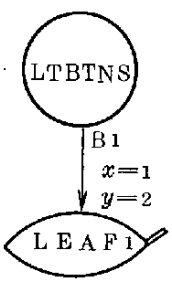
ＤＸ，ＤＹ　：　ブランチに格納される位置情報

ＬＢ　　：　このブランチが表現する図形がライトペンで指摘された時、ＬＢで示されたアドレスに制御がわたる。

Ｄ　　：　このブランチに付加するＤＡＴＡブロックへのポインターを示す。

（例）

ＮＯＤＥ　ＬＴＢＴＮＳ（（Ｂ１，１，２，，ＬＥＡＦ１，ＲＯＵＴＩＮＥ））とすると図の様な構造が作り出される。又ＬＴＢＴＮＳが表示された時、ライトペン・ディテクトを受けると、プログラム上ＲＯＵＴＩＮＥと云うラベルが付けられたステートメントに制御が渡る。



〔図３－７〕

② BRANCH （BLKPTR，FROM，DX，DY，P，TO，LB，DATA）

BLKPTR ： 新たにブランチ・ブロックを創成しそのアドレスをBLKPTRに知らせる。

FROM ： ブランチの始点となるノードを指定する。

TO ： ブランチの終点となるノード又はリーフを指定する。

DX，DY，LB，DATAについては①と同様

③ DETACH（BRANCH1，BRANCH2…）

BRANCH1，BRANCH2……をストラクチャーから取り除く

④ ATTACH BRANCH1，NODE1

NODE1で示されたノードにBRANCH1で示されたブランチを付加する。

o 図形出力

表示はノードを対象として行われる。

ディスプレー・ノードと呼ばれる特定のノードに、対象とするノードをブランチで結合することにより表示される。これには次の様なステートメントが用意されている。

① NEWESP（BRANCH1，BRANCH2…）

BRANCH1，BRANCH2をディスプレー・ノードに結合する。この時それまでにディスプレー・ノードに付加されていたブランチは取り除かれる。

② ADDDSP（BRANCH1，BRANCH2……）

①と同じだが、それまでにディスプレー・ノードに結合されていたブランチはそのままの状態に置かれる。

③ SUBDSP（BRANCH1，BRANCH2……）

BRANCH1、BRANCH2をディスプレーノードから取り除く。

(4) 割り込み処理機能

インタラクションの過程は Question & Answer という形式で記述する。（典型的な同期処理）

Questionを与えるステートメントにWHICH、WHEREがあり、これが実行されると割り込み待ちの状態に入る。

WHICH ： ライトペン、ライトボタンの割り込みを受けつける。

WHERE ： トラック・マーク割り込みを受けつける。

（例） WHERE COORD，LABEL

プログラムはこの文を実行すると割り込み待ちの状態に入る。この状態でトラックマークを適当に設定しキーを押すとＣＯＯＲＤにトラックマークの位置情報が格納されＬＡＢＥＬで示されるステートメントに制御が移行する。

4－4　DIAL

概　　要

ＤＩＡＬはユタ大学で開発されたもので、グラフィックシステムに於けるいくつかの試案をテストする目的で開発された実験用の言語であるといわれている。

言語作成にあたって、既存の言語をベースとする方法をとらなかったこともあって、非グラフィカルな処理に関する機能には多少制約がある。しかし作成者等は大抵のグラフィック・プログラムを開発する上ではこの程度の言語で十分だと考えているようである。

ＤＩＡＬの大きな特徴は、イメージ表現に於けるディスプレー・プロセデュアの採用と割り込み処理に於けるステート・ダイヤグラムのアプローチであろう。

後述するＭＥＴＡＶＩＳＵのグラフィック・プロセデュアもＤＩＡＬのそれとほぼ同じ扱いをしている。又ＡＩＤＳに於ける割り込み処理の方法も基本的にはＤＩＡＬと同じ扱いと云えるだろう。機能的には未だ完全に満たされたとはいい難いが、グラフィック・データの扱い、割り込み処理に関する扱いの一つの基本的な手法を提案したという意味で興味深いものである。

開　　発

ＡＲＰＡの委託による。

Information Research Laboratory, William M. Newman University of Utah

開発された時期は１９６８～１９６９頃とみられる。

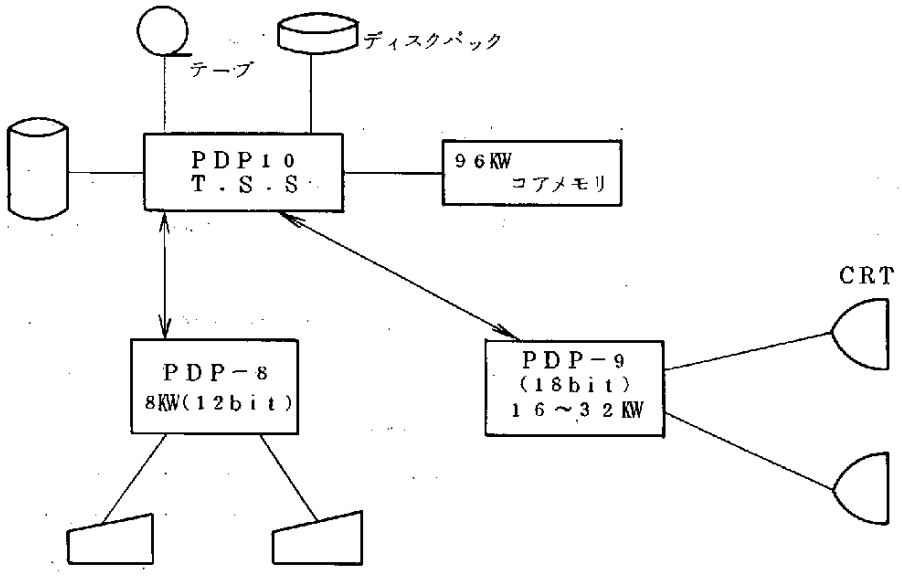
使用システム

ＰＤＰ１０，ＰＤＰ９による２プロセッサ方式を採用している。

又、ＣＲＴはＵＮＩＶＡＣ１５５９を使用している。

（ライトペン、アルファニューメリック・キーボード、ファンクション・キー附属）ＯＳについては述べられていないがＰＤＰ１０はＴ．Ｓ．Ｓ．で動作する。

〔図４－１〕はハードウェア構成を示したものである。



テレタイプ×40　　〔図4－1〕

言語タイプ

ＡＬＧＯＬタイプの言語で、ＥＵＬＥＲに近い機能を備えている。ＤＩＡＬプロセッサにより翻訳される。

＊ＥＵＬＥＲ … ＥＵＬＥＲはグラフィックスのデータ・ストラクチャーを研究する目的で作成された言語で、アルゴルのサブセットとしての機能の他、リスト処理機能、マトリックス演算機能を備えている。ＤＩＡＬにはこれ等の機能は無い。

発表論文

ユタ大学研究レポート　1969.

"An Experimental Display programming Language for the PDP10 Computer"

機　能

(1) 非グラフィカルな機能

ＡＬＧＯＬに近い機能を持つが、フローティング・データを扱う事は出来ない。又 GOTO 文がない。

(2) データ・ストラクチャー・ハンドリング

ＤＩＡＬ自体は配列処理の機能しか持たない。他の言語とリンケージを取ることは可能だ

から、ＥＵＬＥＲ等の機能を利用する方法を取っていると思われる。

(3) 図形記述、表示機能

図形の定義は、図形出力ステートメントを用いてディスプレー・プロセデュアを定義する事によって行われる。ディスプレー・プロセデュアの内部で他のディスプレー・プロセデュアを引用する事も可能であるから図形の階層関係の表現はディスプレー・プロセデュアのネストとして表現してゆく事が出来る。

表示は、フレームを対象として行われる。従ってあるディスプレー・プロセデュアを表示しようとするなら、あらかじめこのディスプレー・プロセデュアをフレームとして定義しておかなくてはならない（フレーム・プロセデュアと呼ばれる）。

表示命令は、このフレーム名を指定して出される。ディスプレー・ファイルはフレーム単位で管理されているから、画面に対する図形の修正、削除等はフレーム単位で行う事になる。

o 図形出力ステートメント（表示データを作成するという意味でディヴァイスへの出力という意味ではない）

図形出力ステートメントはディスプレー・プロセデュアの内部でのみ使用可能であり、次の５つのステートメントからなる。

① ＬＩＮＥ　x，y

現位置から、x，yだけ変位した点まで直線を引く。

② ＬＩＮＥ　ＴＯ　x，y

現位置から、現座標系の（x，y）の点まで直線を引く。

③ ＭＯＶＥ　x，y

現位置からx，yだけ変位した点にビーム位置を移す。

④ ＭＯＶＥ　ＴＯ　x，y

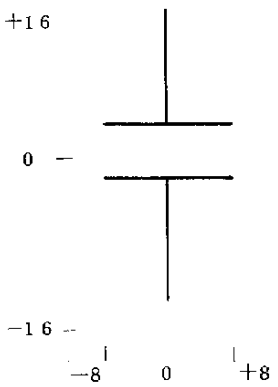
現座標系の（x，y）の点にビーム位置を移す。

⑤ ＤＩＳＰＬＡＹ＂ＳＴＲＩＮＧ，ＡＴ x，y＂

（x，y）の位置にＳＴＲＩＮＧを表示する。

次の例は〔図４－２〕をディスプレー・プロセデュア　ＣＡＰＡＣとして定義したものである。

```
CAPAC← MOVE TO 0,−16; LINE TO 0,−2;
       MOVE −8,0; LINE 16,0,;
       MOVE −16,4; LINE 16,0,;
       MOVE −8,0; LINE TO 0,16;
```



〔図4－2〕

又次の例は〔図4－3〕をディスプレー・プロセデュアABとして定義したものである。

. ANSWER=

(100,200)

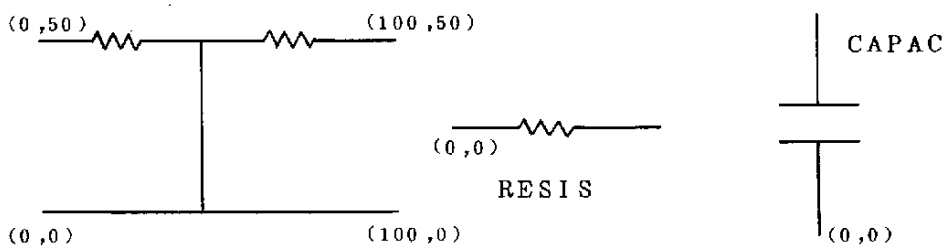
```
AB← DISPLAY"ANSWER=",N AT100,200;
```

〔図4－3〕

o ディスプレー・プロセデュア

ディスプレー・プロセデュアは、先に述べた基本図形出力ステートメントで定義される以外に、他のディスプレー・プロセデュアを引用して定義することも出来る。

例えば〔図4－4〕に於て、RESIS，CAPACがすでに定義済みのディスプレー・プロセデュアであれば、回路全体はディスプレー・プロセデュアCCTとして次のように表現される。



〔図4－4〕

```
CCT←CAPAC AT50,0; RESIS AT 0,50; RESIS AT50,50;
      MOVE TO 0,0; LINE TO 100,0
```

プロセデュア間の座標関係は、呼び出された側のプロセデュアの原点が、呼び出された位置に一致する様に管理されている。

上の例では、ＣＡＰＡＣの原点はＣＣＴ座標係の(３２,２４)に移される。

又ディスプレー・プロセデュアを引用する時、ＳＩＺＥオプションを指定することが出来る。例えばＣＡＰＡＣ　ＡＴ３２,２４　ＳＩＺＥ＝ｍ として引用するとＣＡＰＡＣはｍ／１０２４のスケールファクターでスケーリングされて引用された事になる。

ディスプレー・プロセデュアによる図形表現にはいくつかの問題がある。その一つは図形構造がプログラム構造として表現されるため、動的な図形構造の変更が困難となることである。ＤＩＡＬではディスプレー・プロセデュアにアーギュメントを持たせること、又プロセデュア内部に通常のプログラム命令を書く事を認める事によって上記の問題に対処している。

例えば次の例は、ＩＦ文とアーギュメントを使用して実行時の状態によってＲＥＳＩＳ又はＣＡＰＡＣを表示するディスプレー・プロセデュア　ＰＲＯＣＸ を定義したものである。

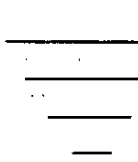
(例)

```
PROC X ← NEW FLAG; IF FLAG=0 THEN RESIS AT 0,0;
            ELSE CAPAC AT 0,0;
```

(注)　ＮＥＷ　ＦＬＡＧ はＦＬＡＧが仮引数であることを示す。

実引数はＰＲＯＣＸ〔ａ〕ＡＴ２，４の形で引き渡す。

又次の例では、ディスプレー・プロセデュアにくり返し文を用いることにより、規則性のある図形を端的に表現している。



```
GROUND← MOVE4,2;
        FOR K←Ø STEP1 UNTIL4 DO
        BEDIN LINE2*K,Ø;
        MOVE −(2*K+1),1 END;
        MOVE TO4,6; LINE Ø,2;
```

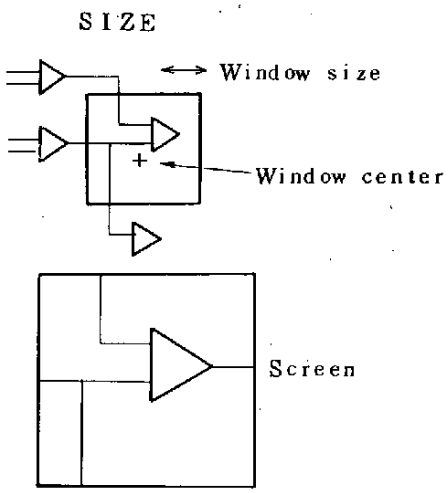
〔図４－５〕

o　フレーム・プロセデュア

表示は、フレームを対象にして行われるから、表示しようとするディスプレー・プロセデュアはあらかじめフレームとして定義しておかなくてはならない。この過程はフレーム・プロセデュアと呼ばれ次の形式をとる。

F1←WINDOW　CCT　AT　250,300　SIZE200;

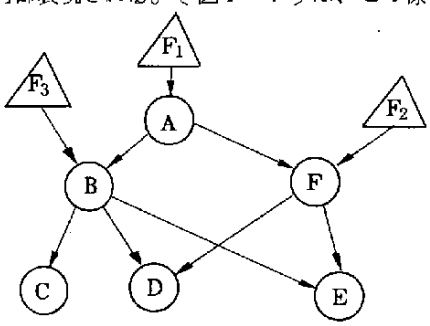
CCTは表示しようとするディスプレー・プロセデュア名又F1はフレーム名称である。WINDOWは、CCTの特定の範囲だけをとり出す操作を指定するもので、ATでWINDOWの中心、SIZEで範囲を指定する。(〔図4－6〕参照)



〔図4－6〕ウインドイング

o　グラフィック入出力

ディスプレー・プロセデュア及びフレーム・プロセデュアは一つのプログラム構造として内部表現される。〔図4－7〕は、この様子を示したものである。



〔図4－7〕

F1～F3はフレーム名で識別されるエントリーポイントを示し、A～Fはそれぞれディスプレー・プロセデュアである。(表示出来るディスプレー・プロセデュアは、A，B，F)表示データは、このプログラムを実際に走らせることにより生成される。出力命令FRAMEは、このプロセデュアを

呼び出し実行させる働きをする。（ＣＡＬＬ文にあたるものである）。

例えば、フレームＦ３を指定してＦＲＡＭＥ　Ｆ３；とすると、Ｆ３に制御がわたりＢ、Ｃ、Ｄ、Ｅが順次実行され、その過程で表示データの作成が行われる。そして最終的にフレームＦ３の表示データが完成すると、これをディスプレー・バッファーに転送し画面に表示する。

又システムはフレーム・リストを持ち、ディスプレー・ファイルの内容をフレーム単位で管理している。従って画面上の図形の修正、削除はフレーム単位で行う事が可能である。

特にフレームの削除を指定する出力命令にＤＥＬＥＴＥがある。次に示す例は、ＦＲＡＭＥ、ＤＥＬＥＴＥ命令がディスプレー・ファイルにどの様に作用するかを示したものである。

（例）

フレーム操作ステートメント

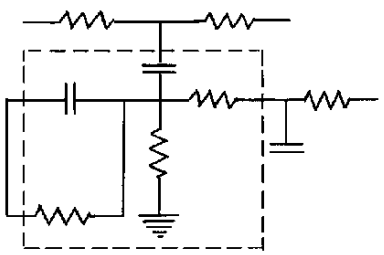
Statement	Effect	Display File Contents
FRAME F1	Generate F1	F1
FRAME F2	Generate F2	F1,F2
FRAME F1	Regenerate F1	F1,F2
FRAME F3	Generate F3	F1,F2,F3
DELETE F2	Remove F2	F1,F3
DELETE F2	No effect	F1,F3
DELETE F1	Remove F1	F3
FRAME F3	Regenerate F3	F3
DELETE F3	Remove F3	Empty

最後に〔図4－8〕をディスプレー・プロセデュアとして定義しＣＲＴに表示するまでの過程を示すＤＩＡＬのプログラム例を紹介しておこう。

```
TITLE EXAMPLE
BEGIN
NEW VERT,HORIZ,FR1,K;
DISPLAY PROCEDURE CCT,32:RESIS,CAPAC,8:GROUND;

1:CAPAC← "NEW HV; IF HV=HORIZ THEN
```

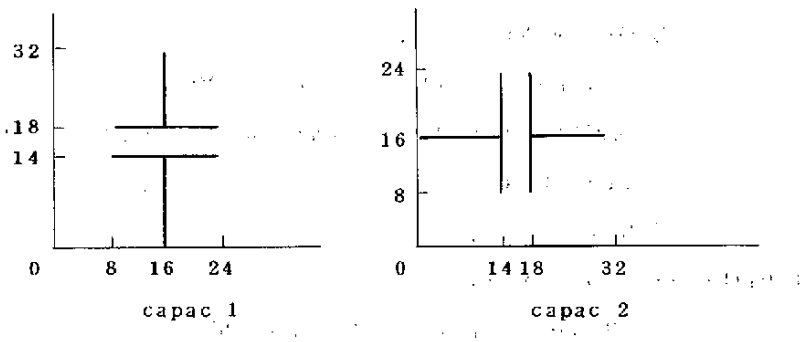
```
        BEGIN MOVE 16,Ø; LINE Ø,14; MOVE -8,Ø;LINE 16,Ø;
          MOVE -16,4; LINE 16,Ø; MOVE -8,Ø; LINE Ø,14
        END ELSE
        BEGIN MOVE Ø,16;LINE 14,Ø; MOVE Ø,-8;LINE Ø,16
         MOVE 4 -16,LINE Ø,16,MOVE Ø,-8; LINE 14,Ø
       END
2:RESIS←NEW HV; IF HV=HORIZ THEN
       BEGIN MOVE Ø,1,LINE 4,Ø,
         FOR K←1 STEP 1 UNTIL 3 DO
         BEGIN LINE 2,4, LINE 4,-8, LINE 2,4 END;
         LINE 4,Ø
       END ELSE
       BEGIN MOVE 16,Ø; LINE Ø,4;
         FOR K←1 STEP 1 UNTIL 3 DO
         BEGIN LINE 4,2; LINE -8,4, LINE 4,2 END;
         LINE Ø,4
       END ;
3:GROUND←"MOVE 4,2;
        FOR K←Ø STEP 1 UNTIL 4 DO
        BEGIN LINE 2*K,Ø, MOVE -(2*K+1),1 END;
        MOVE TO 4,6; LINE Ø,2";
4:CCT←"RESIS[HORIZ] AT Ø,Ø;
      MOVE TO Ø,16; LINE Ø,32; MOVE TO 32,16; LINE Ø,32;
       LINE 12,Ø;
      CAPAC[HORIZ] AT Ø,32,
      CAPAC[VERT] AT 28,48;
      RESIS[VERT] AT 28,16;
      GROUND AT 36,Ø SIZ,16,
      RESIS[HORIS] AT 44,32;
      RESIS[HORIZ] AT 76,32;
```



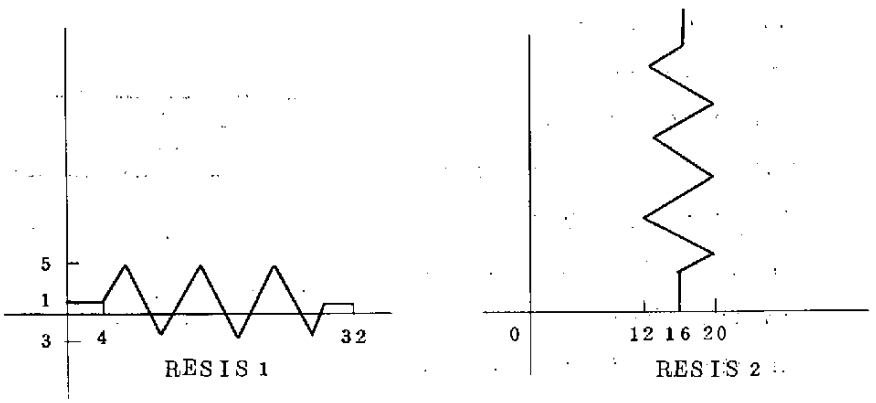
〔図4－8〕

```
        CAPAC〔VERT〕 AT 6∅,16;
        RESIS〔HORIZ〕 AT 12,64;
        RESIS〔HORIZ〕 AT 44,64";
5:FR1   "WINDOW CCT AT 36,36 SISE 32";
6:HORIZ←∅; VERT←1;
7:FRAME FR1
  END
```

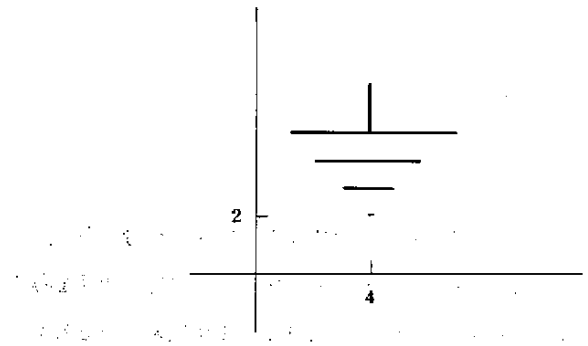
1：ＣＡＰＡＣは仮引数ＨＶをもつディスプレイ・プロセデュアで定義される。もし、ＨＶがＨＯＲＩＺであるならば、ｃａｐａｃ １で定義され、他の場合はｃａｐａｃ ２で定義される。



2：ＲＥＳＩＳは仮引数ＨＶをもつディスプレイ・プロセデュアで定義される。もしＨＶがＨＯＲＩＺならばｒｅｓｉｓ １で定義され、他の場合はｒｅｓｉｓ ２で定義される。



3.：ＧＲＯＵＮＤはディスプレイ・プロセデュアで次の図のように定義される。



4：ＣＣＴは回路図のディスプレイ・プロセデュアである。

既に定義されたディスプレイ・プロセデュア ＣＡＰＡＣ，ＲＥＳＩＳ，ＧＲＯＵＮＤのポジションを指定することにより、回路図を定義する。

5：ディスプレイ・プロセデュアＣＣＴに指定された領域のウインドイング処理をして、フレームＦＲ１を定義する。

6：ＨＯＲＩＺに０，ＶＥＲＴに１をアサインする。

7：フレームＦＲ１が指定され、ウインドイングされたＣＣＴの表示データが作成され、ディスプレイ画面にこれを表示する。

(4) アテンション・ハンドリング

このシステムの作成者であるW. M. NEWMANは、インタラクティヴ・プロセスの表現に有限オートマトンの概念をとり入れ、その構想を具体化して ＮＥＴＷＯＲＫ ＤＥＦＩＮＩＴＩＯＮ ＬＡＮＧＵＡＧＥを発表した。（A system for interactive graphical programming by William M. Newman SJCC 1968）

ＤＩＡＬに於ける割り込み処理の扱いもその延長とみる事が出来るだろう。

ＤＩＡＬでは、その時に許されるアクション（割り込み要因）とリアクション（それに対する動作、即ち割り込み処理）との関係で定義されるプログラムの状態を考え、これをプログラム・ステートと呼ぶ。割り込み処理過程はプログラム・ステートの定義を与える手段と、プログラム・ステートを遷移させる手段を考えれば割り込み処理過程の一つの表現形式が考えられる。ＤＩＡＬに於ては、この様な形式で割り込み処理過程が記述される。

o プログラム・ステートの定義

プログラム・ステートはＤＵＲＩＮＧ文とそれに続くＯＮステートメント群によって定義さ

れる。

```
DURINGステート番号　DO
        BEGIN
        ON　ステートメント
        ……
        END;
```

ステート番号は、プログラム・ステートを識別するためにつけられた番号である。

ONステートメントは次の形式で割り込み要因とその処理プロセスの対応関係を与える。

（処理プロセスとはDIALステートメントで記される割り込み処理過程である）

ON　要因　DO　プロセス　END

例えば、次の例でステート12に於て割り込みが発生すると

```
DURING　12　DO
        BEGIN　ON　CHAR　"A" DO　プロセス1　END
                ON　CHAR　"P" DO　プロセス2　END
        END;
```

DURING以下の一回のON文が順次実行され、要因が一致すると右辺で定義された処理プロセスが実行される事になる。

o　コンディション名の種類

ON文で指定できる条件名（割り込み要因を示す）には次のものがある。

① キャラクター割り込み　（CHAR）

（例）

ON　CHAR　DO	テレタイプからキーインされる任意のキャラクターによる割り込みを示す。
ON　CHAR"Q"DO	テレタイプからキーインされるキャラクタQによる割り込みを示す。

② メッセージ割り込み　（SYM）

（例）

ON　SYM　DO	:	CR又はTABで終る任意のメッセージ割り込みを示す。

ON SYM" XYZ" DO： CR又はTABで終るストリングXYZの割り込みを示す。

③ マウス・スイッチ割り込み （SW）

ON SW DO ： マウス上の任意のスイッチによる割り込みを示す。

④ ライトペン・ディテクト割り込み （HIT）

ON HIT27 DO ： CRT上に表示されたディスプレー・プロセデュア27がライトペン・ディテクトされた時。

（注） ディスプレー・プロセデュアは、ASオプションを用いて識別名（表示された図形を論理的に識別する名前）をつける事が出来る。

（例） RESIS AT2,4 AS17 ;

o プログラム・ステートの遷変

プログラムをあるステートに移動させる命令にENTER文がある。例えばENTER5とすると、これはプログラム・ステートを現在の状態から、DURING5 で定義されたステートに移行することを、指示したことになる。

o 割り込み情報を調べるための予約語

DIALでは、割り込み情報を格納する予約語を次の様に決めている。

INCHAR ： キャラクター割り込みが発生した時、その文字コードを格納する。

INSYM ： メッセージ割り込みが発生した時、その文字列を格納する。

INVAL ： 数値割り込みが発生した時、その値を格納する。

INX, INY, INSW ： マウスSWによる割り込みが発生した時、それぞれマウスのx, y座標、スイッチ番号を格納する。

HITN ： ライトペン・ディテクトが発生した時、識別名を格納する。

以下にDIALに於ける割り込み処理の例をあげる。

```
TITLE RECTS
BEGIN
NEW NR,K,TEMP,NEWX,NEWY,SIZX,SIZY,CREATE;
NEW KS[Z[50,2],RX[S0],RY[SU],FR1,FR2;
DISPLAY PROCEDURE NEWR,RECT,LAYOUT;

% RECT IS CALLED BY LAYOUT TO DRAW ALL STORED RECTANGLES %
```

```
RECT←"NEW N,LINE  RSIZ[N,10,Ø;LINE Ø, RSIZ[N,2];
            LINE -RSIZ[N,10,Ø;LINE Ø,-RSIZ[N,2]";

% LAYOUT CREATES DISPLAY OF ALL STORED RECTANGLES,
  OMITTING TEMPORARILY DELETED RECTANGLE TEMP %

LAYOUT-"FOR K-1 STEP 1 UNTIL NR DO
        IF K TEMP THEN RECT[K] AT RX[K],RY[K] AS K";

% NEWR DRAWS NEWLY DEFINED RECTANGLE %

NEWR←"MOVE TO NEWX,NEWY,
      LINE SIZX,Ø; LINE Ø,SIZY;LINE-SIZX,Ø;LINE Ø,
      -SIZY";

% FRAME FA1 CREATES ALL STORED RECTANGLES,
  FR2 CREATES NEW RECTANGLE %

FR1←"WINDOW LAYOUT AT 512,512 SIZE S12 AS 51";
FR2←"WINDOW NEWR AT 512,512 SIZE 512";

% THIS CODE EXECUTED AT START %

CREATE←TEMP←NR←Ø;
ENTER 2;

% BASE MODE STATE %

DURING 1 DO
BEGIN ON CHAR"G" DO ENTER 2;
      ON CHAR"M" DO ENTER 5;
      ON CHAR"S" DO ENTER 6;
      ON CHAR"D" DO ENTER 7
END;
```

```
DURING 2 DO
  ON SW DO BEGIN NEWX←INX;       % FIRST CORNER %
                 NEWY←INY;
                 ENTER 3
           END;

DURING 3 DO
  ON SW DO BEGIN SIZX←INX-NEWX;       % SECOND CORNER %
                 SIZY←INY-NEWY;
                 FRAME FR2;            % DISPLAY NEW
                                         RECTANGLE   %
                CREATE←1;          % 1E NEW RECT CREATED %
                ENTER 4

DURING 4 DO
  ON SW DO BEGIN NEWX←INX;    % ON EACH NEW POSN,
                                  REPOSITION  %
                 NEWY←INY;
                 FRAME FR2    % AND DISPLAY IN NEW
                                  POSITION %
           END;

DURING 5 DO
  ON HIT S1 DO BEGIN TEMP←HITN; % TEMP=RECTANGLE TO
                                  BE MOVED  %
                     SIZX←RSIZ[HITN,1];
                     SIZY←RSIZ[HITN,2];
                     FRAME FR1; % DISPLAY WITHOUT
                     THIS RECTANGLE %
                     CREATE←1;
                     ENTER 4
           END;
```

```
DURING 6 DO
  ON HIT 51 DO BEGIN SIZX←RSIZ[HITN,1]; % DUPLICATE
                                          THIS ONE %
                     SIZY←RSIZ[HITN,2];
                     CREATE←1;
                     ENTER 4
                END;

DURING 7 DO
  ON HIT 51 DO BEGIN TEMP←HITN;        % DELETE THIS ONE %
                     FRAME FR1;
                     ENTER 8
                END;

DURING 8 DO
  ON CHAR"N" DO BEGIN TEMP←Ø; % N MEANS RESTORE IT %
                      FRAME FR1;
                      ENTER 7
                END;

DURING ALL DO
  BEGIN ON CHAR "P"DO % P MEANS MAKE CHANGE PERMANENT %
    BEGIN IF TEMP#Ø THEN
      BEGIN NR←NR-1;        % TEMP=RECTANGLE TO BE
                                          DELETED   %
        FOR K←TEMP STEP 1 UNTIL NR DO
        BEGIN RSIZ[K,1]←RSIZ[K+1,1]; % FILL GAP LEFT
                                       BY DELETING
              RSIZ[K,2]←RSIZ[K+1,2];
              RX[K]←RX[K+1];
              RY[K]←RY[K+1]
        END;
```

```
        TEMP←ø
      END;
      IF CREATE=1 THEN
      BEGIN CREATE←ø;  % IF NEW RECTANGLE CREATED,ADD IT %
        NR←NR+1;       % AND INCREMENT NUMBER %
       RRSIZ[NR,1]←SIZX;
        RSIZ[NR,2]←SIZY;
        RX[NR]←NEWX;
        RY[NR]←NEWY
      END;
      DELETE FR2;      % REMOVE TEMPORARY DISPLAY %
      FRAME FR1;
      ENTER 1
    END;
    ON CHAR "X" DO     % ON X, RESTORE TO ORIGINAL STATE %
    BEGIN DELETE FR2;
          FRAME FR1;
          CREATE←TEMP←ø;
          ENTER 1
    END
  END
END
```

4－5　AIDS

概　要

AIDSはFORTRANをベースとして作られたグラフィック言語である。図形データの扱いに関しては、GRINⅡとほとんど同じ形式をとっている。又割り込み処理の扱いも基本的にはDIALと同じで、プログラムステートの遷移という形でこれを表現してゆく。原理的なことは、それぞれGRINⅡ、DIALで述べられているので、ここでは言語体裁を簡単に紹介するのにとどめる。

開　発

NATIONAL SECURITY AGENCY T.R.STACK S.T.WALKER 開発された時期は1970頃とみられる。

使用システム

OSについては述べられていない。

ハードウェア構成は次の通りである。

処理装置：SEL 840MP（64KW, 24bITS／W）

CRT：SEL 816A

ライトペン，ファンクションキー，ASCⅡキー・ボード付

言語タイプ

FORTRANをベース言語とするプリ・プロセス方式。

発表論文

1971 SJCC

AIDS－Advanced Interactive Display System

機　能

(1) 数式処理機能

FORTRANに同じ

(2) データ・ストラクチャー・ハンドリング

図形データ構造の中に、非グラフィカルなデータも格納出来るようになっている（レーベル・ブロックと呼ばれる）。構造操作は後述するインスタンスにレーベル・ブロックを付加したり、取りはずしたりする操作と、レーベル・ブロックからデータをリトリーブする操作が出来る様になっている（この辺の扱いはGRINⅡと同じである）。レーベル・ブロックの構造化は、インスタンスの構造化に全く従う形になる。

(3) 図形記述・表示機能

図形を構造化して扱うが、表現形式は図形表現と云うよりは、むしろ構造記述という感じが強い。

o グラフィック・データ

図形データとして次の４つのタイプがある。これもむしろ、データ・ストラクチャーを構成する要素と考えた方がよいかもしれない。

① イメージ

構造化の対象となる最もプリミティヴな図形要素（アトム）である。実際の図形データを蓄えている。

② セット

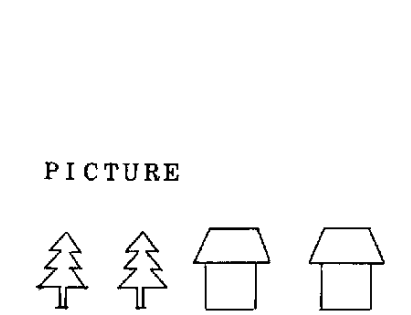
イメージ、セット、インスタンスにより構成され、構造化された図形を表わす。

③ インスタンス

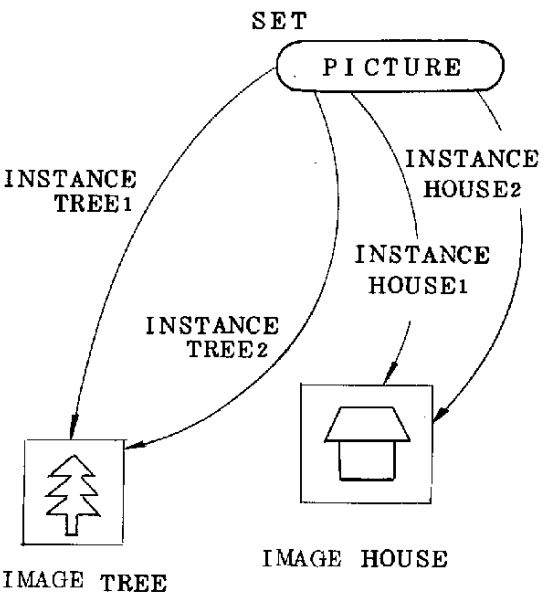
位置情報を蓄え、イメージ、セットの間の関連を表現する。

〔図５－２〕は〔図５－１〕の図形を、イメージ、セット、インスタンスで表現した一例である。矢印のついた線はインスタンスを表わしている。

（例） IMAGE、SET、INSTANCEの概念図



〔図５－１〕



〔図５－２〕

o　図形データの定義

前述した様に実際の図形データはイメージに蓄えられる。イメージに格納される図形データを定義するための次の２つのステートメントがある。

① A LINE FROM x,y TO x′,y′

(x,y) (x′,y′)の間の線分を定義する。

② TEXT "文字ストリング",x,y

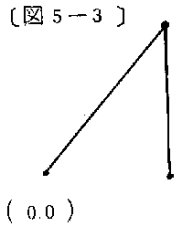
(x,y)の位置に文字ストリングで指定された文字列を書く。

③ INSERT

イメージに図形データを格納する。

(例)〔図５－３〕を表わすデータをイメージに格納する。イメージはMOUNTAINと名付けられている。

〔図５－３〕




```
IMAGE MOUNTAIN ……(宣　言)
INSERT IN TO MOUNTAIN:LINE FROM
0,0 TO 4,3, LINE FROM 4,3 TO 4,0
```

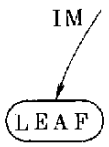
o　図形の構造化ステートメント

イメージ、セット、インスタンスを構造化するためのステートメントには次のものがある。

① DEFINENS

インスタンスをあるイメージ、又はセットに付加する。

(例)　インスタンスIMをセットLEAFに付加する例



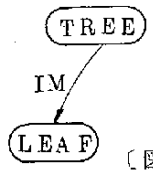
〔図５－４〕

```
IM DEFINES LEAF
```

② CONTAINS

インスタンスをセットに結びつける。

（例） 前の例でさらにＩＭをセットＴＲＥＥに結びつける。



IM DEFINES LEAF;
TREE CONTAINS IM;

〔図５－５〕

③ ＰＯＳＩＴＩＯＮ（インスタンス名称） ＡＴ ｘ，ｙ

インスタンスに位置情報を与える。

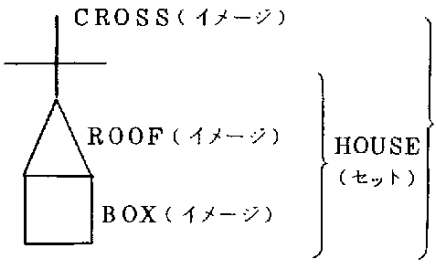
（例） ③の例でＰＯＳＩＴＩＯＮ ＩＮ ＡＴ ２，４とするとＬＥＡＦはＴＲＥＥの座標系で（２，４）の位置を原点として書かれることになる。

④ その他インスタンスを取り除く命令 ＤＥＴＡＣＨ，不要なエレメントをフリー・ストレージに返却するＤＥＳＴＲＯＹ等の命令もある。

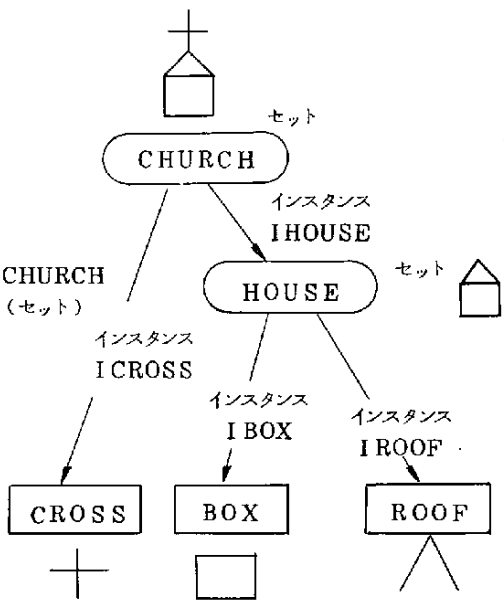
o グラフィック出力

セットを指定してＳＨＯＷステートメントを実行することにより表示される。

以下に、ＡＩＤＳを使って実際に図形表現をする過程を示す。〔図５－６〕は対象とする図形構成、〔図５－７〕はその階層関係をセット、イメージ、インスタンスで表わしたものである。



〔図５－６〕



〔図５－７〕

プログラム例

```
SET CHURCH,HOUSE                                          セットの宣言
INSTANCE ICROSS,IBOX,IROOF,IHOUSE                         インスタンスの宣言
IMAGE CROSS,BOX,ROOF                                      イメージの宣言
INSERT  INTO CROSS: LINE FROM 0,1 TO 2,1                  イメージCROUS
                  , LINE FROM 1,2 TO 1,0                  のデータを定義する
INSERT  INTO BOX  : LINE FROM 0,0 TO 0,2                  イメージBOXの
                  , LINE FROM 0,2 TO 2,2                  データ定義
                  , LINE FROM 2,2 TO 2,0
                  , LINE FROM 2,0 TO 0,0
INSERT INTO ROOF  : LINE FROM 0,0 TO 1,2                  イメージROOF
                  , LINE FROM 1,2 TO 2,0                  のデータ定義
POSITION ICROSS AT 0,4                                    インスタンス
POSITION IBOX   AT 0,0                                    ICROSSに位置
POSITION IROOF  AT 0,2                                    情報を入れる。
POSITION IHOUSE AT 0,0
CHURCH CONTAINS ICROSS AND IHOUSE
                                  CHURCHとICROSS,IHOUSEの結合
HOUSE CONTAINS IBOX AND IROOF
                                  HOUSEとIBOX,IROOFの結合
ICROSS DEFINES CROSS              ICROSSとCROSSの結合
IHOUSE DEFINES HOUSE
IBOX   DEFINES BOX
IROOF  DEFINES ROOF
SHOW   CHURCH
```

＊図形データはディスプレー・オーダの形で格納されている。従ってイメージ構造はそのままディスプレー・ファイルとして扱われ実行される。例えば上の例でDETACH IHOUSEとすると、図形構造HOUSEからCHURCHが取り除かれ、画面からもHOUSEが消える事になる。

(4) アテンション・ハンドリング

割り込み処理過程は、プログラム・ステートの遷移と云う形で記述される。基本的には、ＤＩＡＬに於る扱いと同じである。詳細についてはＤＩＡＬを参照されたい。

ここではステートメント機能についてのみ説明する。

o WHEN ステートメント

WHEN IN ステート番号、IF コンディション名、THEN処理プロセスの形式で、プログラム・ステートの定義を与える。

ステート番号は、プログラム・ステートの識別番号、コンディション名は割り込み要因の名称を書く。又処理プロセスには、ＡＩＤＳステートメント又はFORTRAN ステートメントによる割り込み処理過程が書かれる。

例えば、WHEN IN 5,IF LB A,THEN SHOW PICTUREとすると、ステート5に於てはライトボタンAの割り込みが許され、もしライトボタンAが押されたらPICTUREが表示される事を示す。

o WAIT ステートメント

プログラムをアイドル・ウエイトさせ割り込み待ち状態に入る。

o ENDWAIT

割り込み処理状態でこの文が実行されると、メインプログラムはＷＡＩＴを解除され、その下に続くステートメントに制御が移る。

o CHECK

ＷＡＩＴは、どこかでENDWAITが出されるまでプログラムを待ち状態に置くのに対し、CHECK はアテンションキューを調べもし割り込みが発生していたら直ちに処理をし、その後CHECK 以下に続くステートメントに制御を移す。

o ENABLE

ENABLE ステート番号の形式で、あるプログラムステートをエナーブルする、次にエナーブルが出されるまでこの状態は有効である。

4－6　GPL／I

概　要

GPL／Iは、PL／Iをベース言語として作成された汎用グラフィック言語である。その機能の大きさ、又整理された言語体系からして、現在までに開発されたグラフィック言語の中でも、最も完成された言語の一つと見ることが出来るだろう。

はじめに、GPL／Iを開発するにあたって作成者等が意図した設計方針について簡単に紹介しておこう。

o　設計方針

(1)　どんなグラフィック・アプリケーション・プログラムでも大低は非グラフィカルな演算機能を必要とする。従って、なるべくなら既存の言語をベースとして作成する事が望ましいだろう。又ベース言語によるアプローチには次の様な利点が考えられる。

o　グラフィックのステートメントを考える時の文法的な基盤が与えられる。

o　作成に要する時間が短縮される。

o　ベース言語を拡張、改良出来る。

o　ユーザがベース言語を知っていれば、言語を学習するための時間を節約出来る。

(2)　マシン・インディペンデント、ディヴァイス・インディペンデントであること。

即ち言語仕様が特定のマシン、ディヴァイスを前提に決定されてはならない。又ディヴァイス間の互換性を考えるには、デフォルトルールがうまく適用されるように考えるべきである。

(3)　イメージ構造はユーザから見えない存在でなくてはならない。

図形を内部的に構造化して扱う事は色々な面で必要であるが、例えばＡＩＤＳで見られたようにユーザがこの構造を意識しながら図形を記述するような形態は好ましくない。ユーザに対しては、図形記述に適した文法を与え、ユーザの記述を評価してこれを構造化して扱うのはシステムがやることである。

(4)　デバッグ機能について考慮すべきである。

グラフィック・プログラムのデバッグは、コンピュータとターミナルの両方を占有する場合が多く、又インタライティブ・プロセスの動作をチェックするためには多くの時間を必要とする。これは時間的にも経済的にも余り好ましい事ではない。ターミナルを使わずにプログラム・テストが出来るようにするためには、デバイスのソフトウェアー・シュミレータを備える事が必要である。

その他(1)に関連して、ＰＬ／Ｉをベース言語として選んだ理由として、作成者は次の様に述べている。

① 割り込み処理機構を備えている。

② データ・タイプが豊富であり、構造化機能を備えている。

③ 入出力機能が大きい。

④ 言語に拡張性がある。

開　発

ボーイング・コンピュータ・サービス・CO, Inc

by DAVIDN. SMITH, 開発された時期は１９６８～１９７１頃

使用システム

はっきりしないが、恐らくOS／360であろう。ディヴァイス・インディペンデントをねらいとしている。グラフィック入出力装置はライトペン、ファンクションキー、アルファニューメリック・キーボードを備えていることを前提としている。

言語タイプ

ＰＬ／Ｉをベース言語とするプリプロセス方式をとっている。

発表論文

1971 SJCC GPL/I-A PL/I extension for computer graphics

機　能

(1) 数式処理機能

ＰＬ／Ｉに同じ

(2) データ・ストラクチャー・ハンドリング

特に機能的拡張は無い。ＰＬ／Ｉに同じ

(3) 図形記述表示機能

グラフィック・データとして新たに２つのデータ・タイプ、イメージとベクトルが設けられている。

又これ等のグラフィック・データの間に適用される。いくつかのグラフィック・オペレータが用意されており、図形操作、図形の構造化はグラフィック・データ、グラフィック・オペレータによるイメージ・エクスプレッションと云う形で表現される。グラフィック入出力には、拡張されたGET，PUT ステートメントが使用される。

o ベクトル

ベクトルは最も基本的な図形データで、次の３つの目的に使用される。

① 位置を表わす量として

② スケールを表わす量として

③ 座標軸のまわりの回転を表わす量として

各々の具体的な例については後述するとして、ここではその基本的な性格について説明しよう。

a ベクトルの定義

ベクトルは、各座標軸成分を要素とするエクスプレッションとして表わされ、2次元又は3次元の座標量を表現する。

各成分の表現には次の表記法が使用される。

```
14.7X        ……… x成分が14.7
  2Y
10100BX
```

又各成分のエクスプレッションとして、ベクトルは次の様に表現される。

```
VEC1＝43.5X＋2.2Z＋1.4X  ;  (3D)
VEC2＝8X＋2Z  ;             (2D)
VEC3＝8X＋4Y  ;             (2D)
```

その他ベクトルに対して適用されるいくつかのベクトル・ファンクションがあり、ベクトルはこれらのファンクションによって定義する事も出来る。

b ベクトルの演算

ベクトル間に定義されるオペレータとして次の4つのオペレータが認められている。

＋	ベクトルの和を定義する。
－	ベクトルの差を定義する。
＊＊	ベクトルの内積を定義する
＊＊＊	ベクトルの外積を定義する

（例） VFC1とVEC2の内積、外積は次の様に表わされる。

```
VEC1 *.* VEC2
VEC1 *** VEC2
```

o イメージ

イメージは、図形のまとまりを表現する図形変数で、構造化、操作の対象となる最も重要な図形データの概念である。

イメージの幾何学的な性格は、後述するイメージ・エクスプレッションを通じて定義される。

又イメージはこの他に、宣言時に与えられるアトリビュートを保有し、他の図形的特質（表示された時の状態を表わすものが多い）は、このアトリビュートによって与えられる。

① イメージ・データ

イメージの実データを定義する図形量をイメージ・データと呼び、これにはPOINT（ベクトルで表わされる）、TEXT 及び、その他のイメージ・ファンクションがある。

（例） イメージ・データの例

POINTS	7.2X＋3.3Y－1.5Z
TEXT	"DEPRESS ANY KEY"
FUNCTIONS	ARC（POINT1，POINT2，3.1415） COPY（IM）

② グラフィック・オペレータ

イメージとイメージ又はベクトルの間で定義されるいくつかのグラフィック・オペレータがあり、図形操作、図形の構造化はこれらのオペレータを用いて為される。

＋＞ 併合を表わす（Inclusion）

（例） A，Bをイメージとすると

A＋＞Bは1つのイメージを表わし、それはA，Bを含む。

III 結合を表わす（Connection）

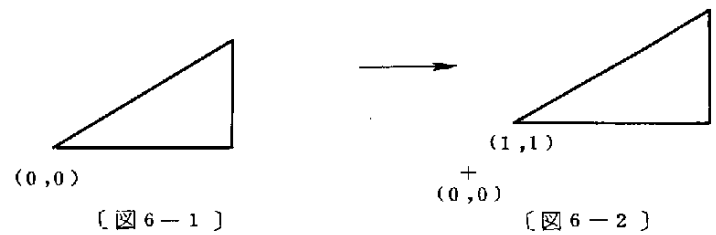
（例） P1，P2をPOINTS（ベクトル）とすると

P1IIIP2はP1、P2で結ばれるラインを表わす

@ 位置付けを表わす（Positionning）

（例） M1が〔図6－1〕の図形を表わす時、〔図6－2〕は

M1@（1X＋1Y）として表わされる



〔図6－1〕　　〔図6－2〕

＜＞ スケーリングを表わす

（例） 〔図6－1〕の例でM1を1/2倍した図形は次の様に表わされる。

M1<>(2X + 2Y)

各成分は、成分方向のスケール値を表わす。

＊＞ 座標軸のまわりの回転を表わす

（例） イメージAを2軸のまわりに180° 回転した図形を次の様に表現する。

A＊＞3.141592

又〔図6－3〕は、これ等のオペレータ間の優先関係を示したものである。

〔図6－3〕

OPERATOR	PRIORITY
***	Highest
.	
** ¬ prefix + prefix -	
* /	
infix + infix -	
<¬<> -> <= ¬= >= =	
&	
:	
<> *> @	
||| +>	Lowest

③ イメージ・エクスプレッション

イメージは、グラフィック・オペレータ、グラフィック・データ のエクスプレッションとして定義される。図形の階層関係もこの中で暗に定義されたことになる。

（例1）

SCENE1 MAN

DOG

P1

P2

〔図6－4〕

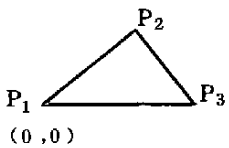
P1、P2をベクトル、MAN、DOG、SCENE1をイメージとすると、SENE1は次の様に定義される。

P1＝2X＋2Y ；

P2＝4X＋8Y ；

SCENE1＝MAN@P1 +> DOG @P2 ；

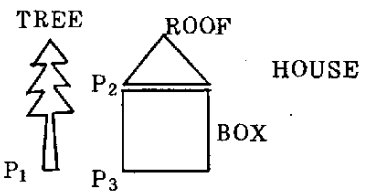
（例２）　次の例はイメージTRIANGLEを定義したものである。



〔図６－５〕

```
DCL   TRIANGLE IMAGE,
      (P1,P2,P3) VECTOR;
  P1=0X+0Y;
  P2=2X+2Y;
  P3=3X+0Y;
TRIANGLE= P1 ||| P2 ||| P3 ||| P1;
```

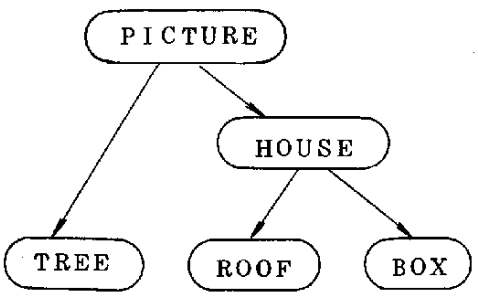
（例３）



〔図６－６〕

〔図６－６〕は木と家からなる図形を示している。又〔図６－７〕はその階層関係を示したものである。これをイメージ・エクスプレッションで表現してみると

（Ｐ１、～Ｐ３は位置を表わすベクトルである）



〔図６－７〕

```
HOUSE=ROOF @P2 +>BOX@P3
PICTURE=TREE@P1+>HOUSE
```

④　イメージ・ファンクション

イメージを扱ういくつかのファンクションが用意されている。その主な機能を挙げると次の様になる。

①イメージの定義、②イメージのグルーピング、③イメージの構造を調べる、④ライトペン・

ートの役割は、イメージが表示された時の表示条件を与えるものが多い。

又アトリビュートにはスタティック・アトリビュートとダイナミック・アトリビュートがあり、後者はアトリビュート・ファンクションを用いることにより実行時にその性格を変えることが出来る。

以下にイメージ・アトリビュートのいくつかの例を挙げる。詳しくはAppendix を参照のこと。

INTENSITY（ｎ）：ｎは０～９を記し、イメージが表示された時の輝度を指定する。

SCALE　（――）：イメージが表示される時にスケーリングされる。

COLOR　（―－）：イメージが表示される時の色を指定する。

FLICKER（－－－）：イメージが表示された時、点滅することを指示する。

＊ アトリビュートのダイナミックな変更について

アトリビュートは擬変数ＡＴＴＲを使用して動的に変更することが出来る。

例えば、イメージＡに関して、そのアトリビュートTHICKNESS、REGION を再定義するには次の様にする。

ATTR（Ａ）＝THICKNESS（２）＋REGION（X1、X2）；

右辺はアトリビュート・エクスプレッションと呼ばれるもので、複雑なアトリビュートに関する操作も、これを通じて容易に行なえるようになっている。

（例） REGION をアトリビュート・ファンクションとして次の様にすると。

ATTR（ALPHA）＝ATTR（BETA）－REGION（BETA）；

イメージBETAのアトリビュート・セットからREGION属性を取り除いたもの、ALPHAのアトリビュートにする。

o グラフィック入出力機能

図形入出力は、ＰＬ／Ｉの標準入出力機能を拡張した形で行なわれる。

① ファイルに関して

ＧＰＬ／Ｉは次の４つのファイル・タイプを持つ。

STREAM，RECORD，TRANSIENT，GRAPHIC

この内、GRAPHICが新たに追加されたもので、これには３つのタイプがある。各々は次の目的で使用される。

GRAPHIC
- DESIGN ：このファイルに対しては、表示、修正、インタラクションが可能。（ＣＲＴへの出力に相当する）
- DISPLAY：表示だけしか出来ない（プロッタへの出力に相当する）
- STORAGE：イメージの保存ファイルで表示はされない。後で検索、表示することが可能。

② 入出力ステートメントに関して

図形入出力は、イメージを対象にして行なわれる。

入出力命令にはＧＥＴ、ＰＵＴが使用され、通常のＰＬ／Ｉ入出力形式をとる。

（例1）イメージCAR，HOUSE，CLOUDSはファイルABCに出力される。

```
PUT FILE(ABC) LIST(CAR,HOUSE,
  (CLOUDS(I) DO I=1 TON));
```

（例2） IM1，IM2を表示する。

```
PUT EDIT(IM1,IM2)(2G(SCALE(V)));
```

Gはグラフィック・フォーマットを示し、ここではスケール・オプションが指示されている。

同様にＧＥＴステートメントはＳＴＯＲＡＧＥファイル又はＤＥＳＩＧＮ ファイルからイメージを取り出すのに使用される。又、ＧＰＬ／ＩではＧＥＴ、ＰＵＴによるイクスプリシットな入出力と別に、インプリシットな入出力がある。

これはＡＣＴＩＶＥ 属性を持ったイメージに適用される。

ＡＣＴＩＶＥ 属性を持ったイメージが表示されている時、プログラム上でこれに操作が加えられると、システムは自動的に表示されたものにも修正を加え、常に画面とイメージの関係を等しく保って呉れる。

その他、特殊な機能としてアニメーション機能がある。

これは出力命令ＡＮＩＭＡＴＥにより行なわれる。

（例）

```
ANIMATE FILE(X)LIST(CAR)(OFFSET(0.2X+1.7Y))
EVENT(EV);
```

上の例で、イメージＣＡＲは出力されると同時に、毎秒0.2 X＋1.7 Yインチの割合で動き出す。

（動作はイメージＣＡＲが消去されるか、ファイルが閉じられるか、事象変数ＥＶに完了が

ディテクトを発生したイメージを知らせる。等

ここでは、その中でも特に重要な、COPY、INSTANCE についてのみ説明する。詳しくはAPPENDIXを参照されたい。

（COPYファンクション）

1つのイメージAが、他の多くのイメージから参照されて使用されている時、Aに変化をもたらすと、Aに関連するすべてのイメージに変化をきたすことになる。

これが不都合である場合、Aをコピーし、コピーしたものを参照するメカニズムを組むことも可能である。COPYファンクションはこの様な目的に使用される。

例えば

```
A＝XX ;
B＝A  ;
A＝YY ;
```

と定義すると、最終的にAもBもYYに変る。

一方
```
A＝XX ;
B＝COPY（A）;
A＝YY ;
```

とするとAはYYに変わるが、BはAのもとの値XXを保っている。

（INSTANCEファンクション）

イメージ・エクスプレッションに於て、イメージ間の図形的階層関係は容易に定義されるが、図形の論理的なグルーピングに関する操作は含まれていない。

例えば、IMX＝A+>B+>C+>Dとして定義されたIMXが表示された時にIMXの一部がライトペンで指適されると、IMXを構成する最も下位のイメージが指適の対象とされる。即ち識別されるのはA、B…DでIMXではない。IMXを一つの論理的なまとまりとして定義するには、ここで述べるINSTANCEファンクションが使用される。

先の例でIMXをINSTANCEファンクションでグルーピングした例を下に示す。

```
IMY＝INSTANCE（IMX）
```

IMYを表示すると、前の例と全く同じ図形が表示されるが、ライトペン・ディテクトに対してはIMYが一つの対象となる。

⑤ イメージ・アトリビュート

前にも述べたように、イメージは各種のアトリビュートを持つ事が出来る。アトリビュ

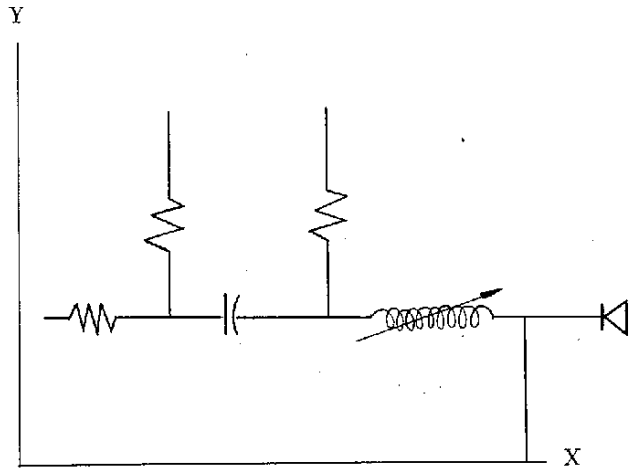
知らされるまで継続される）

この例では、動作を変位で与えるOFFSET オプションが使われているが、このほかに、回転量で与えたりスケール量で与えたりする事も出来る。以下にその他のオプションを示す。

ROTATE	THICKNESS
OFFSET	PERSPECTIVE
SCALE	VIEWPLANE

以下に、〔図6－8〕の回路図をイメージ・エクスプレッションとして表現したプログラム例を示す。

```
DECLARE(NODE1,NODE2,NODE3) VECTOR,
        CIRCUIT IMAGE:
DECLARE
 (RESISTOR,CAPACITOR,VARINDUCT,
   DIODE)RETURNS(IMAGE);
NODE1=2X+2Y;
NODE2=3X+2Y;
NODE3=4X+2Y;
CIRCUIT=(RESISTOR*>180Z)@NODE1+
          (RESISTOR*>90Z)@NODE1+
          CAPACITOR@NODE1+
          (RESISTOR*>90Z)@NODE2+
          VARINDUCT@NODE2+
          DIODE@NODE3+
          NODE3|||(4X+0Y);
CAPACITOR: PROCEDURE RETURNS(IMAGE);
 DECLARE CAP IMAGE;
 CAP=(0.0X+0.0Y)|||(0.45X+0.0Y)+
       (0.45X+0.2Y)|||(0.45X-0.2Y)+
       ARC((1.0X+0.0Y),(0.597X+0.1Y),60.0Z)
       +(0.05X+0.0Y)|||(1.0X+0.0Y);
 RETURN(INSTANCE(CAP));
END CAPACITOR;
```



〔図6－8〕

(4) アテンション・ハンドリング

割り込み処理は、ＰＬ／Ｉの標準機能を拡張して使っている。

拡張された条件名は次の通りである。

DESIGN, FUNCTION KEY, KEYBOARD, REGIONBOUNDARY
IMAGEINTERRUPT, LIGHTPEN, CHARACTERDISPLAY
IMAGEDISPLAY.

各要因に対して割り込みの禁止、許容はデフォルト値が決められて居り、これを再定義するには、ラベル接頭語を使用する。

又、発生した割り込みは、各ＦＩＬＥごとにキューイングされ割り込みキューが作られる。

割り込みキューから、割り込みが取り出されるのは次の３つの場合である。

① ＴＡＫＥ文、又はＷＡＩＴ文が実行された時

② ファイルが閉じられた時

③ タスクが終了した時

以下に割り込み処理ステートメントの例をあげる。

割り込み処理ステートメントの例

```
1)  TAXE FILE(X) LIGHTPEN FUNCTIONKEY;
2)  TAXE LIGHTPEN; /* ALL FILES */
3)  TAKE FILE(X) NONE; /* CLEAR QUEUE FOR FILE X*/
4)  TAKE FILE(X1) ALL,"FILE(X2)" DESIGN;
5)  ON LIGHTPEN(ADAGE) GO TO LPEN;
6)  ON ENDKEY(TUBE) BEGIN; END;
7)  ON LIGHTPEN(X) IDENT(A,B,C)
        BEGIN;
            GO TO LAB(IDENT);
            LAB(1):
                ...
            LAB(2);
                ...
            LAB(3);
                ...
            END;
```

最後にＧＰＬ／Ｉで組まれたプログラムの例として、次のような動作をするためのプログラムを示す。

まず、円とその中心点を表示する。その円をピックすると円は大きくなり、中心点をピックすると小さくなる。その動作をくり返し、プログラムを終了させたい時には、キャラクタ・ストリング"ＥＮＤ　ＰＲＯＧＲＡＭ"をピックする。

```
SAMPLE: PROCEDURE OPTIONS(MAIN);
S1: DECLARE
        IBM2250 GRAPHIC FILE,
        (CENTER,CIRCLE,MENU) ACTIVE(IBM2250) IMAGE,
        RADIUS FLOAT BIN(21) INIT(2.0),
        LP(3) LABEL;
```

```
S2:  CENTER＝(5.0X+5.0Y);
S3:  CIRCLE＝ARC(5.0X+5.0Y),VECTOR(RAI)IUS+5.0,5.0),360.0);
S4:  MENU="END PROGRAM",2"@(.1X+.1Y);
S5:  PUT FILE(IBM2250) SIGNAL;
S6:  ON LIGHTPEN IDENT(CIRCLE,CENTER,MENU) BEGIN;
S7:  GO TO LP(IDENT);
     LP(1):/* DETECT ON CIRCLE
        RADIUS＝MAX(RADIUS+0.25,6.0);
        CIRCLE＝ARC(5X+5Y),VECTOR(RAI)IUS+5  5.0),360Z);
        GO TO END;

        LP(2):/* DETECT ON CENTER */
           RADIUS＝MIN(RAD1US-0.25,0.5);
           CIRCLE＝ARC((5.0X,5.0Y),VECTOR(RADIUS+5.0,5.0),360Z);
           GO TO END;
        LP(3):/* END PROGRAM */
           ERASE FILE(BW2250);
           GOMPLETION(ENDPROGRAM)＝"1"B;
     END:END;
S8:  WAIT(ENDPROGRAM);
END SAMPLE;
```

プログラム例の説明

S1 ： IBM2250をファイルとして宣言
CENTER、CIRCLE、MENUはACTIVE属性をもつイメージとして宣言する。即ち、プログラム内でこれらのイメージが操作された時、CRT画面にも反映される。

S2 ： 円の中心点CENTERは(5.0,5.0)の点として定義され、IBM2250ファイルに出力される。

S3 ： 円CIRCLEは半径2インチの円として定義されIBM2250ファイルに

出力される。

S４　：　ＭＥＮＵは、高さ0.２インチ、文字が置かれる左下端の点（ 0.１，0.１）のキャラクタ・ストリング＂ＥＮＤ　ＰＲＯＧＲＡＭ＂で定義される。

S５　：　コンソール・アラームが鳴り、ユーザにプログラムのスタートをしらせる。

S６　：　ライトペン割り込みが起った時の動作をＥＮＤ：ＥＮＤ；までのステートメントで定義する。ＩＤＥＮＴオプションでは３つのイメージを指定している。

S７　：　ライトペンで３つのイメージのうちいづれかがピックされた時、ＩＤＥＮＴファンクションは指定されたイメージに対応させて１か２あるいは３を返す。それによって、ＬＰ（１），ＬＰ（２），ＬＰ（３）にそれぞれ分岐する。

ＬＰ(1)：　円の半径を0.２５インチ拡張する。その時、即にディスプレイされているＣＩＲＣＬＥは、この拡張されたＣＩＲＣＬＥにおきかえられる。

ＬＰ(2)：　円の半径を0.２５インチ縮少して、ＬＰ(1)と同じ動作が行われる。

ＬＰ(3)：　ＣＲＴ画面がクリアーされる。プログラムを終了させるためのCOMPLETIONでイベント変数ＥＮＤ ＰＲＯＧＲＡＭを設定する。

S８　：　イベント変数ＥＮＤＰＲＯＧＲＭで事象を指定してＷＡＩＴ状態に入る。

APPENDIX.

1. グラフィック・オペレータ

+>	inclusion
III	Connection
@	Positionning
< >	Scaling
*>	Rotation

2. アトリビュートの種類

アトリビュート	適用
ACTIVE(---)	IMAGE
QVIESCENT(---)	IMAGE
ASSOCIATE(---)	FILE
ATTRIBUTE	DATA
BLANKED	IMAGE
UNBLANKED	IMAGE
CANONCIAL	ENTRY
COLOR(---)	ENTRY
DASHED(---)	IMAGE
DATAFORM(---)	DATA
DEFINED	IMAGE
DETECTABLE	IMAGE
DESIGN	FILE
DISPLAY	FILE
STORAGE	FILE
FLICKER(---)	IMAGE
GRAPHIC	FILE
HARDCOPY	FILE
SOFTCOPY	FILE
IMAGE(-)	DATA

INTENSITY(n)	IMAGE
LOCATION	IMAGE
PAGESIZE(V.)	FILE
PERSECTIVE(---)	IMAGE
PROTECTED	IMAGE
UNPROTECTED	IMAGE
REGION(---)	IMAGE
ROTATE(---)	IMAGE
SCALE (---)	IMAGE
STREAM	IMAGE
STRUCTURE(---)	IMAGE
STYLE(---)	IMAGE
THICKNESS(n)	IMAGE
UPDATE( )	FILE
VECTOR(n)	DATA
VIEWPLANE	IMAGE

3. ベクトル操作組込み関数

関　数　名	ファンクション
XPROD(V1,V2)	ベクトルの外積を求める
DPROD(V1,V2)	ベクトルの内積を求める
ANGLE(V1,V2)	ベクトル間の角度を求める
MAG(V)	ベクトルの大きさ(絶対値)を求める
VECTOR(S1,S2[,S3])	スカラーをベクトルに変換する
XMAG(V)	ベクトルの各要素の大きさを求める
YMAG(V)	
ZMAG(V)	

4. アトリビュート組み込み関数

（APPENDIX－Dのアトリビュート参照）

SCALE	PERSPECTIVE
REGION	ACTIVE
COLOR	FLICKER
BLANKED	INTENSITY
STYLE	DASHED
SHIFT	THICKNESS
ROTATE	PAGESIZE
VIEWPLANE	UPDATE

5. コンディション組み込み関数

関　数　名	ファンクション
IDENT	IDENTオプションで指定されたアイテム番号が返される。
LPDETCT	ライトペンでピックされたイメージが返される。

6. イメージ組み込み関数

関　数　名	ファンクション
INTERSECTION	2つの3Dイメージの相貫するイメージを表わす。
VIEW	3Dイメージを投影面に投影したイメージを表わす。
TEXT	キャラクタ・イメージを表わす。
ARC	弧又は円を表わす。
NULLI	空イメージを表わす。
I.MINTR	割り込みを発生したイメージを知らせる。
INSTANCE	グルーピング・ファンクション

その他CONCAT，SWEEP，LPTRACK等のファンクションもある。

7. イメージ構造を調べる組み組み関数

COPY(IM)　　イメージのストラクチャーをコピーする。

IMAGE ∅(A，B)　　イメージAに於けるイメージBのシーケンス番号を返す。

その他INIMAGE，SUBELEM，EXPAND等がある。

8. アトリビュート代入文について

一般形

{アトリビュート・データ・エレメント / ATTR(イメージ)} 〔，… 〕＝アトリビュート・エクスプレッション；

ここで、アトリビュート・エクスプレッションは次の様に定義される。

{アトリビュート / アトリビュート・データ・エレメント / アトリビュート・エクスプレッション / (アトリビュート・エクスプレッション)} [{+ / −} アトリビュート・エクスプレッション]

アトリビュート・エクスプレッションは左から右に評価される。

(例)

ATTR(ALPHA)＝ATTR(BETA)−REGION(BETA)+COLOR(RED)

9. イメージ代入文

一般形

イメージ〔，… 〕＝イメージエクスプレッション

ここでイメージ・エクスプレッションは次の様に定義される。

{
ベクトル・エクスプレッション
イメージ
イメージ・エクスプレッション {+> / |||} イメージ・エクスプレッション
イメージ・エクスプレッション {* > / @ / <>} ベクトル・エクスプレッション
(イメージ・エクスプレッション)
テキスト・アイテム
}

4—7 METAVISU

概　　要

METAVISUはIRIAに於ける、グラフィック研究プロジェクトの名称である。グラフィック・システムのソフトウェアを総称してMETAVISUと呼んでいるらしい。

正式な言語名称は、特に述べられていないので、ここでは便宜上METAVISUと呼ぶことにする。

プロジェクトMETAVISUについては次の様に紹介されている。

「IRIAでは、CII10070の下で、汎用のグラフィック・ソフトウェアMETAVISUを開発した。

これは、ESOPEシステム（T．S．Sシステム）の一環を為すもので、真の意味でのマン・マシン・コミュニケーションを可能にするグラフィック・ソフトウェアである。インタラクティヴ・グラフィック・システムとして要請される機能を満足するため、次の4つの機能を考慮した。

① 非グラフィカルな機能（通常の言語に於ける演算処理機能）

② ダイナミックなデータの構造化機能

③ イメージ、生成、操作機能

④ インタラクティヴ・プロセスの記述機能

使い易くするためには、特に①〜③はハイ・レベルであることが必要である。我々はこれを会話型PL／Iをベースに作成することにより、この要求を満している。」

ここで述べられているように、METAVISUはPL／Iをベースとする汎用グラフィック言語である。

グラフィック・データーの表現には、グラフィック・プロセデュアと呼ばれる形式を採用し、高度な図形記述、操作が出来る様になっている。又関連データの処理に関してはLEAPタイプのデータ・ストラクチャーを備え、強力に、これに対処し得る能力を備えている。

その他、特にマルチプロセッサー・システムと云う事を考慮して、インタラクション言語をして全く別の言語を設けると云う特徴的な扱いをしている。

開　　発

IRIA（L' INSTITUT DE RECHERCHE D'INFORMATIQUE ET D'AUTOMATIQUE）仏国

使用システム

ＴＳＳシステムＥＳＯＰＥの下で動作する。

そのハードウェア構成を〔図７－１〕に示す。

セントラルプロセッサーＣＩＩ１００７０の下に２台のグラフィック・コンソールＶＵ2,０００がある。

一台は直結で、もう一台は電話回線で結ばれたサテライト・プロセッサーＳＩＮＴＲＡ１０１６の下にある。

ホストプロセッサーＣＩＩ１００７０は、Ｇ．ＭＩＸ＝2.2μＳの処理能力を持ち、主記憶容量は１２８KW，１３２ｂｉｔｓ／Ｗである。又コンソールＶＵ2,０００の構成は次の様になっている。

○ＣＲＴ

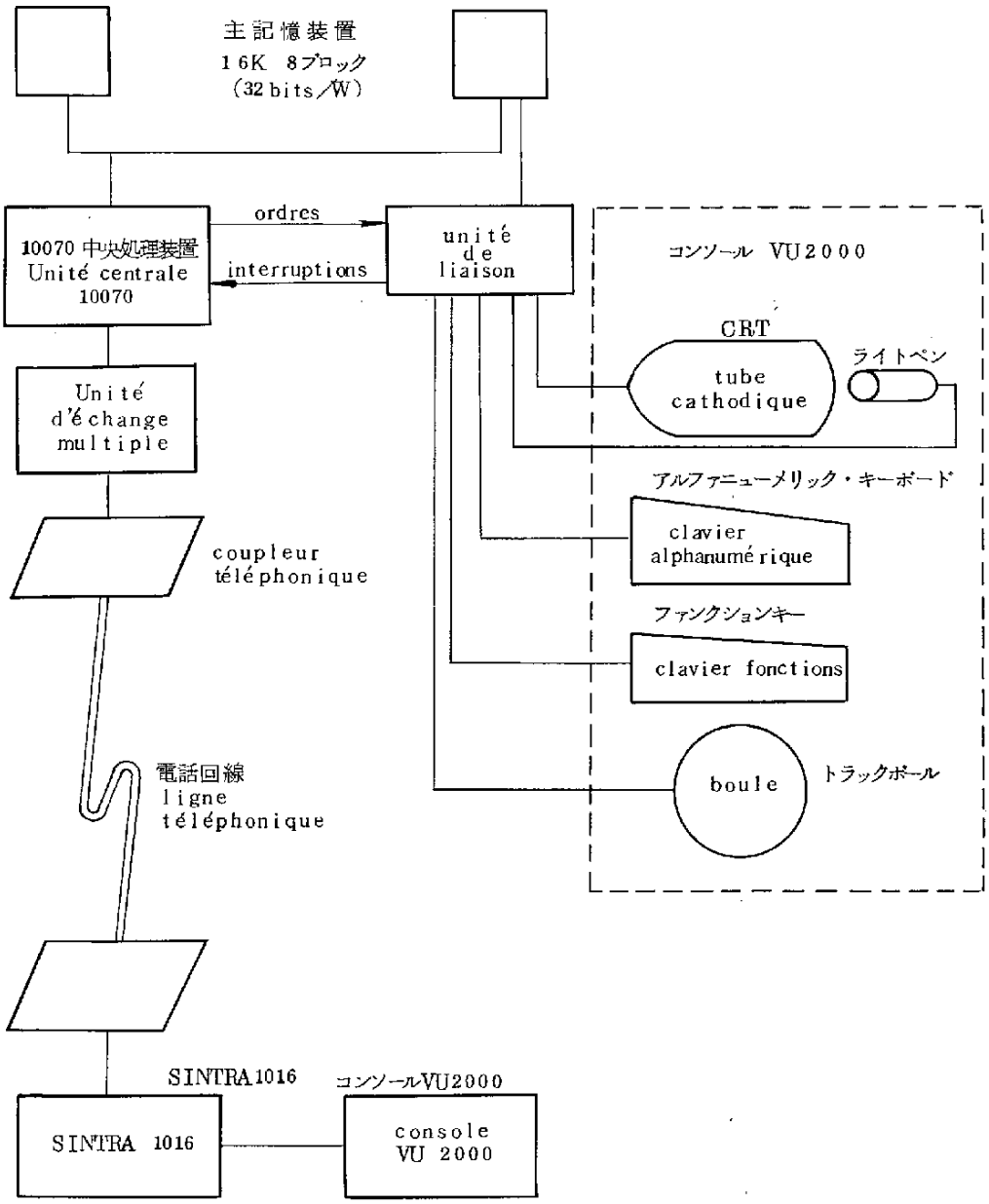
○ライトペン

○ファンクション・キー

○アルファニューメリック・キー

○トラッキング・ボール

○ディスプレー制御装置


主記憶装置
16K 8ブロック
(32 bits/W)
10070 中央処理装置
Unité centrale
10070
ordres
interruptions
unité
de
liaison
コンソール VU2000
CRT
tube
cathodique
ライトペン
Unité
d'échange
multiple
アルファニューメリック・キーボード
clavier
alphanumérique
coupleur
téléphonique
ファンクションキー
clavier fonctions
boule
トラックボール
電話回線
ligne
téléphonique
SINTRA1016
コンソールVU2000
SINTRA 1016
console
VU 2000

〔図7－1〕

言語タイプ

図形処理とデータ・ストラクチャーを扱う部分は、ＣＰＬ／Ｉと呼ばれる会話型ＰＬ／Ｉをベースにして作成されている。インタラクションを扱う部分は全く別の言語として備えている。インタラクション言語についての詳細は明らかにされていない。

発表論文

Bulletin de L'IRIA №8　1971

「Un software de base pour console graphique」

機　　能

(1) 非グラフィカルな機能

ＰＬ／Ｉに同じ。

(2) データ・ストラクチャー・ハンドリング

データ・ストラクチャーは、特にグラフィックの分野に限らず、汎用的に利用出来る様なものを目標として作成された。言語体系はＰＬ／Ｉを基本にしている。構造、機能はほぼＬＥＡＰと同じである。連想記憶についてはＬＥＡＰで述べられているので、ここではデータ・タイプと、処理ステートメントについてのみ紹介する。

○データ・タイプ

アイテム、トリプル、セット、アイテム変数がある。各々についてはＬＥＡＰを参照されたい。

（例１）　アイテムの宣言例

```
DCL  ITEM  FATHER(0:3,-1:4)CHARACTER(6)
       VARYING;
```

（例２）　トリプルの生成と除去

```
MAKE  <FATHER,JEAN,PIERRE>;
ERASE<FATHER,JEAN,PIERRE>;
```

（例３）　セットにメンバーを追加、削除する。

```
INCLUDE  JEAN  IN  SONS;
REMOVE  JEAN  FROM  SONS;
```

（例４）　アイテム変数の代入文

```
ASSIGN  X  ←  FATHER;
```

（例５）　セット・エクスプレッションをセットに代入する。

```
CHILDREN = SONS U DAUGHTER;
```

（例６）　アソシェーティヴ　FOR　EACH　文の例。

```
FOR EACH <FATHER,≒X,PIERRE>
DO INCLUDE X IN SONS;
```

＊　≒XはＬＥＡＰのローカル変数にあたる。

（例７）　次の例は、ＰＩＥＲＲＥの息子、全員の年令の和を求めるプログラム例である。年令は各アイテムのデータとして既に定義されているものとする。

```
    I = 0 ;
FOR EACH <FATHER,X,PIERRE>
DO I = I + DATUM(X) ;
```

(3)　図形記述表示機能

グラフィック・データの扱いは、ＤＩＡＬとよく似た方法をとっている。

ＤＩＡＬのディスプレー・プロセデュアにあたるものをＭＥＴＡＶＩＳＵではグラフィック・プロセデュアと呼んでいる。

図形は、グラフィック・プロセデュアの定義と云う形で表現され、その階層関係はグラフィック・プロセデュアのネストとして表現される。

図形操作に関しては、ＤＩＡＬより、さらに機能が強化され、座標移動、スケーリングの他に回転、マスキング、ウィンドーイング等の操作を加える事が出来るようになっている。（マスキング、ウィンドーイングについては後述する）又、図形出力はＣＡＬＬ文で直接グラフィック・プロセデュアを呼び出す形で行われる。

図形表現に、グラフィック・プロセデュアと云う手段を採用した点に関して、作成者等は次の様に述べている。

①　グラフィック・プロセデュアの処理のために、ベース言語のコンパイラーを、一部修正する事を余儀なくされた。

②　図形構造は、データ・ストラクチャーを必要としないから、領域が節約されるだろう。又表示データの生成が直接的だから処理速度は速いだろう。

③　グラフィック・プロセデュアの中では、グラフィック・ステートメントばかりで無く、通常のＰＬ／Ｉステートメントがそのまま使えるので大変便利である。（ＰＬ／Ｉが分っていれば、いくつかのグラフィック・ステートメントだけを理解すればすむ）

○図形出力文

基本的な図形を定義するための図形出力文として次の５つのグラフィック・ステートメントが用意されている。

① 位置付け命令

MOVE〔WITH{キャラクター・ストリング / キャラクター変数}〕Σ{TO Xi,Yi / OF △Xi,△Yi};

TO Xi, Yi は現座標系の(Xi,Yi)にビーム位置を設定する。

OF △Xi,△Yi は現位置から△Xi,△Yi だけ変位した点にビーム位置を移す。

WITHオプションはその位置にキャラクター・メッセージの表示をすることを指示する。

(例)

```
TPOS
:
DCL TL CHAR(12) INITIAL(≪＋＊＋＊＋＊＋＊＋＊≫);
:
DO 1=1 TO 12;
MOVE WITH SUBSTR (TL,I,1)TO TPOS (1,1)
TPOS(1,3);
END;
```

② 直線の表示

LINE 〔WITH t〕〔FROM Xo,Yo〕Σ{TO Xi,Yi / OF △Xi,△Yi}

FROMオプションは直線の始点を指示する。省略時は現在位置が指定されたものとする。

TO,OFについては①と同じである。

WITHオプションは、tの値によって、直線の種類、点線、鎖線、ブランク・ラインを指示することが出来る。

(例) サイン・カーブを表示するプログラム例

```
MOVE TO 0,0 ;
PAI=3.141 ;
P =PAI/10. ;
DO I=0 TO 20 ;
```

```
LINE TO P*I,SIN(P*I) ;
END ;
```

③ テキストの表示

TEXT{キャラクター・ストリング / キャラクター変数}〔WITH D,H〕

現位置から指示したキャラクター・ストリングが表示される。

WITH・オプションはD，Hでキャラクタの方向（縦，横）とキャラクタの大きさを指示する。

（例） TEXT "EXAMPL"

④ ビームのコントロール

LIGHT I,S,C

表示する時の明るさ、色等を指示するもので、この命令が実行された以後はずっとこの制御が有効になる。

Iは輝度，Qは点滅，Cは色を指示する。

⑤ RLIGHT

LIGHTで指定した制御情報を、以前のLIGHTで指定された制御情報にもどす。

○グラフィック・プロセデュア

グラフィック・プロセデュアは1つの図形モデルとしての役割りを果す。

グラフィック・プロセデュアの定義は、PL／Iのプロセデュア定義と同様な形をとり、通常のPL／Iステートメント及びグラフィック・ステートメントから構成される。

呼び出しも、プロセデュアの呼び出しと全く同じ形式、（CALL ステートメント）で行われるが、アーギュメント以外に、次の9つのオプションを指定する事が出来る。

① スケールオプション

SCALE a，b： a，bで、それぞれX方向、Y方向のスケリング・ファクターを与える。

② 座標回転のためのオプション

ROT r： rで回転角を与える。

③ 座標移動のためのオプション

AT x，y： x，yでローカルな座標（x，y）を与える。

④ 変換マトリックス・オプション

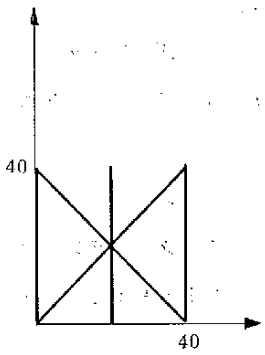
①～③の回転、移動、スケーリングのための情報は、次の形式の変換マトリックスとして与える事も出来る。

TRANS t_{11} , t_{12} , t_{21} , t_{22}

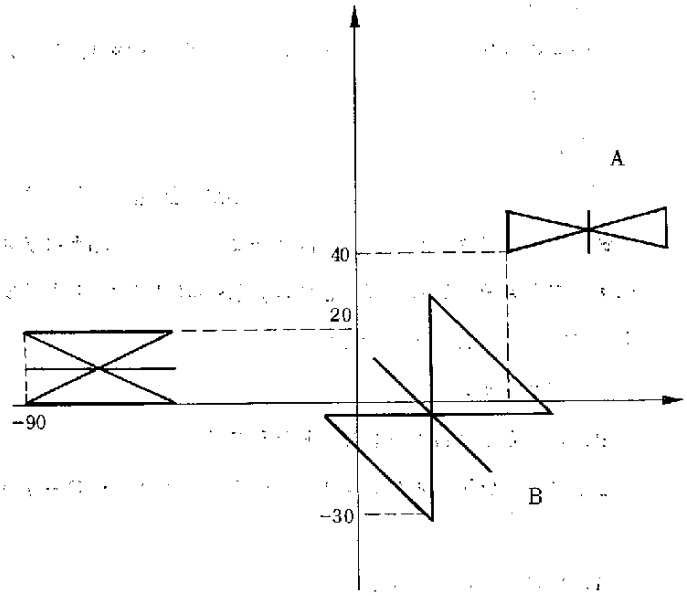
以下に①～④のオプションを利用して図形を記述した例をあげる。

（例） 次のプログラムは、〔図７－２〕を１つの図形モデルとして、グラフィック・プロセデュアPAPILLONとして定義し、これをもとにして〔図７－３〕に示された図形A，B，Cを画面に表示する例である。

```
    プログラム                                          コメント
PAPILLON: PROCEDURE ;                          ┐ グラフィックプロセデュ
LINE OF 40,40 OF 0,-40 OF-40,                  │ アPAPILLONの定
40 OF 0,-40 ;                                  │ 義
MOVE OF 20,0 ;                                 │ 〔図７－２〕を定義した
LINE OF 0,40 ;                                 │ ことになる。
END ;                                          ┘
CALL PAPILLON AT 40,40 SCALE 1 ,1／4 ……〔図7－3〕
                                                   Aにあたる。
CALL PAPILLON AT 20,-30 ROT PAI／4 ……〔図7－3〕
                                                   Bにあたる。
CALL PAPILLON AT -90,20 SCALE 0.5,1 ROT
-2*PAI
```



〔図７－２〕



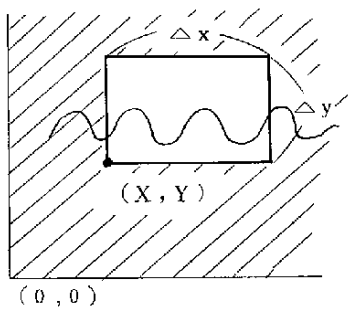
〔図７－３〕

⑤　マスク・オプション

ＷＩＴＨＩＮ　Ｘ，Ｙ，△ｘ，△ｙの形式で対象とする図形モデルの一部（辺の長さが△ｘ，△ｙの長方形の部分）を取り出す操作を指定する事が出来る。

Ｘ，Ｙ　、△ｘ，△ｙは〔図７－４〕の様に決められている。

図で斜線で示した部分が取り除かれる。

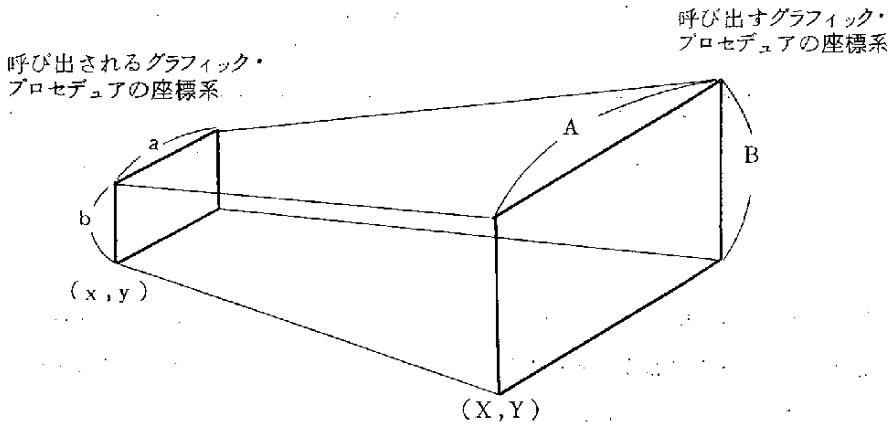


〔図７－４〕

⑥ ウインドー・オプション

ＯＮ　ＴＯ　Ｘ，Ｙ，△ｘ，△ｙの形式で、対象とする図形モデルを呼び出す側の座標系の、どの位置に、どれだけ引き伸ばして配置するかを指定する。

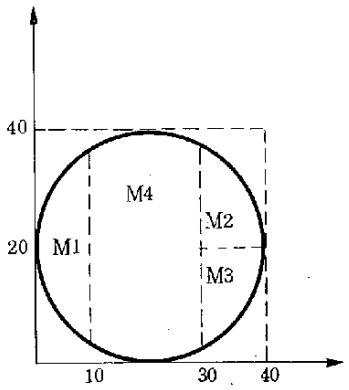
〔図７－５〕にＷＩＴＨＩＮとＯＮ　ＴＯの関係を示す。



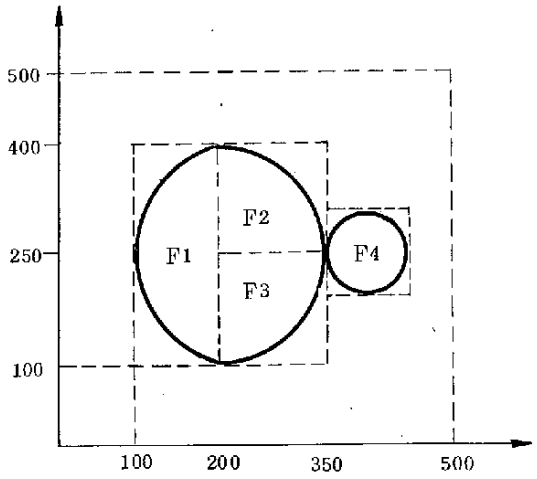
〔図７－５〕

ＷＩＴＨＩＮ，ＯＮＴＯオプションを使用して、謂わゆる切りばり細工の様な事が出来る。

次の例は、〔図７－６〕の円をグラフィック・プロセデュアＣＥＲＣＬＥとして定義しておき、これを図形モデルとして、〔図７－７〕の複合図形をＤＥＳＳＩＮとして定義し、表示する過程を示したものである。



〔図7－6〕



〔図7－7〕

```
 DESSIN:PROCEDURE  ;                          DESSINの定義
 CALL CERCLE WITHIN 0,0,10,40……F1に対応する
ON TO 100,100,100,300  ;
 CALL CERCLE WITHIN30,20,10,20……F2に対応する
ON TO 200,250,150,150  ;
 CALL CERCLE WITHIN 30,0,10,20……F3に対応する
ONTO 200,100,150,150  ;
 CALL CERCLE WITHIN 0,0,40,40……F4に対応する
ONTO 350,200,100,100  ;
 END
 CALL DESSIN WITHIN 0,0,500,500……DESSINの表
ONTO 0,0,1000,2000  ;                          示
```

⑦ 表示条件をコントロールするためのオプション

呼び出すグラフィック・プロセデュアの表示条件（明るさ、色等）を指示する。

```
 CALL DESSIN LIGTH I,S,C
```

⑧ 表示された図形に識別を与えるためのオプション

AS{数式 / アイテム名称}

この名前はライトペン・ピックの時に返される名前となる。

ここで識別名にアイテム名がそのまま使えるのは大変便利である。

これによってデータ・ストラクチャーのアイテムと、グラフィック・プロセデュアの対応をはかる事が出来るからである。（プロブレム・データとグラフィック・データの関連をつける事が出来ると云う事）。

○グラフィック入出力

特別に出力ステートメントと云うのは無い。ＣＡＬＬステートメントでグラフィック・プロセデュアを呼び出すことによって出力される。

(4) アテンション・ハンドリング

前にも述べたが、インタラクション言語は、又別の言語として備えていると云う。詳細については述べられていないので、以下にその構想についてだけ紹介しておこう。

―インタライクティヴ・プロセスを定義するには、有限オートマトンの概念が便利である。

この構想のもとに、割り込み処理プロセスを記述する言語を考えた。

又、マルチプロセッサー方式の時にはロードシェアと云う事も考慮しなくてはいけない。この点に関しては次の様に考えた。

割り込み処理は単純なカテゴリー（位置情報のフトアーとか、単純な図形処理等）と複雑なカテゴリー（問題モデルへのアクセス、解析等）が考えられるが多くは前者で占められている事が多い。

そこで、単純な処理機能を持った言語を別に備えることにした。この言語で書かれたプログラムはサテライト側で動作するプログラムとなる。又このプログラム内で処理し切れないものは、メインプログラムで定義されているプロセデュアを呼び出して処理させる事が出来る様になっている。

この様な方法は、プログラムが２本に分割され別々にコンパイルされ実行されるという繁雑さはあるが、きめの細かい割り込み処理が実現出来ると云う点では優れているだろう。―

4—8 CORAL(Class Oriented Associative Language)

概　　要

ＣＯＲＡＬは１９６４年Ｌｉｎｃｏｌｎ Ｌａｂｏｒａｔｏｒｙで開発された、データ・ストラクチャ操作のためのアクションサブルーチンとマクロステートメントをもつアソシェテイブ・プログラムミング・ランゲッジである。このシステムはＴＸ—２の下にインプリメントされ、マクロステートメントの表現にはＴＸ－２のアセンブラ・システムＭＡＲＫ Ⅳと Ｌｉｎｃｏｌｎ Ｗｒｉｔｅｒ Ｋｅｙｂｏａｒｄのキャラクタ・セットが使用されている。

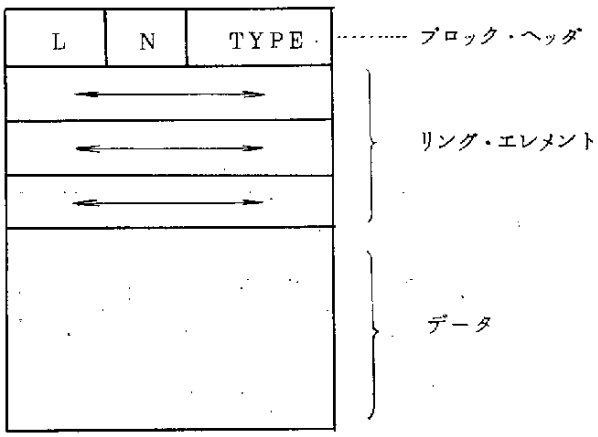
ＣＯＲＡＬのデータ・ストラクチャは、従来のリング・ストラクチャ（ＳＫＥＴＣＨＰＡＤ）と比較すると次のような特徴をもつ。従来はリング・リストが両方向リストとして表現されていたのに対しＣＯＲＡＬでは、スタート・タイと呼ばれるリング・スタート・エレメントへの直接のポインタを導入している。リングの上には、スタート・タイ、バック・タイと呼ばれるポインタが交互に置かれる。従ってリングの上の任意の位置から、そのリングスタートを調べたい時には、次々にリングをたどることなく、このスタート・タイを用いて、直接スタート・エレメントに到達することができる。

もう一つの特徴はナブの導入である。ＳＫＥＴＣＨＰＡＤでは関連を表現するリング・スタートとリング・タイの動的な拡張ができなかったが、ＣＯＲＡＬでは二つのリングをナブで結びつけることにより、リング・タイの動的な拡張を可能にしている。しかしリング・スタートの動的な拡張性に関しては考慮されていない。

又ＣＯＲＡＬにおけるストラクチャ操作は、構成要素のフィジカルなアドレスを意識したポインタ操作を通じて行われる。そしてポインタの演算、参照のためにはアキュムレータが使用されている。

データ構造

データ・ストラクチャの基本的な要素としてブロックがある。ブロックは〔図８－１〕で示すように、ブロック独自の情報を保持するブロック・ヘッダとデータ部、それに各ブロック間の関連を表現するリング・エレメントで構成されている。



〔ブロックの内部構造〕

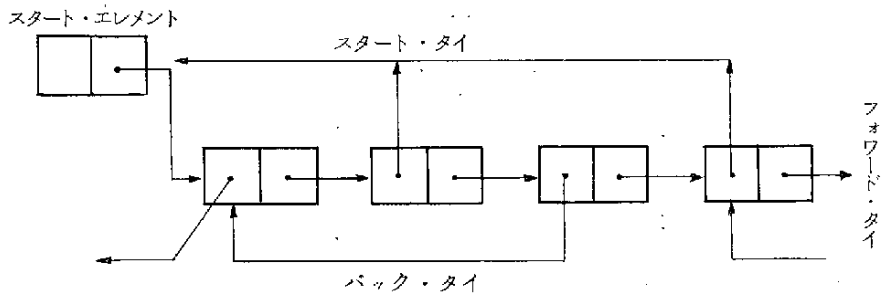
〔図８－１〕

ブロック・ヘッダ：ブロックの大きさ、リング・エレメントの個数、ブロック・タイプを示す。

リング・エレメント：リング・エレメントには、スタート・エレメントとエレメントがあり、他のブロックとの関連を表現するために用いられる。

データ：ブロックが保有する独自の情報を蓄える領域である。

〔図８－２〕は、リング上でリング・エレメントがどのようにリストされるかを示したものである。



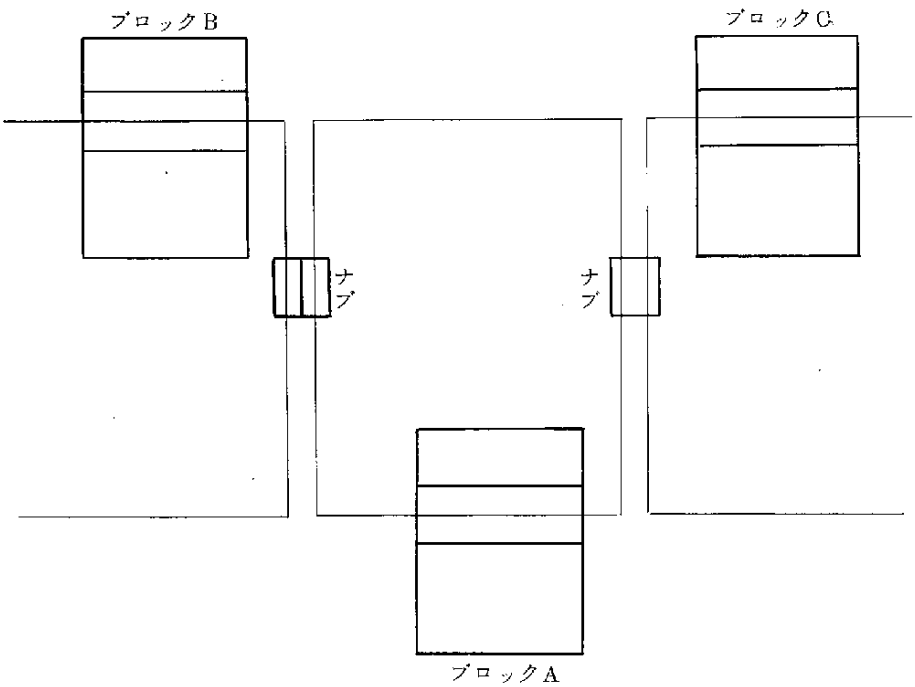
〔図８－２〕

リング上には必ず一つのスタート・エレメントがあり、他のエレメントがそのスタート・エレメントに従属する形をとっている。リング上のエレメントには交互にバック・タイとスタート・タイが持たしてある。スタート・タイは、スタート・エレメントへの直接のポインタであり、バック・タイは一つ前のエレメントへのバック・ポインタである。この二種のポインタを用いることによって検索効率をあげている。

例えば、エレメントがどのリングに属するかを知りたい時、リング上のポインタを次々にたどることなく、スタート・タイによって直接スタート・エレメントが得られる。又バック・タイとフォワード・タイによって前後のエレメントのアドレスを知ることができるため、任意の位置にエレメントを挿入したり削除したりする事も容易に出来る。

スタート・エレメント、エレメントは創成時にそれに割り当てられる領域長が決定されているが、エレメントに関してはこれを動的に拡張する必要がおきると、ナブと呼ばれる特定のコネクターを設けて、この拡張を可能にしている。

例えばブロックＡの、ブロックＢとブロックＣに対する関連を表現するには、ブロックＡのリングとブロックＢのリングを、又ブロックＡのリングとブロックＣのリングをナブで結びつける。〔〔図８－３〕参照〕



〔図8－3〕

ステートメント機能

ＣＯＲＡＬでは、データ・ストラクチャ操作は全てポインタを指示して行われ、ポインタの演算、ポインタの参照は、アキュムレータを仲介にして行なわれる。又オペレータはポインタ・パラメータとニューメリック・パラメータを持ち、オペレータ・シンボルはＬｉｎｃｏｌｎ Ｗｒｉｔｅｒ Ｓｙｍｂｏｌが使われている。

主なＣＯＲＡＬオペレータの標準パラメータは次の７つである。

パラメータ・シンボル

A	アキュムレータ
P	ポインタ

Q	参照されるポインタ
N	ニューメリックな値
S	ポインタ
R	レベル
TB	ブロック・タイプ

さらに"→S"はマクロ終了後、アキュムレータの内容をポインタSに格納することを示す。又"☛R"はマクロ終了後、制御がステートメントRにわたされることを示す。

CORALのステートメント機能の主なものとしては次の5つがあげられる。

(1) ポインタ操作

(2) ブロックの創成・消去

(3) エレメントの挿入・削除

(4) テ ス ト

(5) リング上のエレメントに対する処理

以下に個々の機能に関して、ステートメントの例をあげて説明する。

(1) ポインタ操作

• ↑ P↑N → S☛R

Pはブロック内のあるロケーションをさすポインタである。そのロケーションにNプラスしたロケーションを求める。その値はアキュムレータに入っており、アキュムレータからポインタSに格納され、制御はステートメントRに移る。

即ち、ブロック内の処理に用いられる。

• ⊁ P⊁N → S☛R

ポインタPから、リング上を右まわりにN個目のエレメントのロケーションを求め、その値はポインタSに格納され、制御はステートメントRに移る。このマクロ・ステートメントは、リング上のエレメントの検索などに用いられる。

この他、ポインタ操作のためのマクロ・ステートメントがいくつかある。

(2) ブロックの創成・消去

ブロックは、マスタ・ブロックと呼ばれるブロックで定義されたブロック・タイプを指定することにより、そのタイプの形状で、マスタ・マスタ・ブロックにアロケートされる。マスタ・マスタ・ブロックとはリスト・ストラクチャ用の領域である。

• ⊕ X⊕L → S☛R

マスタ・マスタ・ブロックとして、Xで指示されるロケーションにリスト・ストラクチャ領域として、大きさLの領域をとる。

ブロックの先頭ロケーションがSに格納される。

- ⊖　　TB ⊖ L×LL → S ☛ R

ブロック長Lでリスト長のLLのブロック・タイプTBのマスタ・ブロックを創成する。マスタ・ブロックの先頭ロケーションはSに格納される。

- Ⓑ　　TB Ⓑ L → S ☛ R

タイプTB、ブロック長Lのブロックを創成する。

もしLが省略されても、タイプTBのマスタ・ブロックで指定された長さにする。

この他ブロック消去のマクロ・ステートメントがある。

(3) エレメントの挿入、削除

- ⊳　　P ⊳ Q → S ☛ R

リング上でポインタQの右側にポインタPを挿入する。もし、Pがリング・スタートであれば、エラーメッセージがだされる。

アキュムレータにはPの値が入り、その値をポインタSに格納する。

- ⓣ　　P ⓣ → S ☛ R

Pをリングからはずす。即ち、ポインタPは自分自身を指すようにする。もしPがリングスタートであれば何もしない。

この他リング消去、エレメントの挿入、削除に関するマクロ・ステートメントがいくつかある。

(4) テ ス ト

- H　　P H　JYES | JNO ☛ R

Pがリング・スタートを指しているか。もし指していればJYESへ、そうでなければJNOへ制御が移る。

- ▣　　P ▣ TB　JYES | JNO ☛ R

Pがタイプ TBのブロックを指しているか？

- K　　PKN → S ☛ R

ポインタPにNプラスしたロケーションの内容をアキュムレータにロードし、その値をSに格納する。

・ [I]　　P [I] → S ☛ R

ポインタPを含むブロック長をSに格納する。

この他ブロックのリスト長、ブロック・タイプなどを調べるマクロ・ステートメントがある。

(5) リング上のエレメントに対する処理

・ [▷]　　P[▷]ACTION ∧ PDL → S ☛ R

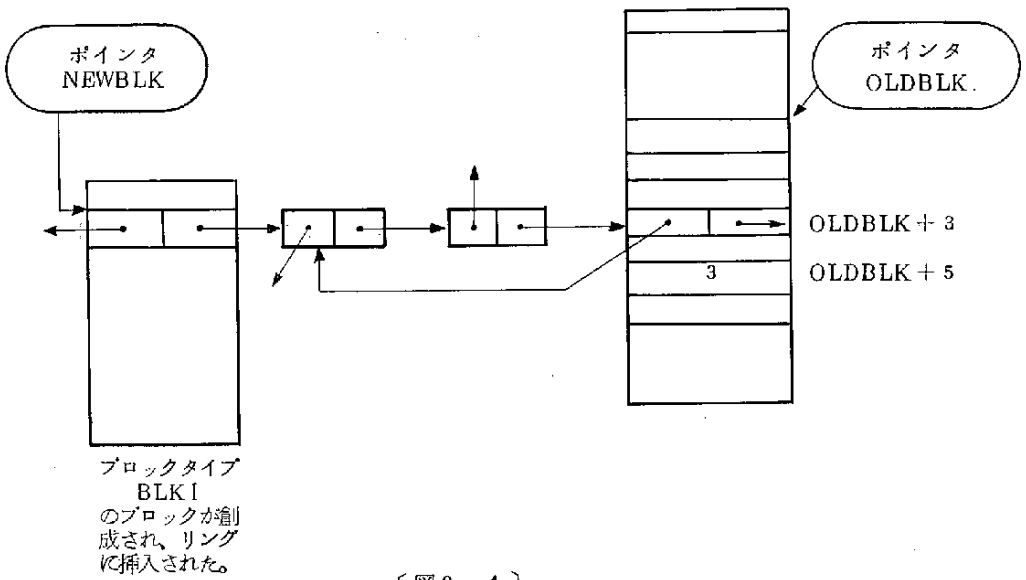
エレメントPを除いたリング上の各々のエレメントに対し、処理ACTIONをしながらリングPを右にまわる。通常Pはリング・スタートを指し示し、そのリング上のエレメントにある処理をしたい時に用いる。

これらのマクロ・ステートメントにおいてオペレータは、算術式のようにネストされることが許されている。次にその例をあげる。

〔例〕

(Ⓑ BLKI → NEWBLK) ⊜ (COLDBLK ↓ 3)↓(OLDBLK [K] 5))

タイプBLKIのブロックが創成される。このブロックをポインタOLDBLKに3プラスしたロケーションから、OLDBLKに5プラスしたロケーションの内容だけ、リングを左まわりに3移動したロケーションに挿入する。(〔図8－4〕参照)



〔図8－4〕

4—9 APL(Associative Programming Language)

概　　要

ＡＰＬは、General　Moter 社で開発された汎用アソシェーティブ・プログラミング．ランゲッジであり、図形処理言語に限らず、ＣＡＤ，ＩＲ等、相互に関連しあったデータを扱う必要がある分野を対象として考えられた。システムはＰＬ／Ｉをベースとして作成され、ＰＬ／Ｉのフルセットとしての機能に、ストラクチャ・ハンドリングのための若干の機能と領域管理の機能を持つページング・ソフトウエアが追加された形で実現されている。その機能上の特徴をあげると次の様になる。

(1) データ・ストラクチャ内の要素、又はそのデータをシンボリックに操作できる。

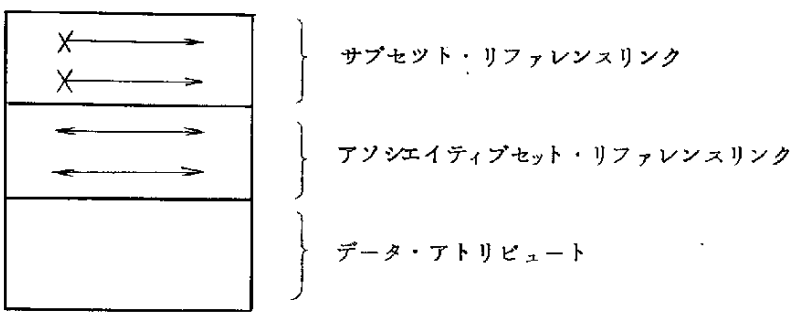
(2) ストラクチャ操作のために設けられたステートメントは、プロセデュア・コールの形式をとらずに、通常のステートメントの形式をとっている。（プリプロセッサが介入する。）

(3) ＰＬ／Ｉのポインタ修飾形式で、その周辺の要素をＮ次元に参照できる。

(4) ユーザが定義したり、参照したりするストラクチャ・ストレッジは絶えずページング・ソフトウエアが監視していて、必要に応じて主記憶、２次記憶の間でページの交換を行ってくれる。従ってユーザはコア・スペースの大きさを意識することなく、論理的には無限の広がりを持っ領域を持っていると考えてさしつかえない。

データ構造

データ・ストラクチャの基本的な要素はエンティティと呼ばれる。エンティティは、〔図９－１〕で示すように関連を表わすリンク部と独自の情報を格納するデータ部をもっている。



〔エンティティの内部構造〕

〔図９－１〕

サブセット・リファレンスリンク　：

エンティティが持つカテゴリを表わすもので、ＣＯＲＡＬのスタート・エレメントにあた

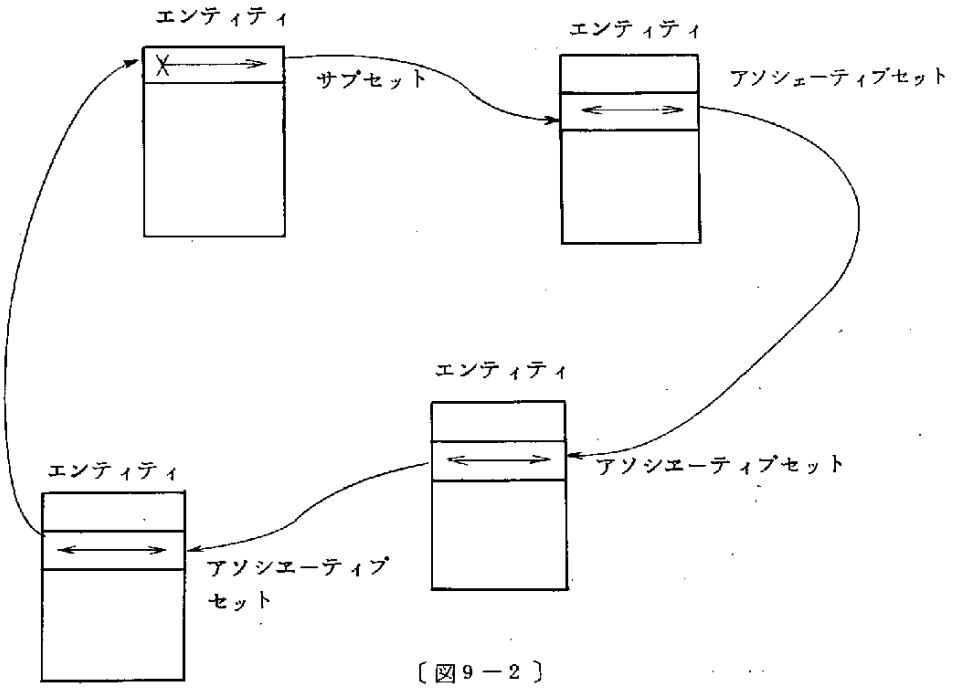
る。

アソシエーティブセット・リファレンスリンク　：

あるセットのメンバーになるためのリンク部であり、ＣＯＲＡＬのエレメントにあたる。

サブセット・リファレンスリンクとアソシエーティブセット・リファレンスリンクの関連を〔図９－２〕に示す。

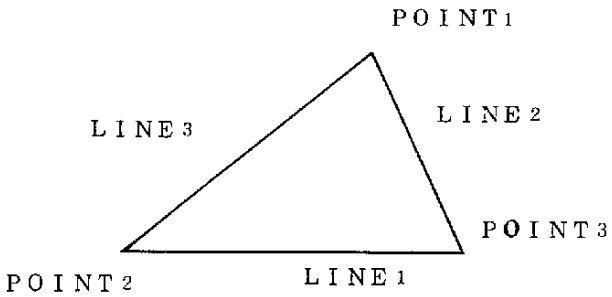
〔図９－２〕で示すようにリング上には、必らず一つのサブセットがあり、そのサブセットは従属するアソシエーティブセットをリストしている。そしてこのリングはあるカテゴリに属す、エンティティをまとめることにより関連を表現している。



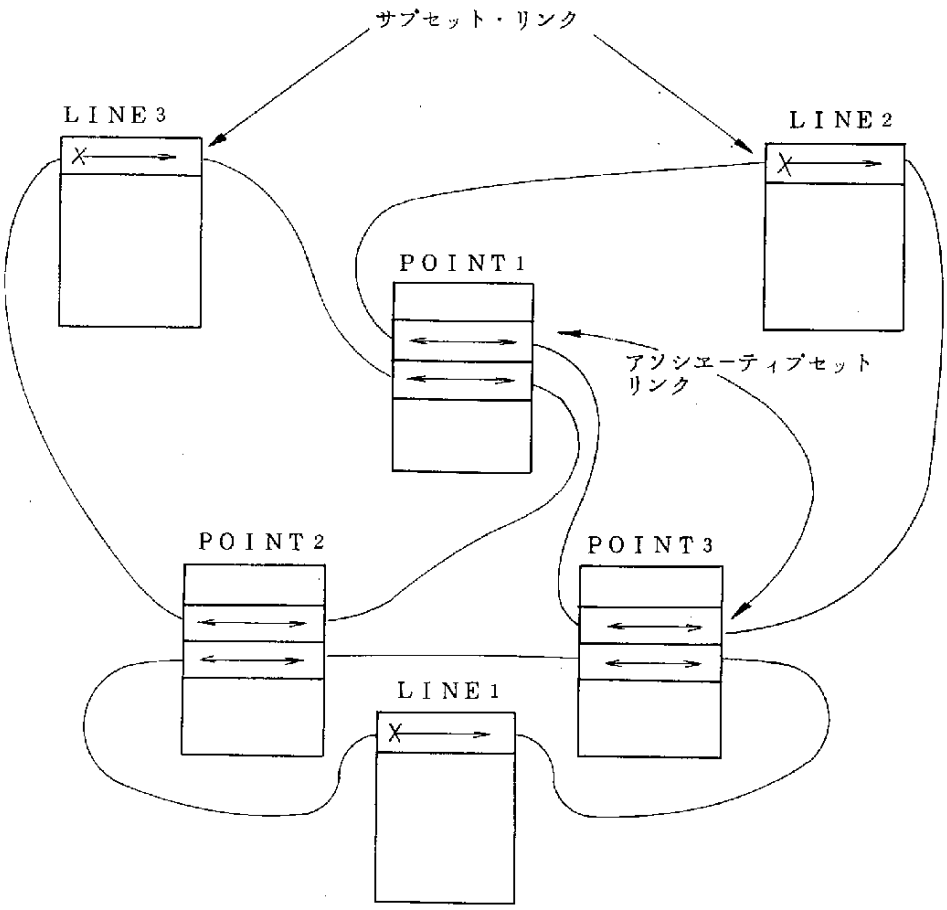
〔図９－２〕

以下に、ＡＰＬストラクチャを使って関連を表現する例を示す。

〔図９－３〕のような三角形の点と線の関連をＡＰＬストラクチャで表現すると〔図９－４〕のような構造となる。


POINT1
LINE3
LINE2
POINT3
POINT2
LINE1

〔図9－3〕


サブセット・リンク
LINE3
LINE2
POINT1
アソシエーティブセット
リンク
POINT2
POINT3
LINE1

〔図9－4〕

データ・アトリビュート：

データ・アトリビュートはエンティティの独自の情報を記述するデータで次のような二種類のアトリビュートがある。

① ダイレクト・アトリビュート

通常使われているデータ。

（例） 点の座標値

② アソシエーティブ・アトリビュート

他のエンティティ間、あるいは他のエンティティとそれがメンバーとなっているセット間の関連によって決定される値。

（例） 線の端点間の距離

ステートメント機能

ＡＰＬにおけるストラクチャ操作のためのステートメントは、機能上次の４つに分類される。

(1) データ・セットの宣言

(2) エンティティ・セットの創成

(3) ストラクチャ要素の参照

(4) ストラクチャ操作

(1) データ・セットの宣言

ストラクチャ内で使用されるエンティティのタイプをあらかじめ宣言文で定義しておく。エンティティはこのエンティティ・タイプを指定することにより、同じパターンで創成される。

エンティティ・タイプはプログラムの先頭で宣言され、サブセット、アソシエーティブセット、アトリビュートの詳細を構造体の形で次のように記述する。

```
DECLARE  1  エンティティ・タイプ              ENTITY,
         2  サブセット・ネーム                 SET,
            .....
         2  アソシエーティブセット・ネーム     MEMBER,
            .....
         2  アトリビュート・ネーム  ATTRIBUTE  データアトリビュート,
            .....
```

2　アトリビュート・ネーム　　ＡＴＴＲＩＢＵＴＥ＊，

……

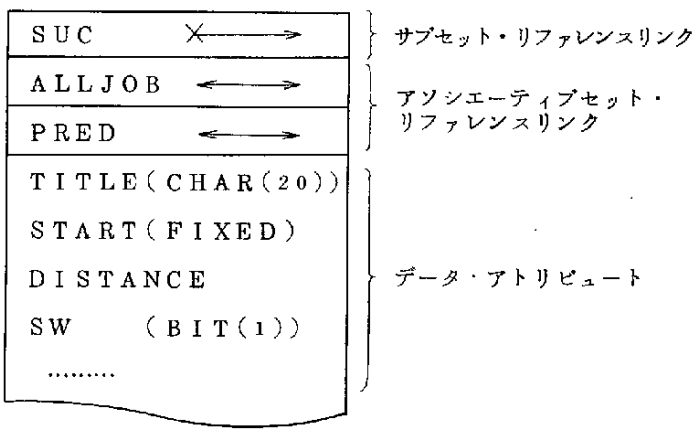
ここでＥＮＴＩＴＹ，ＳＥＴ，ＭＥＭＢＥＲ，ＡＴＴＲＩＢＵＴＥはリザーブドワードであり、次のような属性を表わす。

ＥＮＴＩＴＹ	エンティティ属性
ＳＥＴ	サブセット属性
ＭＥＭＢＥＲ	アソシエーティブセット属性
ＡＴＴＲＩＢＵＴＥ	ダイレクト・アトリビュート属性
ＡＴＴＲＩＢＵＴＥ＊	アソシエーティブ・アトリビュート属性

次の様に宣言されたエンティティ・タイプが具体的にどの様な形状に対応するかを以下に例示する。

```
（例）  ＤＣＬ  1  ＪＯＢ        ＥＮＴＩＴＹ
              2  ＳＵＣ        ＳＥＴ
              2  ＡＬＬＪＯＢ    ＭＥＭＢＥＲ，
              2  ＰＲＥＤ       ＭＥＭＢＥＲ，
              2  ＴＩＴＬＥ     ＡＴＴＲＩＢＵＴＥ  ＣＨＡＲ(20)，
              2  ＳＴＡＲＴ     ＡＴＴＲＩＢＵＴＥ  ＦＩＸＥＤ，
              2  ＤＩＳＴＡＮＣＥ  ＡＴＴＲＩＢＵＴＥ＊，
              2  ＳＷ          ＡＴＴＲＩＢＵＴＥ  ＢＩＴ(1)，
                 ……
```

エンティティ・タイプ　ＪＯＢ



(2) エンティティ，セットの創成

① エンティティの創成

エンティティの創成はＣＲＥＡＴＥステートメントで行われる。

CREATE　エンティティ・タイプ　CALLED　エンティティ変数；

既に宣言されたエンティティ・タイプを指定することにより、エンティティはそのエンティティ・タイプと同じ形状で記憶領域にアロケートされる。そしてこのエンティティを参照するためには、エンティティ変数と呼ばれるものをこのエンティティに一致させる。そして以後このエンティティ変数を使用してシンボリックに扱うことが出来る。

次の例では、エンティティ・タイプＪＯＢのエンティティが宣言され、さらにＸがエンティティ変数であることが宣言されている。そしてＣＲＥＡＴＥステートメントによりエンティティ・タイプＪＯＢのエンティティが創成され、エンティティ変数Ｘに対応づけられる。以後このエンティティはＸで参照できる。

```
（例）  DCL  1  JOB       ENTITY,
             2  SUC       SET,
             2  ALLJOB    MEMBER,
             2  PRED      MEMBER,
             2  ID        ATTRIBUTE  CHAR(20),
             .....
        DCL  X  ENTITY;
             .....
        CREATE   JOB   CALLED X ;
             .....
```

② セットの創成

セットとは、ある同じ条件を満たすエンティティのあつまりである。関連を表現し、それに対してなんらかの処理をする時、ある同じ条件を満たすものに、ある処理を施すという場合が多い。このような時セットは有効である。

前に述べたような、サブセットでアソシエータをリストしたリングもセットである。さらにサブセットをもたずアソシエータだけのリンクで表わされるセットがある。即ちセットには次の二種がある。

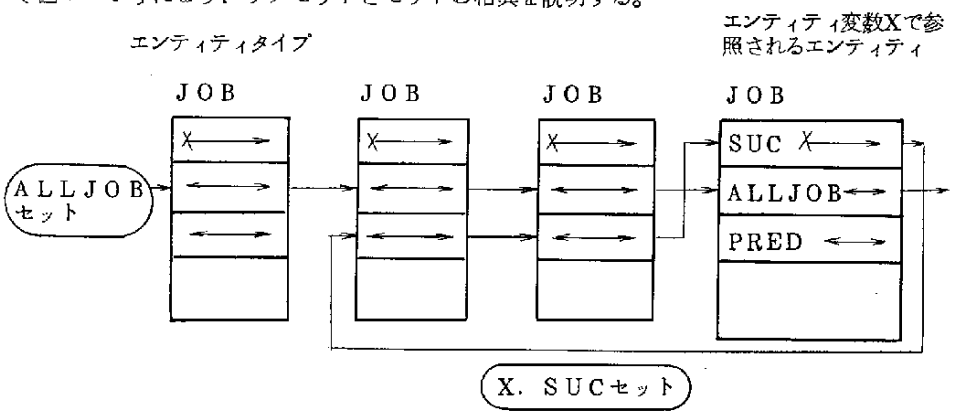
a　サブセット

（サブセットによって作成されるセット）

b　セット

（サブセットとは独立に作成されるセット）

〔図９－５〕により、サブセットとセットの相異を説明する。



〔図９－５〕

X. SUCセットはサブセットSUCにリンクされるアソシエーティブセットの集合を表わすaタイプのセットである。ALLJOBセットはエンティティ・タイプJOBのアソシエーティブセットALLJOBの集合を表わすbタイプのセットである。

aタイプのセットは、エンティティが創成された時に定義され、bタイプのセットは、CREATEステートメントで定義される。共に創成時は空セットである。

次にbタイプのセットがCREATEステートメントで創成される例を示す。

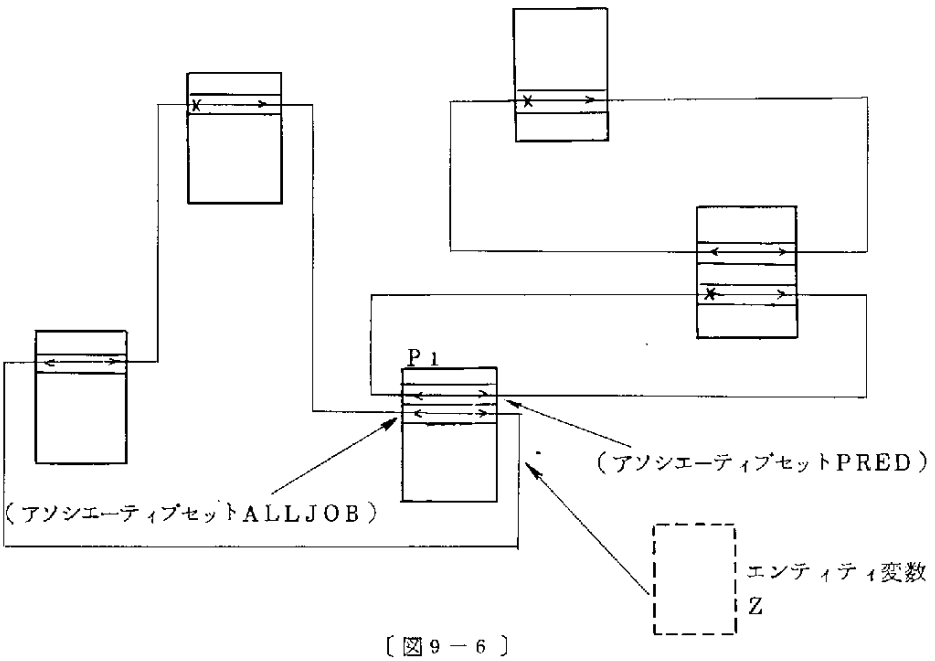
CREATE　セット・ネーム；

（例）　CREATE　ALLJOB

セットALLJOBが創成される。

(3)　エンティティを参照するための機能

前にも述べたように、エンティティを参照するためにはそのエンティティに適当なエンティティ変数を対応ずける（〔図９－６〕参照）。それによってエンティティの要素は次のように参照することができる。



〔図9－6〕

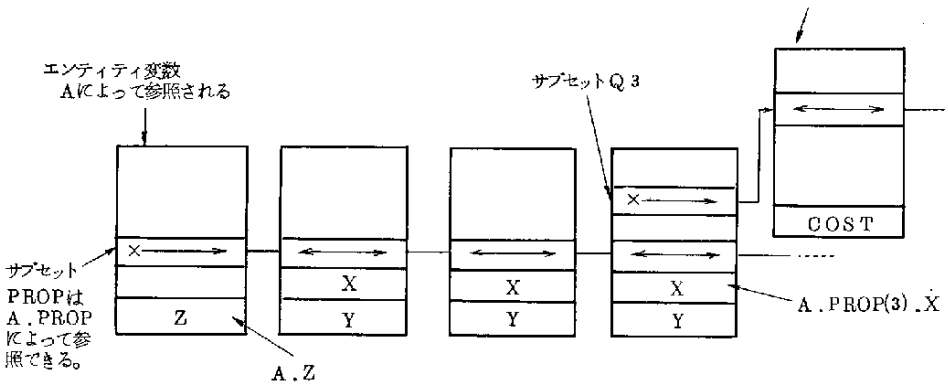
例えばＰ１がエンティティ変数Ｚで参照される時、アソシエーティブセットＰＲＥＤ，ＡＬＬＪＯＢはそれぞれＺ．ＰＲＥＤ，Ｚ．ＡＬＬＪＯＢで参照される。一般にエンティティのセットを参照するためには次のような形がとられる。

＜エンティティ変数＞・＜セット・ネーム＞

又、ダイレクト・アトリビュートの値も同様に参照できる。

またＡＰＬでは、データを多次元に参照できる機能がある。

例えば次のようなリングを考える。



〔図9－7〕

ここでAはサブセットＰＲＯＰを持ち、サブセットＰＲＯＰの三番目のエンティティはＸという値を持っている。このＸを参照するためにはＡ．ＰＲＯＰ（３）．Ｘという表現をする。又エンティティＡのＺの値はＡ．Ｚで参照できる。さらにわかりやすくするために例をあげる。サブセットＰＲＯＰの初めの三つのエンティティのＸを加え、最初のエンティティのＹにその値を代入するには次のようなステートメントで表現される。

Ａ．ＰＲＯＰ（１）．Ｙ＝Ａ．ＰＲＯＰ（１）．Ｘ＋Ａ．ＰＲＯＰ（２）．Ｘ＋Ａ．ＰＲＯＰ（３）．Ｘ

又エンティティＢのＣＯＳＴは

Ａ．ＰＲＯＰ（３）．Ｑ３（１）．ＣＯＳＴ

と表現され、何次元にでも参照可能である。

さらにセットの最終エンティティからの参照もできる。

サブセットＰＲＯＰの最終エンティティのＸの値を参照するには次のように表現できる。

Ａ．ＰＲＯＰ（－１）．Ｘ

アソシエーティブ・アトリビュートの参照は次のような形をとる。

〔エンティティー１，エンティティー２〕．アトリビュート

又は

〔エンティティ，セット〕・アトリビュート

例えばエンティティＥ１とＥ２間のアトリビュートＤＩＳＴＡＮＣＥがあればその値は次のように表現される。

〔E1，E2〕.DISTANCE＝5；

(4) データ・ストラクチャ操作

データ・ストラクチャ操作のためのステートメントは、PL／Iステートメントと並行してプログラム中で使用され、その主な機能としては次のようなものがある。

① エンティティ・セットの創成（(2)参照）

② セットのメンバー変更

③ エンティティ・セットの消去

④ 検　索

これらのステートメントの詳細を次に述べる。説明でアンダ・ラインの引いてあるものはキーワードである。

① セットのメンバー変更

- INSERT エンティティ変数 IN セット

指定されたセットに指定されたエンティティをメンバーとして追加する。

（例） INSERT POINT1 IN ELEMENT

セットELEMENTにエンティティPOINT1が追加される。

- REMOVE エンティティ変数 FROM セット；

  REMOVE エンティティ変数 FROM ALL；

指定されたエンティティを指定されたセット、あるいに全てのセットから取り除く。

（例） REMOVE POINT1 FROM ALL

全てのセットからエンティティPOINT1を取り除く。

② エンティティ・セットの消去

- DELETE {エンティティ変数 / セット}

ファイルから指定されたエンティティを消去し、そのスペースは、フリーストレッジに返す。セットを指定した時は、そのセットのメンバであるすべてのエンティティを消去する。

（例） DELETE POINT1

エンティティPOINT1を消去する。

③ 検　　索

●FIND　エンティティ変数＝エンティティ・スペック

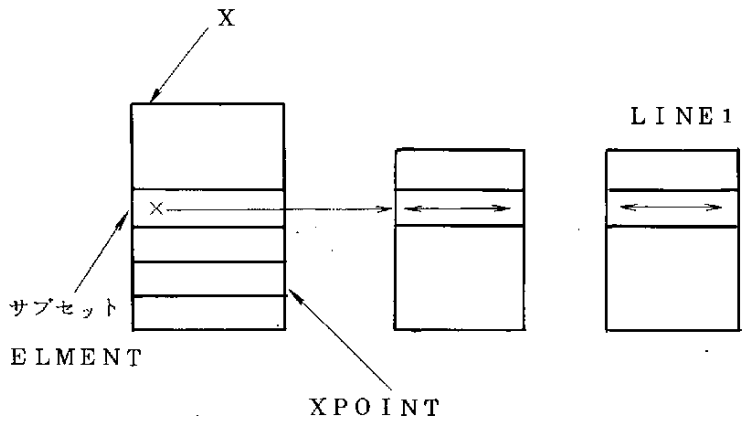
〔，WITHβ〕〔，UNTILβ〕〔，ELSE．ステートメント〕；

あるエンティティ集合の中から条件を満すエンティティを探し、処理を施す。

初めに指定されたエンティティを含むサブセットを持つエンティティを検出する例を示す。

FIND　X＝ELMENT　　LINE1；

とした時、エンティティLINE1を含むようなサブセットELMENTをもつようなエンティティを検出する。（〔図9－8〕参照）



〔図9－8〕

ここではPL／Iの60文字セットの場合に用いられ、CONTAINTSとして使われている。

さらに　FIND　X＝ELMENT　　LINE1

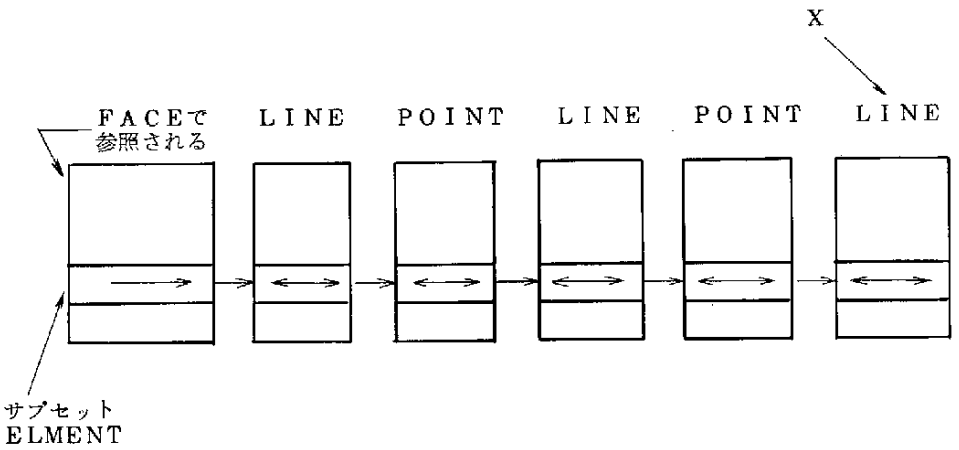
WITH　X.XPOINT＝3；

とした時、前の例の条件を満たすエンティティのうちさらにXPOINTの値が3である条件を満たすエンティティを検出する。UNTILは同様に終了条件をつけ加える時に用いる。

次にセットのメンバーであるエンティティを検出する例を〔図9－9〕に示す。

FIND　X＝(3)LINE　　FACE.ELMENT

FACEのELMENTセットに含まれる三番目のエンティティ・タイプLINEのエンティティを検出する。



〔図9－9〕

(3) ＬＩＮＥはリングを順次検索していく時にエンティティ・タイプＬＩＮＥで作られているエンティティのみに関して三番目のエンティティを表現している。

ＦＩＮＤステートメントにおいては、検索するエンティティは一意的に決まるが、次に述べるＦＯＲ　ＥＡＣＨステートメントはある条件を満足するエンティティを順次検索し、処理する機能をもつ。

○ FOR EACH エンティティ変数＝エンティティ．スペック
〔，WITHβ〕〔，UNTILβ〕；
END；

ＤＯループの中でＦＩＮＤステートメントを実行させる機能をもつ。ＷＩＴＨ，ＵＮＴＩＬもＦＩＮＤステートメントと同様の意味をもつ。

（例）

```
FOR EACH   Z=POINT IN ELMENT,
                        WITH   Z.X=0;
DELETE Z;
END;
```

セットＥＬＭＥＮＴの中のエンティティ・タイプＰＯＩＮＴのエンティティのうちＸの値としてＯを持つエンティティを順次検索し消去する。

ページング機能

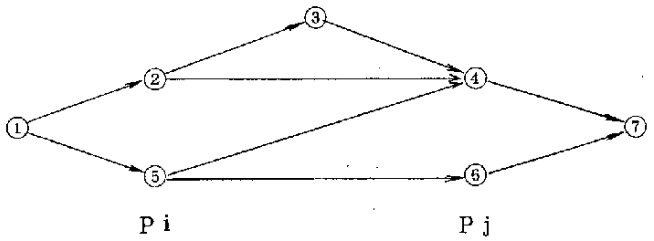
ＡＰＬは、システムが、ソフトウエア．ページング．スーパーバイザを持っている。プログラムのイニシャライズ時にバッファがコア内におかれる。その後要求に応じて必要とされるページが二次記憶からロードされる。エンティティはアドレスとして、エンティティが存在するページとページ内でのフィジカルなロケーションをもち、データ・ページに格納されている。

もし参照するエンティティ、あるいはセットがコアのページ内になければ二次記憶からロードする形をとっている。

又システムはコアメモリで領域が十分である時にはｎｏｎ−ｐａｇｅ−ｍｏｄｅで操作することも可能にしている。この時エンティティのアドレスはアブソリュートに表現される。

以下にこれらのストラクチャ操作ステートメントを使ってネットワークを作成する例を示す。

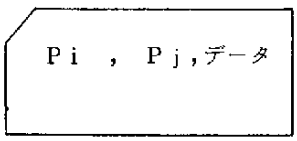
（例）　ＡＰＬによる例



図のネットワークを表現することを考える。

＊　ノードに対して１エレメントを割り当てる。アクティヴイティーはエレメントのつながりとして表現する。

＊　ネットワークデータは外部から次の形で与える。



〔プログラム例〕

```
DECLARE  1  NODE  ENTITY,
         2  PRED  SET    ,
         2  PR    MEMBER,
         2  ALLJOBR MEMBER,
         2  ID    FIXED  ;
DECLARE  ALL NODE SET,PI ENTITY VAR,PJ
         ENTITY VAR;
READ:    GET LIST(KPI,KPJ);
         FIND PI=(1)NODE IN ALLNODE,WITH
         PI.ID=KPI,
         ELSE CALL DEFINE(PI,KPI);
         FIND PJ=(1)NODE IN ALLNODE,WITH
         PI.ID=KPJ,
         ELSE CALL DEFINE(PI,KPJ);
         INSERT PJ IN PI.PRED;
         GO TO READ;
DEFIND:  PROCEDURE (ARG1,ARG2);
         DECLARE ARG1 ENTITY VAR,ARG2
         FIXED;
         CREATE NODE CALLED ARG1;
         ARG1.ID=ARG2;
         INSERT ARG1 IN ALLJOB;
         RETURN;
         END DEFINE;
```

4—10 ASP(Associative Structure Package)

概　　要

ＡＳＰは、ケンブリッジ大学の数学研究所、工学研究所のメンバーによって開発されたもので、ＴＩＴＡＮ（ＡＴＬＡＳⅡ、６４KW、４８ｂｉｔ／Ｗ）のもとにインプリメントされ、ＡＴＬＡＳⅡのMixed Language System（ＭＩＳ）に組み込まれている。

ＡＳＰステートメントは汎用マクロ・ジェネレータML／Ｉ（マクロ・コールを展開して、アセンブラ・コードをジェネレートする）のもとで定義されたマクロステートメントの集まりであり、ＭＬＳに組み込まれている異ったランゲッジで書かれたプログラムと共通フォーマットにコンパイルされ、いっしょにロードすることができる。例えばフォートランからＣＡＬＬすることもできるわけである。

これまでのＣＯＲＡＬ、ＡＰＬにおいてはリンク領域をテーブル形式で表現していたため、実行時に新たな参照関係が生じた時には、改めてブロック全体を作り直さなければならず効率がよくなかった。

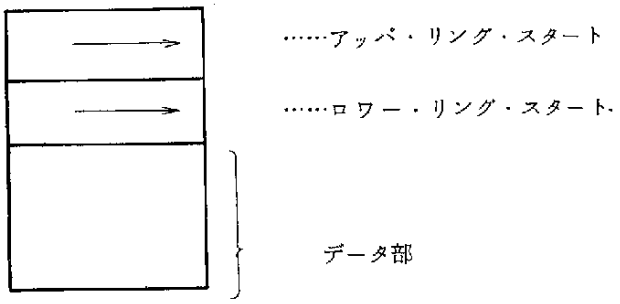
ＡＳＰでは、リンク部をリスト形式にすることにより、リング・スタート、リング・タイの動的な拡張を可能にしている。この様に、ＡＳＰはリング・ストラクチャの中では、その表現力に於て最も柔軟性のある構造を備えていると云えるだろう。しかし、リンク部はこれまでのフォワード・ポインタ、バックワード・ポインタに加えて、そのリングが属しているブロックへのポインタを持たなければならないので、リンク部に費やされる領域は非常に大きなものになっている。（ＡＰＬと比較した場合約２倍）

データ構造

ＡＳＰの扱う、データ・ストラクチャは、エレメント、リング・スタート、アソシエータと呼ばれる三つの基本要素から構成される。以下ではこれらの基本要素について説明する。

(1)　エレメント

エレメントはＣＯＲＡＬのブロック、ＡＰＬのエンティティに対応するもので〔図10—1〕のような内部構造をもつ。ＡＳＰではエレメントはリンク部を内部にもたず、リンク部を管理するリングのリング・スタートをもつ。それらはアッパ・リング・スタート、ロワー・リング・スタートと呼ばれる。



〔エレメントの内部構造〕
〔図１０－１〕

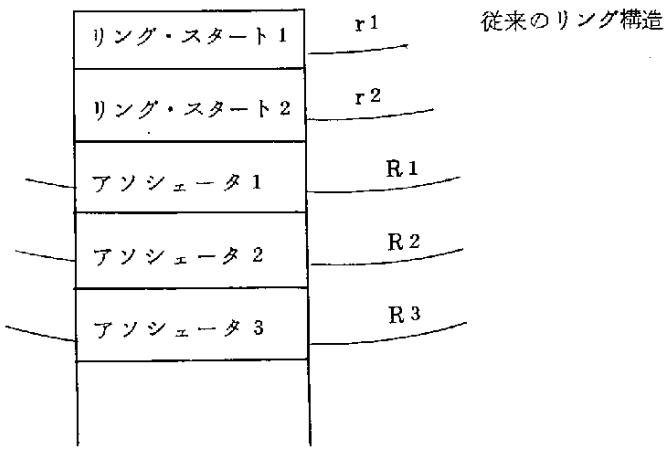
アッパ・リング・スタート；

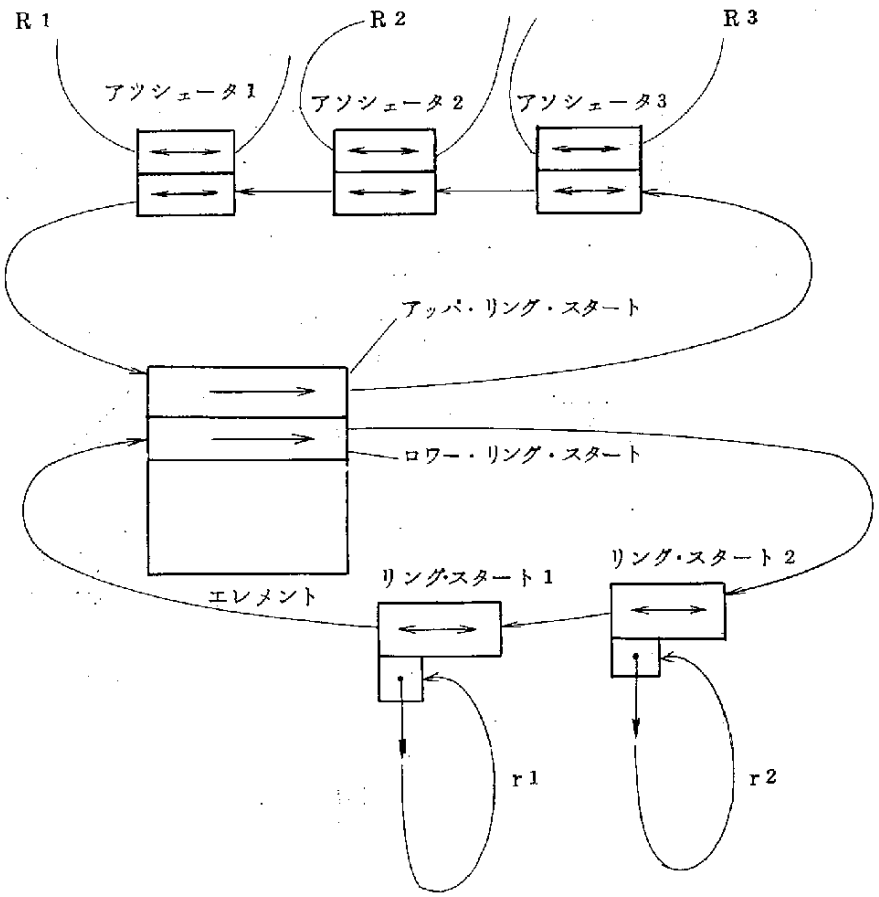
このエレメントがもつリング・スタートの集合を管理するリングのリング・スタートである。

ロワー・リング・スタート；

このエレメントがもつアソシェータの集合を管理するリングのリング・スタート

リング・スタート、アソシェータについては後で述べるが、アッパ・リング・スタートとロワー・リング・スタートの関連を〔１０－２〕に示す。





〔図10－2〕　ASPに於けるリング構造

(2)　リング・スタート

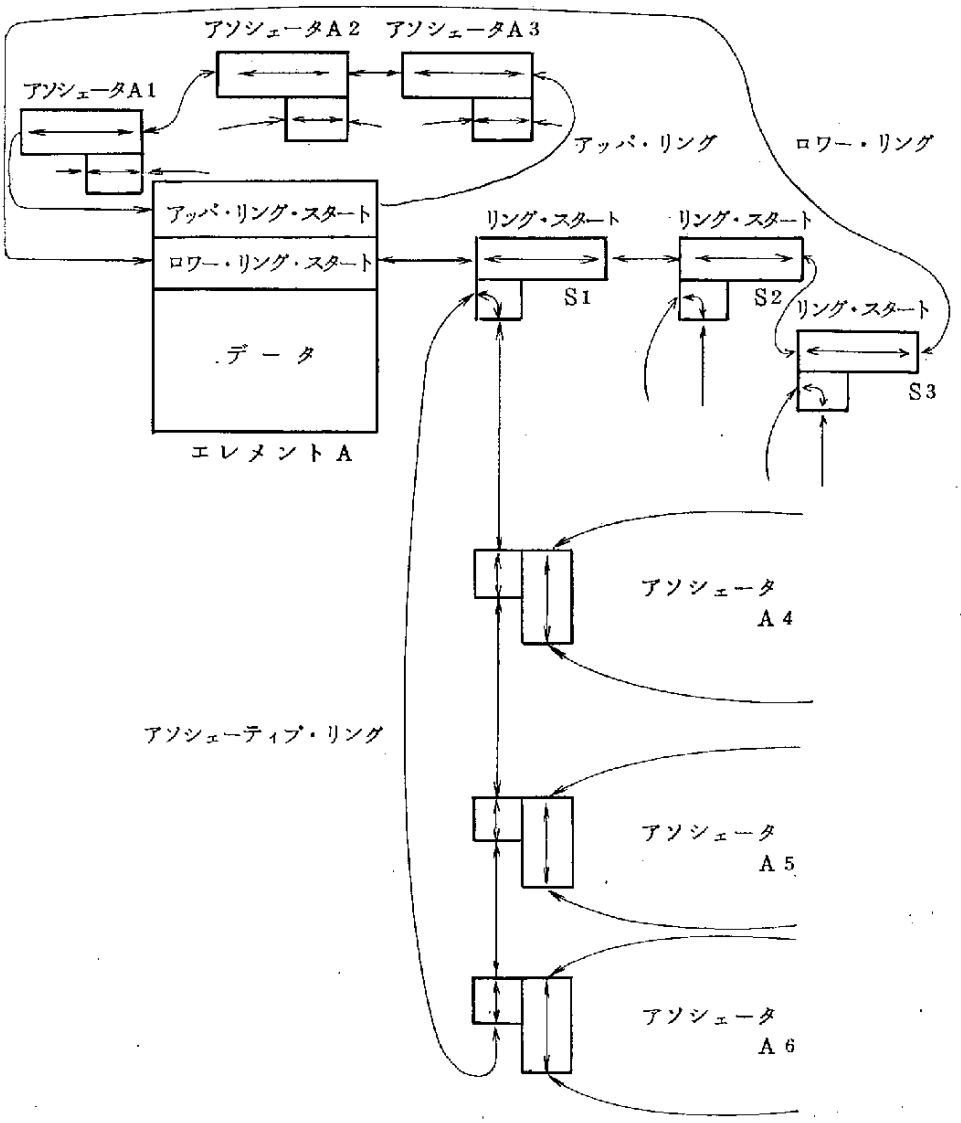
リング・スタートはAPLのサブセットに対応するものでリングのネームとタイプを持つことができる。あるエレメントに属するリング・スタートは、そのエレメントのロワー・リング・スタートからでているロワー・リング上にリストされる。

(3)　アソシェータ

アソシェータはAPLのアソシェーティブ・リングに対応するもので、アソシェーションと呼ばれる値を持つことが出来る。エレメントに属するアソシェータは全て、そのエレメント

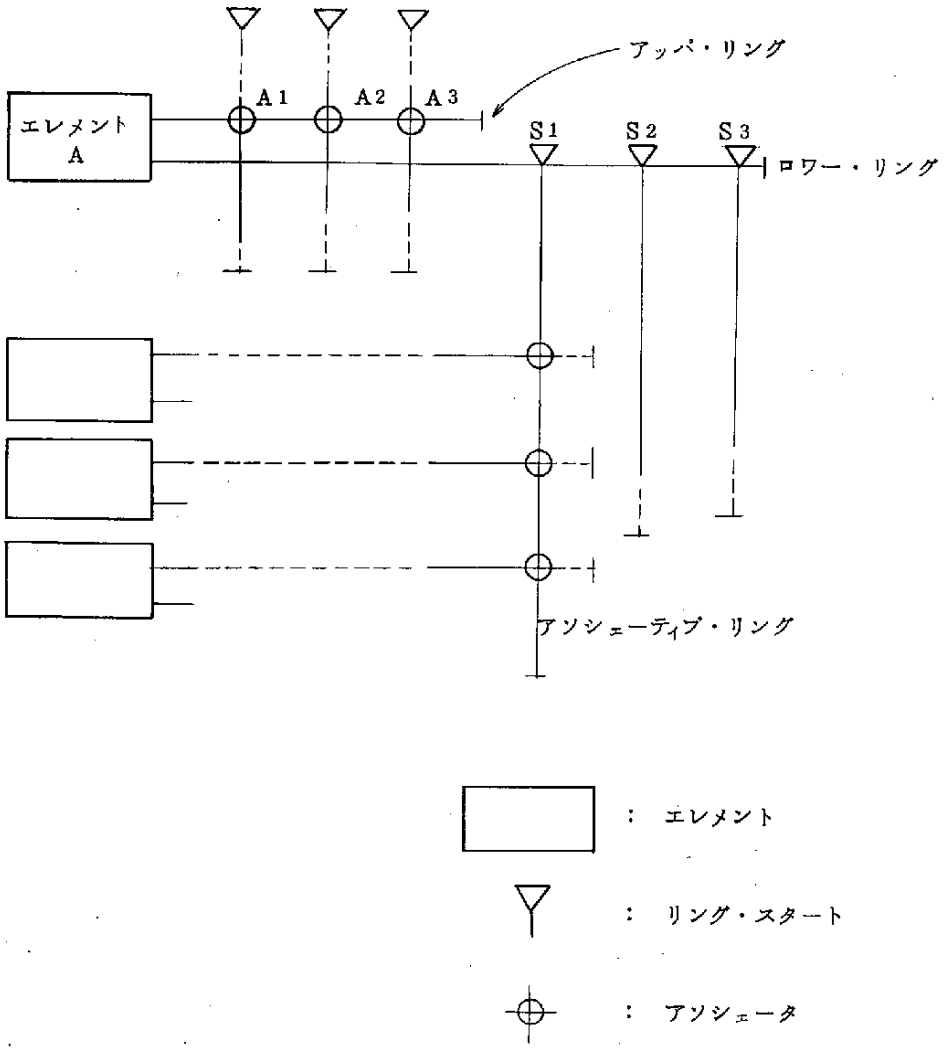
のアッパ・リング・スタートからでているアッパ・リングでリストされる。

各リング・スタートは、それぞれアソシェーティブ・リングを持つことができ、メンバとするエレメントのアソシェータをこのアソシェーティブ・リングでリンクする。即ち、アソシェータはアッパ・リングとアソシェーティブ・リングに同時にリンクされることになる。これらのリングの関連を〔図１０－３〕で示す。



〔ＡＳＰストラクチャによる表現〕
〔図１０－３〕

ストラクチャを理解しやすい図で表わす記法にＡＳＰノーティションがある。前の図を、ＡＳＰノーティションで表わすと〔図１０－４〕のようになる。



〔ＡＳＰ　ノーティションによる表現〕

〔図１０－４〕

ステートメント機能

ＡＳＰステートメントは、マクロ・コールの形式をとり、既存の言語から呼び出して利用することも出来る。

ＡＳＰマクロ・ステートメントは機能上次の４つに分類される。

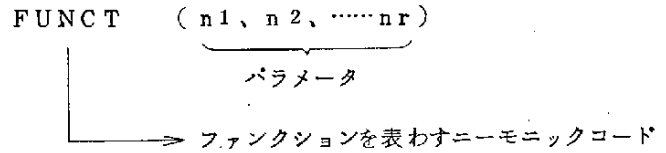
(1) セルの創成、挿入

(2) セルの消去

(3) 検　索

(4) デバッキング

ＡＳＰのストラクチャ操作は、構造要素のフィジカル・アドレスを直接意識するようなポインタ操作であり、ステートメントはポインタをパラメータとしてもち、一般には次のような形式で記述される。



ここでパラメータとしては次のものが与えられる。

Ｅ ： エレメントのアドレス

Ａ ： アソシェータのアドレス

Ｓ ： リング・スタートのアドレス

Ａ ： アソシェーション　（アソシェータがもつ値）

Ｎ ： リングネーム　　（リング・スタートがもつ）

Ｔ ： リングタイプ　　（リング・スタートがもつ）

Ｌ ： ユーザが定義したレベル

一般に、マクロ・コールの結果、値（Ｖ）かポインタ（Ｐ）が返されるが、エレメント、リング・スタート、アソシェータなどの消去の時は、何も返ってこない。

次にこれらのパラメータを使用して機能別に、各ステートメントの説明をする。ここで"→Ｐ"、"→Ｖ"はマクロ・コールの結果として、それぞれポインタ、値が返されることを示す。

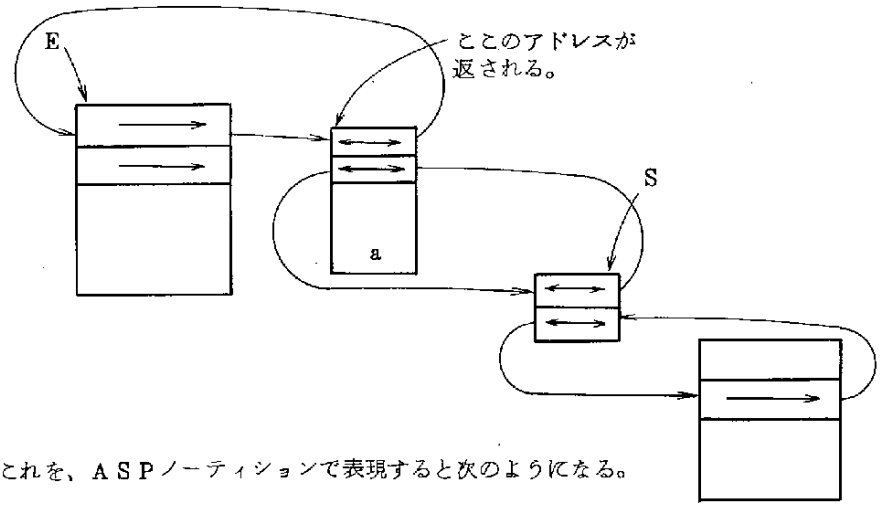
(1) セルの創成、挿入

① ＥＣ（Ｌ）→Ｐ

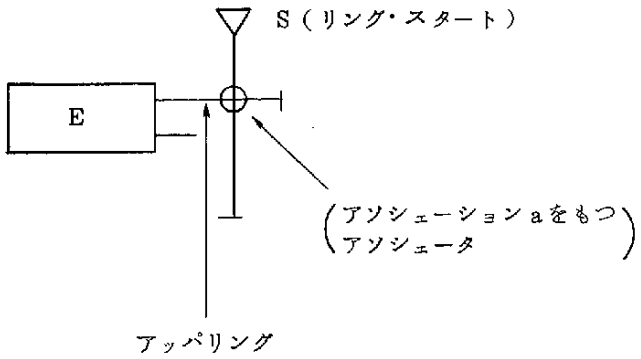
データ長Lのエレメントが創成され、アロケートされた先頭アドレスが返される。

② ＥＩＮ（Ｅ、Ｓ、ａ）→Ｐ

エレメントＥをアソシェーティブ・リングＳにのせ、その時のアソシェーションをａとする。

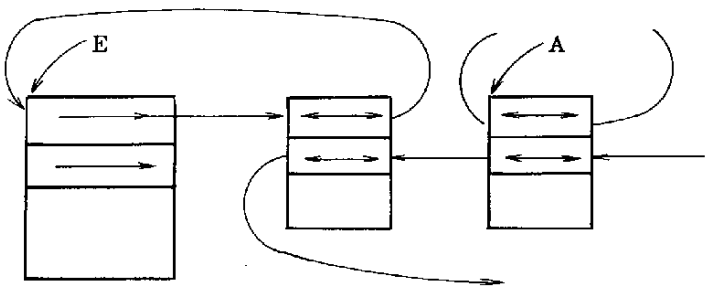


これを、ＡＳＰノーティションで表現すると次のようになる。



③ ＥＩＮＡ（Ａ、Ｅ、ａ）→Ｐ

エレメントＥをアソシェタＡの後に挿入し、アソシェーションをａとする。



④ ＥＩＮＢ（Ａ、Ｅ、ａ）→Ｐ

エレメントＥをアソシェータＡの前に挿入し、アソシェーションをａとする。

⑤ ＣＳＥ（Ｅ、ｎ、ｔ）→Ｐ

エレメントＥのロワーリングに、ネームｎ、タイプｔのリング・スタートを創成する。

⑥ ＣＳＡ（Ｓ、ｎ、ｔ）→Ｐ

リング・スタートＳの後にネームｎ、タイプｔのリング・スタートを創成する。

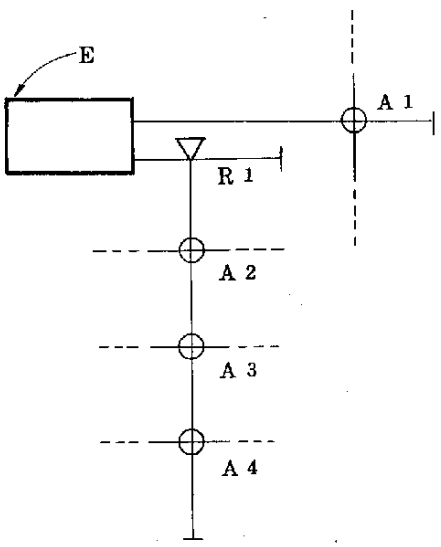
⑦ ＣＳＢ（Ｓ、ｎ、ｔ）→Ｐ

リング・スタートＳの前にネームｎ、タイプｔのリング・スタートを創成する。

(2) セルの消去

① ＥＤＹ（Ｅ）

エレメントＥを消去する。と同時にそのエレメントにのみ関連をもつ、アソシェータ、リング・スタートは消去される。例えば次のように構造化されていたとする。



エレメントＥが消去されると、エレメントＥはもちろんのことリング・スタートＲ１、アソシェータ、Ａ１、Ａ２、Ａ３、Ａ４も全て消去される。

② ＡＤＹ（Ａ）

アソシェータを消去する。

③ ＬＲＤＹ（Ｅ）

エレメントＥのロワー・リング上の全てのリング・スタートを消去する。と同時に、関連の

意味がなくなった、アソシェータ、エレメントも消去される。

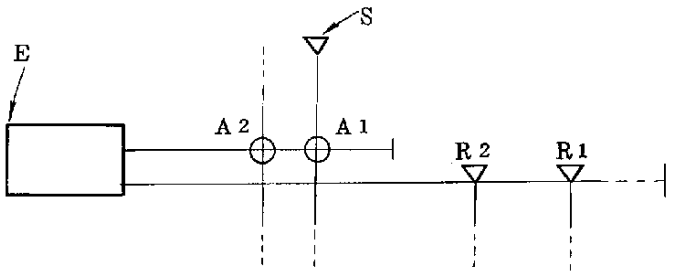
④　ＳＣＬ（Ｓ）

リングＳをクリアする。即ち、リング・スタートＳからのアソシェーティブ・リング上のアソシェータを全て消去する。

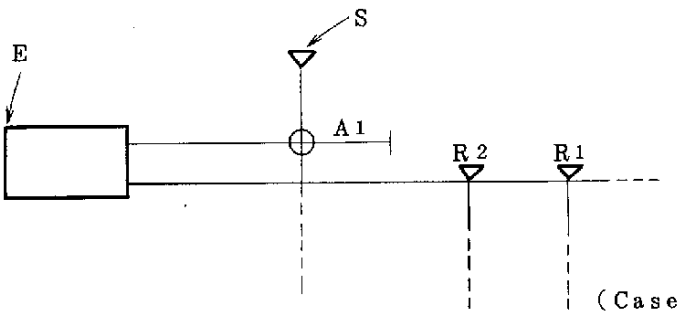
⑤　ＳＤＹ（Ｓ）

リング・スタートを消去する。と同時に関連の意味がなくなったアソシェータ、エレメントも消去される。

例えば次の２つの例で考えてみる。



（Case1）



（Case2）

リング・スタートＳを消去すると

Ｃａｓｅ1の場合は、アソシェータＡ１が消去される。

Ｃａｓｅ2の場合は、アソシェータＡ１、リング・スタートＲ１、Ｒ２さらにエレメントＥまで消去される。

(3) 検　索

ＡＰＬの検索ステートメントでは、ある条件を与えることによって、その条件を満た

す要素を検索できた。ところがＡＳＰステートメントを用いる場合には、ある条件を満たす要素はどのように関連づけられているかを解析し、その結果、ＡＳＰステートメントによって、その要素がアロケートされている位置を知る。すなわちＡＳＰステートメントの検索機能は基本的なものだけであり、高度な検索が必要な時は、ユーザが目的に応じたルーチンをＡＳＰステートメントを用いて作成する。

① ＵＲＦ（Ａ／Ｅ、Ｌ）→Ｐ

アッパ・リング上でアソシェータＡの次のアソシェータを検索する。エレメントＥを指定した時は、アッパ・リング上の最初のアソシェータを検索することになる。もし、次のアソシェータがない時は、レベルＬにいく。

② ＵＲＢ（Ａ、Ｌ）→Ｐ

アッパ・リング上で前のアソシェータを検索する。

③ ＵＲＮ（Ｎ、Ｅ、Ｌ）→Ｐ

エレメントＥのアッパ・リング上のＮ番目のアソシェータを検索する。

④ ＡＥＬ（Ａ）→Ｐ

アソシェータＡのエレメントを検索する。

⑤ ＡＳＴ（Ａ）→Ｐ

アソシェータＡのリング・スタートを検索する。

⑥ ＵＲＬ（Ｅ）→Ｖ

エレメントＥのアッパ・リング上のアソシェータの数を調べる。

ロワー・リング、アソシェーティブ・リングに関しても①～⑥の機能をもつステートメントがある。

⑦ ＡＰＲ（Ａ）→Ｖ

アソシェーティブ・リング上のアソシェータＡのアドレスを調べる。

⑧ ＥＬＧＨ（Ｅ）→Ｖ

エレメントＥのブロック長を調べる。

⑨ ＡＳＳＮ（Ａ）→Ｖ

アソシェータＡのアソシェーションを調べる。

⑩ ＳＮＡＭ（Ｓ）→Ｖ

リング・スタートＳのネームを調べる。

⑪ ＥＣＹ（Ｅ）→Ｐ

エレメントEをコピーする。作られたエレメントのアドレスが返される。

(4) デバッギング

ＡＳＰはデバッギング機能をもっていて、任意のエレメント、アッパ・リング、ロワー・リングの内容、あるいは、アソシェーティブ・リング上のデータを持つエレメント、アソシェータなどをプリントアウトすることが可能である。

① ＰＲＵＲ（Ｅ）

エレメントEのアッパ・リング上の全てのアソシェータをプリントする。

② ＰＲＬＲ（Ｅ）

エレメントEのロワー・リング上の全てのリング・スタートをプリントする。

③ ＰＲＡＲ（Ｓ）

アソシェーティブ・リングＳ上の全てのリング・スタートをプリントする。

④ ＰＲＥＲ（Ｓ）

アソシェーティブ・リングＳ上にアソシェータをもつ全てのエレメントをプリントする。

⑤ ＰＲＡ（Ａ）

アソシェタAをプリントする。

⑥ ＰＲＥ（Ｅ）

エレメントEをプリントする。

⑦ ＰＲＳ（Ｓ）

リング・スタートSをプリントする。

次にＡＰＬを用いてネットワークを記述する例を、ＡＳＰをフォートランから呼ぶ形で、プログラミングした例を示す。

〔プログラム例〕

```
----------------------
1000  READ(5、1100)KPI, KPT
C       ***  STEP1  ***
        ALNODE=EC(LA)
        ALNODS=CSE(ALNODE, TYPEA)
C       ***  STEP2  ***
        ASOCAT=ALNODS
```

```
      DO 100 I=1, 1000
      ASOCAT=ARF(ASOCAT, LC)
      IF(LC. EQ. 1) GO TO 200
      NODEJ=AEL(ASOCAT)
      IF(AREA(NODEJ+ID). EQ. KPJ) GO TO 300
100   CONTINUE
C     *** STEP3 ***
200   CALL DEFINE(KPJ, NODEJ)
C     *** STEP4 ***
300   ASOCAT=ALNODS
      DO 400 I=1, 1000
      ASOCAT=ARF(ASOCAT, LC)
      IF(LC. EQ. 1) GO TO 500
      NODEI=AEL(ASOCAT)
      IF(AREA(NODEI+ID). EQ. KPI) GO TO 600
400   CONTINUE
C     *** STEP5 ***
500   CALL DEFINE(KPI, NODEI)
      PREDA=CSE(NODEI, PRED, TYPEN)
      ASOCAT=EIN(NODEJ, PREDA, ASOC)
      GO TO 1000
C     *** STEP6 ***
600   PREDA=LRF(NODEI, LC)
      ASOCAT=PREDA
700   ASOCAT=ARF(ASOCAT, LC)
      IF(LC. NE. 1) GO TO 700
      ASOCAT=EIN(NODEJ, ASOCAT, ASOC)
      GO TO 1000
      STOP
      END
```

```
      SUBROUTINE  DEFINE  (KPI, NODEA)
                  ----------
      NODEA=EC(LN)
      AREA(NODEA+ID)=KPI
100   ASOCAT=ARF(ALNODS, L)
      IF(L. NE. 1)  GO  TO  100
      ASOCAT=EINA(ASOCAT, NODEA, TYPEN)
      RETURN
      END
```

次に各ステップの動作を簡単に説明する。

STEP1：全てのNODEエレメントを管理するALNODEエレメントの創成と、リング・スタートALNODSの創成

STEP2：NODE　KPJが既にALNODSのアソシェーティブ・リング上にあるかどうかを調べる。

STEP3：リング上になかった時は、NODE　KPJを創成する。

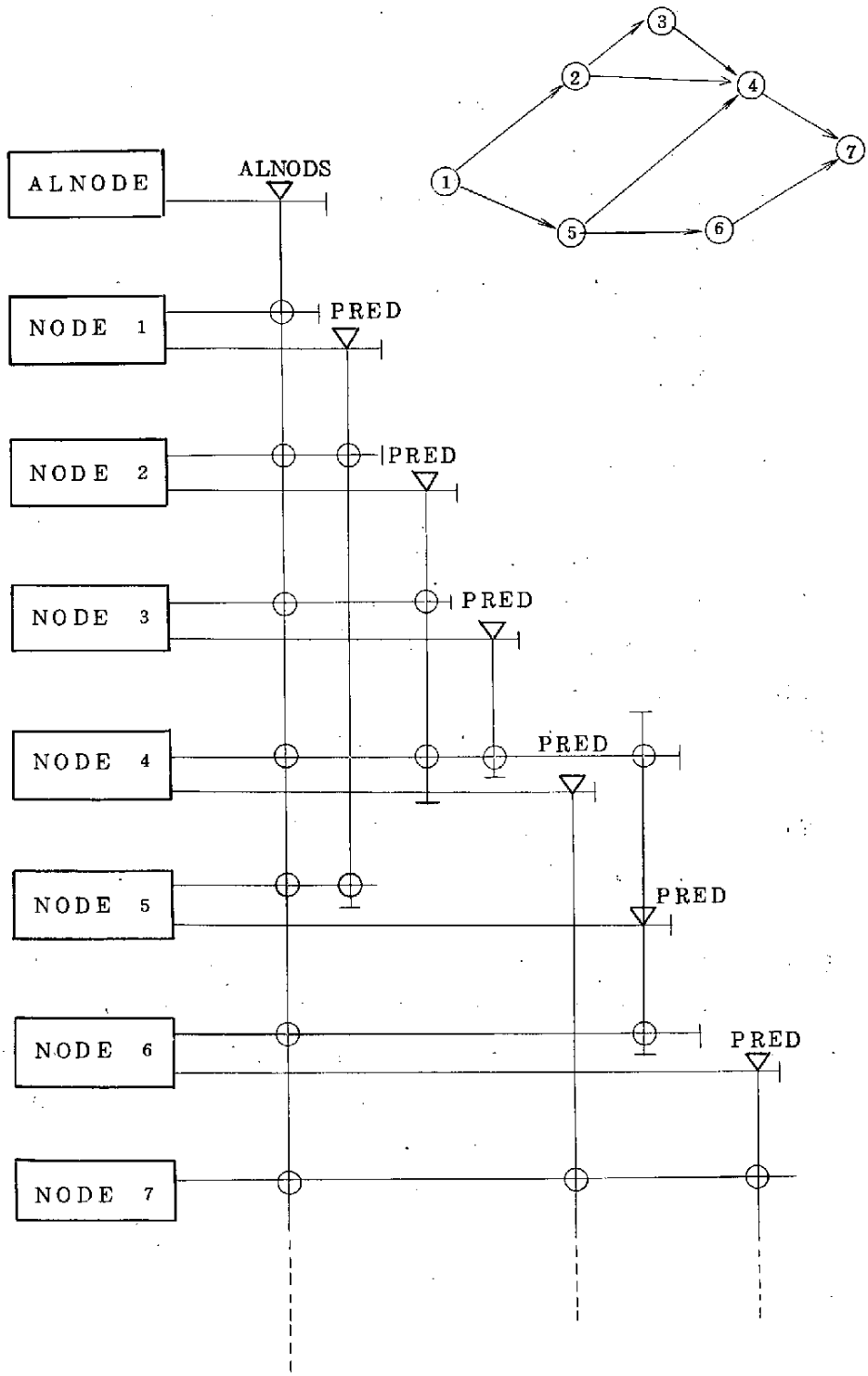
STEP4：NODE　KPIが既にALNODSのアソシェーティブ・リング上にあるかどうかを調べる。

STEP5：リング上になかった時は、NODE　KPIを創成し、さらにリング・スタートPREDを創成し、そのアソシェーティブ・リングにNODE　KPJを挿入入する。

STEP6：リング上にあった時は、NODE　KPIのリング・スタートPREDを検索し、そのアソシェーティブ・リングにNODE　KPJを挿入する。

サブルーチンDEFINE：NODE エレメントを創成し、識別名を、エレメントの先頭アドレス＋IDの場所に格納する。そして、リング・スタートALNODSのアソシェーティブ・リングの最後に挿入する。

このプログラムによってデータ・ストラクチャは次のような形となる。


ALNODE
ALNODS
NODE 1
PRED
NODE 2
PRED
NODE 3
PRED
NODE 4
PRED
NODE 5
PRED
NODE 6
PRED
NODE 7
1
2
3
4
5
6
7

4—11 LEAP

概　　要

ＬＥＡＰは１９６７年Ｌｉｎｃｏｌｎ Ｌａｂｏｒａｔｏｒｙで開発されたシステムで、ＴＸ－２のもとにインプリメントされ、そのコンパイラ作成にはＴＸ－２のコンパイラ・ジェネレータ・システムＶＩＴＡＬが使用されたという。

これまでのリング・ストラクチャでは関連表現をするために各アイテム間をリングリストでいかに効率よく結ぶかということに重点がおかれていて、関連記述はストラクチャを意識しなければならなかった。しかしＬＥＡＰでは〝いかに構造化するか〟というよりむしろ〝いかに表現するか〟という立場から内部構造を意識せずに関連記述ができるような形式がとられている。

関連はオーダード・トリプルと呼ばれるＡ・Ｏ≡Ｖの形式で表現される。例えば、

ＳＴＡＲＴＰＯＩＮＴ・ＬＩＮＥ１≡ＰＯＩＮＴ１

このトリプルはＬＩＮＥ１の始点がＰＯＩＮＴ１であるという関連を表現している。

そしてこのトリプルは順序づけされた３つ組要素（オーダード・トリプル）として二次記憶上の連想記憶に格納される単位でもある。連想記憶にはＡスペース、Ｏスペース、Ｖスペースと呼ばれる三つのスペースがあり、トリプルは構成要素のうちの二つのアイテムにダブルハッシングを施すことにより、各スペースに異った形式で格納される。このことはトリプルを直接的にあらゆる形式（ＳＡＦ）で検索できることを意味している。

このように二次記憶への拡張性、記述性、処理速度の点からみて、ＬＥＡＰはアソシエティブ言語の中では最も高度な部類に属すると云えるだろう。

(1) ストラクチャの基本要素

ＬＥＡＰにおける関連表現の基本的な基素はトリプルと呼ばれる次のような形式で定義される。

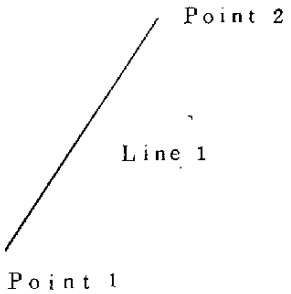
Ａｔｔｒｉｂｕｔｅ ・ Ｏｂｊｅｃｔ ≡ Ｖａｌｕｅ

Ａ ・ Ｏ ≡ Ｖ

このトリプルは関連表現の基本的な単位であると同時に、データ・ストラクチャ内に格納される単位でもある。トリプルを構成する３つの項はアイテムと呼ばれ、そのアイテムは独自のデータを持つことができる。

例えば次の図の線は、ＬＩＮＥ１、ＰＯＩＮＴ２をアイテムとしてＳＴＡＲＴＰＯＩＮＴ

・ＬＩＮＥ１≡ＰＯＩＮＴ１，ＥＮＤＰＯＩＮＴ・ＬＩＮＥ１≡ＰＯＩＮＴ２　のトリプルで表現できる。又；ＬＩＮＥ１は実線であることを示すデータ、ＰＯＩＮＴ１，ＰＯＩＮＴ２は各点の座標値をデータとして持てる。



トリプルは連想記憶と呼ばれる領域に次のような方法で格納される。トリプルを構成するアイテムに二進数とみなすことができるインターナルネームを対応づける。そしてトリプルを構成するアイテムの中の二つのアイテムのインターナルネームの間に二進演算あるいは論理演算を施して、格納するページとそのページ内の相対アドレスを決定する。この方法はダブルハッシング法と呼ばれている。

連想記憶にはＡスペース、Ｏスペース、Ｖスペースと呼ばれるページ化された３つのスペースがあり、トリプルは上記の方法で、これらのスペースに異った形で三通りに格納される。即ち、Ａスペースには、ＡとＯでハッシングされたアドレスにトリプルを格納する。この時Ａがページを決定し、さらにＯでハッシングを施し相対アドレスを決定するためＡはアクセスワード、Ｏはチェックワードと呼ばれる。他のスペースに関しては次のとおりである。

	アクセスワード	チェックワード
Ａスペース	Ａ	Ｏ
Ｏスペース	Ｏ	Ｖ
Ｖスペース	Ｖ	Ａ

トリプルにはＳＡＦと呼ばれる次のような７つの記述形式がある。

Ｆ１　：　Ａ．Ｏ≡Ｖ
Ｆ２　：　Ａ．？≡Ｖ
Ｆ３　：　Ａ．？≡？
Ｆ４　：　Ａ．Ｏ≡？
Ｆ５　：　？．？≡Ｖ
Ｆ６　：　？．Ｏ≡Ｖ
Ｆ７　：　？．Ｏ≡？

Ｆ１はトリプルを定義するとき、トリプルの存在を確かめるとき、又トリプルを消去する

時に使われる形式である。

Ｆ２～Ｆ７の形式は検索の時に使われる。例えばＦ２のＡ・？＝Ｖを検索した時は、Ａスペース内を検索する。というのは前に述べたように、Ａスペースでは、アクセスワードＡによってハッシングされるため、同じＡをもつトリプルは同じページ内に格納されている。同様に？・Ｏ＝Ｖの質問に対しては、Ｏスペースを検索する。

このようにトリプルの表現とトリプルの格納されるアドレスが関連をもち、トリプルを直接アクセスできるため、ＳＡＦによる検索にも高速応答が期待できる。

さらに、ＬＥＡＰでは関連表現をする要素として、トリプルの他にセットと呼ばれるものがある。セットはある条件を満たすいくつかのアイテムの集合であり、トリプルがＡ，Ｏ，Ｖの関連を定義するようにセットはｎ－ｔｕｐｌｅのアイテム間の関係を定義する。しかしセットの要素となるアイテムは順序性を持たない。

例えば　｛ＰＯＩＮＴ１，ＰＯＩＮＴ２，ＰＯＩＮＴ３，ＰＯＩＮＴ４｝のセットは、点を表わすアイテムの集合である。

(2)　連想記憶

連想記憶は二次記憶上のページ化された領域で、一般に〔図１１－１〕のような構造をもつ。

①　インデックス・テーブルはコア内に常駐し、各テーブルの管理をする。

②　ハッシュ・ディクショナリー

アイテム、セットの識別記号（ユーザが与えたもの）とシステムが与えるインターナルネームの対応をとっている。

③　データ・テーブル

アイテムがもつデータを格納する領域

④　メンバー・テーブル

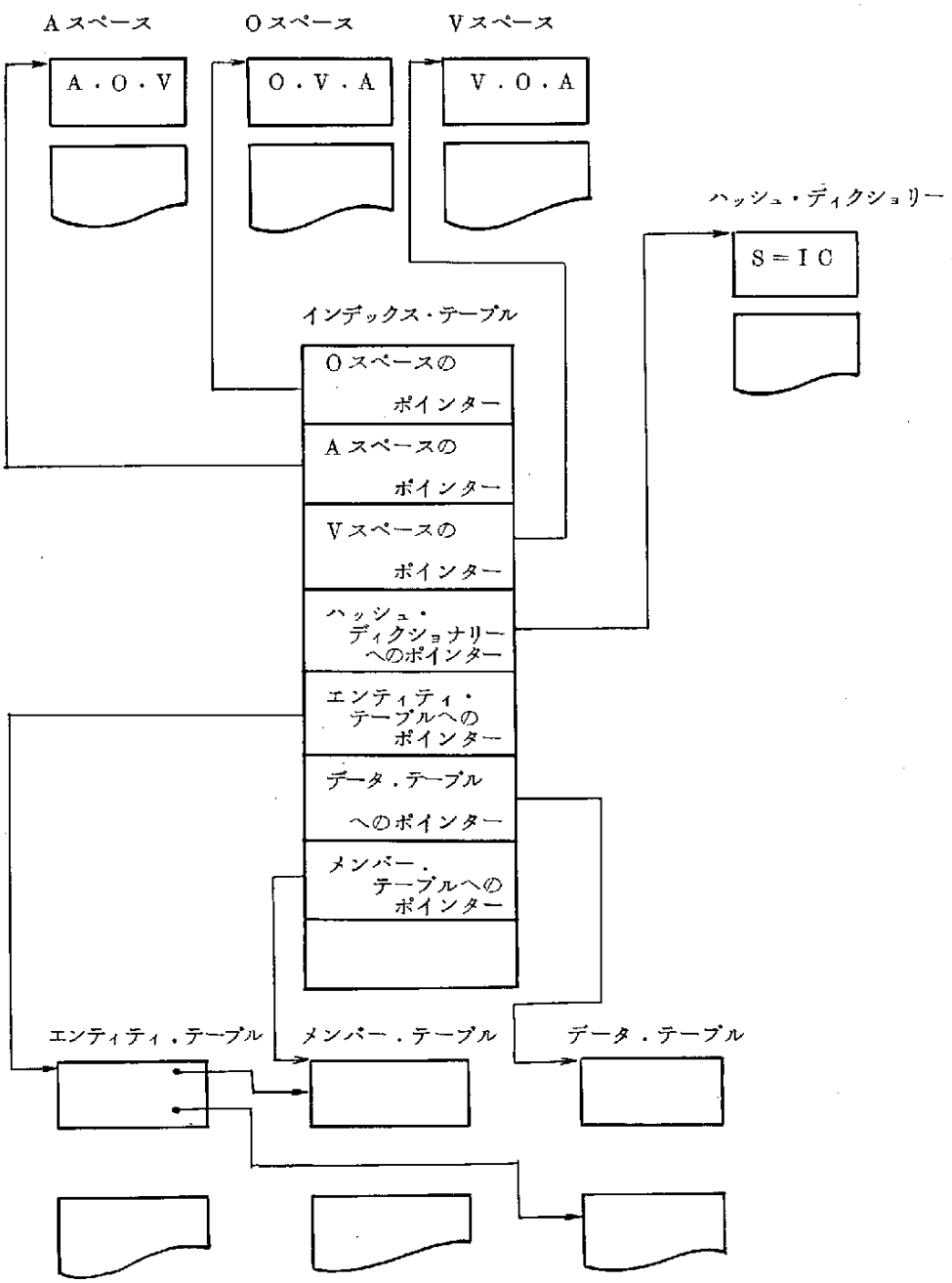
セットのメンバーを表現するための領域

⑤　エンティティ・テーブル

アイテム、セットの特性を格納する。さらにアイテムの場合は、そのアイテムがもつデータが格納されているデータ・テーブルへのポインタをもち、セットの場合はメンバー・テーブルへのポインタをもつ。

⑥　Ａスペース、Ｏスペース、Ｖスペース

連想記憶の最も重要な領域で、トリプルを格納する。前に述べたように、トリプルはアク



〔連想記憶構造のブロック化〕

〔図１１－１〕

セスワードとチェックワードによるダブルハッシング法で決定されたアドレスに格納される。例としてトリプルＳＴＡＲＴ・ＬＩＮＥ１≡POINT1に関して、格納アドレスを求めてみる。ここでＳＴＡＲＴ，ＬＩＮＥ１，ＰＯＩＮＴ１のインターナルネームをそれぞれ１７４５３、２１４１１、１４３７２と仮定する。Ａスペースに対して考えると、まずＳＴＡＲＴのインターナルネーム１７４５３の上二桁をページ表現とみなす。さらに残りの下三桁にＬＩＮＥ１のインターナルネーム２１４１１の上三桁を加えることにより、ページ内のアドレスとする。

ページ　　１７

ページ内の相対アドレス　　４５３＋２１４＝６６７

そこでトリプルＳＴＡＲＴ・ＬＩＮＥ１≡ＰＯＩＮＴ１は、１７ページの６６７番地に格納されるということになる。このダブルハッシング法は、全トリプルに対し、なるべくユニークなアドレスが対応し、かつ小さな領域に格納されるような方法が望ましい。

しかし、実際には$A_1 \cdot O_1 \equiv V_1$、$A_1 \cdot O_2 \equiv V_2$ という二つのトリプルに対し、（$A_1 \cdot O_1$）、（$A_1 \cdot O_2$）でダブルハッシングをした時に、同じページの同じアドレスを指すような結果となることがある。これをコンフリクトと呼ぶ。

又、$A_1 \cdot O_1 \equiv V_1$、$A_1 \cdot O_1 \equiv V_2$ は当然同じアドレスを指す。これはコンフリクトとは区別してマルチプルヒットと呼ぶ。

次に、スペースの中では実際にトリプルがどのような形で表現されるかを、例をあげて説明する。

a：　$A_1 \cdot O_1 \equiv V_1$

b：　$A_1 \cdot O_1 \equiv [A_6 \cdot O_6 \equiv V_2]$　注）

c：　$A_2 \cdot O_2 \equiv V_3$

d：　$A_2 \cdot O_2 \equiv V_4$

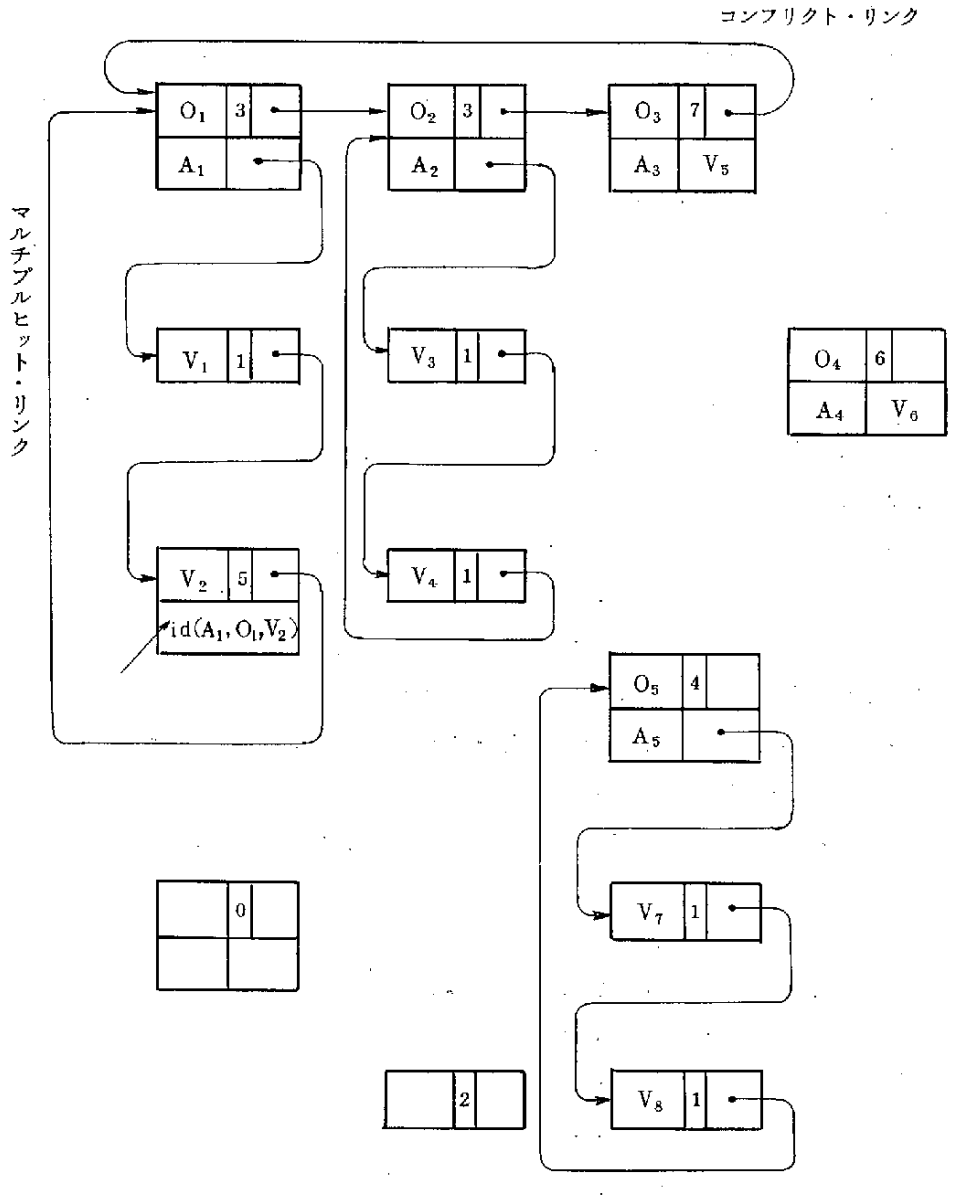
e：　$A_3 \cdot O_3 \equiv V_5$

f：　$A_4 \cdot O_4 \equiv V_6$

g：　$A_5 \cdot O_5 \equiv V_7$

h：　$A_5 \cdot O_5 \equiv V_8$

この８つのトリプルをＡスペースに格納する。ここで（ａ，ｂ）、（ｃ，ｄ）、（ｇ，ｈ）はマルチプルヒットを起し、さらに（ａ，ｂ，ｃ，ｄ，ｅ）がコンフリクトを起したとすると、内部構造は〔図１１－２〕のようになる。



〔図１１－２〕　Ａスペース内の構造

注）

b： $A_1 \cdot O_1 \equiv [A_6 \cdot O_6 \equiv V_2]$ の $[A_6 \cdot O_6 \equiv V_2]$は、ブラケテッド・トリプルと呼ばれ、アイテム．エクスプレションの一つである。

（例） NUMBER．〔PART・SQUARE≡LINE〕≡N4

マルチプルヒット、コンフリフトが起きた時に、初めのトリプルはハッシング結果のアドレスに格納され、次のトリプルからは、別に用意された領域に格納し、マルチプルヒット．リンクでつながれる。

又セルの内容の1～7で表現されている部分は、〔表11－1〕で示すようなセルの状態を表わしている。

〔表11－1〕

タイプ	状　　態
0	空セルを表わす
1	マルチプルヒットのメンバーである。
2	空セルを表わす
3	コンフリクトを起し、マルチプルヒットの先頭セルである。
4	マルチプルヒットの先頭セルである。
5	ブラケテッド．トリプルを持つ。
6	マルチプルヒットもコンフリクトも起さないセルである。
7	コンフリクトを起したセルである。

このようにスペース内では多種のセルを管理しなければならない。さらにコンフリクト、マルチプルヒットが多くなるに従がい、リスト処理を伴うようになり処理速度がおちる。これらの点においてはまだ問題が残されている。

ステートメント機能

LEAPでは、データ．タイプとしてアイテム、アイテム変数、ローカル、セットの4つを設け、独自のデータをもつアイテムをトリプルあるいはセットの形で関連表現をしている。

(1) アイテム

アイテムはユニークな名を持ち、関連づけの基本となる基本要素であり、独自のデータを持つことができる。そのデータ・タイプはベース言語のデータ属性（整数、実数、ブーリアン、アレイ……）をもつ。

(2) アイテム変数

アイテム変数は、アイテムをその値とするもので、検索の時などに、アイテムの肩代りとして使用される。アイテム変数はデータ属性をもっても持たなくてもよい。アイテム変数にアサインされたアイテムからデータを検索するとその属性はアイテム変数の属性はアイテム変数の属性にあわされる。

(3) ローカル

ローカルもアイテムを値とし、アイテムの肩代りをする点はアイテム変数と同様であるが、ローカルは検索を行なうＦＯＲ　ＥＡＣＨステートメント中にあってくり返し変数として使用される。

(4) セット

アイテムの順序性をもたない集合である。

セットには、データ・ストラクチャに実体をもつセットと、オペランド評価の結果システム内部に一時的に作成されるセットがある。これらのセット間の関係を調べる機能はもつがセット間の関係から新たなセット（セットの和集合、差集合、積集合等）を定義する機能はもたない。さらにセット間の関連を記述することもできない。

ここでは、関連は主にトリプルで表現され、セットは単に、ある条件を満たすアイテムの集合として扱われているが、将来はこれをset theoretic data structureへ発展させようという動きがみられる。

ＬＥＡＰステートメントは機能上次のように分類される。

(1) アイテムに関するもの

(2) トリプルの創成と消去

(3) セットに関するもの

(4) 検索ステートメント

(5) 論理演算子

(1) アイテム・エクスプレション

アイテムの表現法は、次のものがある。

① 宣言ステートメント　ＩＴＥＭで宣言されるアイテム

（例） ＲＥＡＬ　ＡＲＲＡＹ　ＩＴＥＭ　ＰＯＩＮＴ１

ＰＯＩＮＴ１は実数で配列のデータをもつアイテムである。

② ＮＥＷＩＴＥＭステートメントにより動的に創成されるアイテム

（例） ＮＥＷＩＴＥＭ → Ｘ

ここでＸはアイテム変数である。新しいアイテムが作られ、アイテム変数にアサインされる。

③ 単項演算子"ｎ"によって動的に創成されるアイテム

（例） ＭＡＫＥ　ＮＵＭＢＥＲ・ＨＡＮＤ≡ｎ２

ＮＵＭＢＥＲ・ＨＡＮＤ≡ｎ２　というトリプルが創成されるが、ｎ２は実行時に作成されるアイテムであり、アイテム名ｎ２により、値として２をもつことを表現している。

④ ブラケッド・トリプル

（例） ＭＡＫＥ　ＮＵＭＢＥＲ．〔ＰＡＲＴ・ＳＱＵＡＲＥ≡ＬＩＮＥ〕≡ｎ４

〔ＰＡＲＴ・ＳＱＵＡＲＥ≡ＬＩＮＥ〕はブラケッド・トリプルと呼ばれるアイテム．エクスプレションである。

○ アイテムの消去

動的に消去されたアイテムは消去することができるが、創成ステートメントで創成されたアイテムは消去できない。アイテムはＲＥＣＬＡＩＭステートメントにより消去される。

（例） ＲＥＣＬＡＩＭ　Ｘ

ＮＥＷＩＴＥＭ → Ｘで創成されたアイテムＸを消去する。

(2) トリプルの創成と消去

○ トリプルの創成

トリプルはＭＡＫＥステートメントにより創成される。

（例） ＭＡＫＥ　ＳＴＡＲＴ．ＬＩＮＥ１≡ＰＯＩＮＴ１

ＳＴＡＲＴ．ＬＩＮＥ１≡ＰＯＴＮＴ１というトリプルを連想記憶に格納する。

○ トリプルの消去

トリプルはＥＲＡＳＥステートメントにより消去される。

（例） ＥＲＡＳＥ　ＳＴＡＲＴ．ＬＩＮＥ１≡ＰＯＩＮＴ１

ＳＴＡＲＴ・ＬＩＮＥ１≡ＰＯＩＮＴ１というトリプルを消去する。

(3) セットエクスプレション

セットは宣言ステートメントで宣言され連想記憶内に実体をもつセットと、参照するために一時的に創成され、連想記憶内には実体をもたないセットがある。

① 宣言ステートメントで宣言されるセット

（例） SET　POINT

セットPOINTが創成される。創成時は空セットである。

② アイテムの集合で表わされる一時的なセット

（例） {POINT1, POINT2, POINT3}

アイテムPOINT1, POINT2, POINT3をカッコでくくった形をとる。このセットは連想記憶に実体をもたず、参照する時に一時的に創成される。

③ ある条件を満たすアイテムの集合であるセット

（例） A、Bをアイテムとした時

A・B：A・B≡Xを満たすようなXの集合を表わすセット。

A＇B：A・X≡Bを満たすようなXの集合を表わすセット

A＊B：A・BUA＇Bを満たすようなXの集合を表わすセット

これらのセットは参照されるだけで連想記憶に実体をもたない。

○ PUTステートメントによりセットにメンバーが追加される。

（例） PUT　POINT　IN　POINT1

セットPOINTにアイテムPOINT1をメンバーとして追加ずる。

○ REMOVEステートメントにより、セットからメンバーが削除される。

（例） REMOVE　POINT1　FROM　POINT1

セットPOINTからメンバーであったアイテムPOINT1を削除する。

(4) 検索ステートメント

あるアイテムのグループに対して、同一の処理をくり返し実行するためにFOR EACHステートメントがある。

FOR EACH <条件文> DO <ステートメント>

条件文を満たすアイテムに対してDO以下の処理をする。

（例） FOR EACH ELMENT・FACE1≡X DO PUT X IN POINT

ここでXはローカルと呼ばれ、アイテム変数と同様にアイテムの肩代りをする。ただし、ローカルはこのFOR EACH文の中でのみ使用される。

<条件文>は次のような形式をとる。

<条件文>:: =<ローカル><ＩＮ><セット>|<他のオペランド>

<他のオペランド>:: =<アイテム．エクスプレション><バイナリー・アソシェーション・オペレータ><ローカル>|<ローカル><バイナリーアソシェーション・オペレータ><アイテム・エクスプレション>|<アイテム．エクスプレション><バイナリー・アソシェーション・オペレータ><他のオペランド>

さらに次のような制約がある。

① 少なくとも一つ以上のオペランドがローカルでなければならない。

② トリプルの要素が３つ共ローカルであってはならない。

③ 単項演算子ｎによる表現は許されない。

（例） ＴＯＭの息子達の年令和を求める。

```
ＳＵＭ← ０
ＦＯＭ ＥＡＣＨ ＦＡＴＨＥＲ．Ｘ≡ＴＯＭ
  ＤＯ ＳＵＭ← ＳＵＭ+ＤＡＴＵＭ（Ｘ）
```

(5) 論理演算子

検索のための、二項演算子'∈'，'⊂'，'≡'と単項演算子'‖'，'γ'，'ＩＳＴＲＩＰＬＥ'がある。

○ 二項演算子

① <アイテム．エクスプレション>∈<セット．エクスプレション>

指示されたアイテムが、指示されたセットのメンバーであればＴＲＵＥ、そうでなければＦＡＬＳＥとなる。

（例） ＰＯＩＮＴ１ ∈ ＰＯＩＮＴ

ＰＯＩＮＴ１がＰＯＩＮＴのメンバーであればＴＲＵＥ、そうでなければＦＡＬＳＥ。

② <セット・エクスプレション> ⊂<セット・エクスプレション>

左項のセットが右項のセットの部分集合であればＴＲＵＥ、そうでなければＦＡＬＳＥとなる。

（例） ＦＡＣＥ１ ＦＡＣＥ２

ＦＡＣＥ１： {ＰＯＩＮＴ１，ＰＯＩＮＴ２，ＰＯＩＮＴ３}

ＦＡＣＥ２： {ＰＯＩＮＴ１，ＰＯＩＮＴ２，ＰＯＩＮＴ３，ＰＯＩＮＴ４}

とするとＦＡＣＥ１ ⊂ ＦＡＣＥ２はＴＲＵＥとなる。

③ ＜セット．エクスプレション＞≡＜セット．エクスプレション＞

左項のセットと右項のセットのメンバーが等しいときは、ＴＲＵＥ、そうでないときはＦＡＬＳＥとなる。

（例） ＦＡＣＥ１ ≡ ＦＡＣＥ２

(2)と同じ条件では、ＦＡＣＥ１≡ＦＡＣＥ２はＦＡＬＳＥ。

○ 単項演算子

① ‖＜セット・エクスプレション＞

指示されたセットのメンバー数を返す。

（例） ‖＜ＦＡＣＥ１＞

前の例と同じ条件では'３'となる。

② ↑＜アイテム・エクスプレション＞

アイテムに値を与える。

（例） ５ → ↑ ＮＵＭＢＥＲ

アイテムＮＵＭＢＥＲにデータとして'５'を与える。

③ ＩＳＴＲＩＰＬＥ ＜トリプル＞

指示されたトリプルがストアされていればＴＲＵＥ、そうでなければＦＡＬＳＥとなる。

（例） ＩＳＴＲＩＰＬＥ ＳＴＡＲＴ・ＬＩＮＥ１≡ＰＯＩＮＴ１

④ ＤＡＴＵＭ ＜アイテム＞

指示されたアイテムのデータをとりだす。

（例） ＤＡＴＵＭ（ＬＩＮＥ１）

次にＬＥＡＰで書かれたプログラム例を示す。

このプログラムは、いくつかの図形の中から周の一番大きな多面体をみつけ、その名前、辺の数、その周をプリントするものである。

そして既にＰＡＲＴ．ＦＩＧＵＲＥ≡ＯＢＪＥＣＴ，　ＴＹＰＥ．ＯＢＪＥＣＴ≡ＰＯＬＹＧＯＮ，ＳＩＤＥ．ＯＢＪＥＣＴ≡ＬＩＮＥのトリプルでアイテムＦＩＧＵＲＥ，ＯＢＪＥＣＴ，ＬＩＮＥの関連は表わされているものとする。

又、多面体の辺の数と、その辺の長さはデータとして格納されているとする。

```
ITEM PART, SIDE, TYPE, POLYGON;
REAL SUM, MAX;
LOCAL X;
REAL LOCAL Y;
INTEGER ITEMVAR LARGEST;
READ(FIGURE);
MAX ← 0;
FOR EACH PART·FIGURE≡X AND TYPE·X≡POLYGON
    DO
  BEGIN SUM ← 0
  FOR EACH SIDE·X≡Y DO
  SUM ← SUM+DATUM(Y);
  IF SUM ≥ MAX THEN
  BEGIN LARGEST ← X;
  MAX ← SUM; END;
  END;
WRITE(LARGEST, DATUM(LARGEST), MAX)
END
```

参　考　文　献

1 POGO : Programming — Oriented Graphics Operation
by B. W. Boehm, V. R. Lamb, R. L. Mobley, and J. E. Rieber
The RAND Corporation Santa Morica California

2 A System for implementing interactive applications
by F. C. Cneu and R. L. Dougherty
IBM SYSTJ vol. 7. №3 & №4 1968

3 A Computer graphics system for block diagiam problems
by L. A. Belaby, M. W. Blasgen, C. J. Evangelisti and
R. D. Tennison

IBM SYST J №2, 1971

4 BIOMOD－An interactive computer graphics system for modeling

by G. F. Groner, R. L. Clark, R. A. Berman and E. C. Deland

The Rand Corporatsn Santa Monica, California

FJCC, 1971

5 INSICHT－An interactive graphic instructional and for system analysis

by M. J. Merritt & R. Sinclair

University of Southern California Los Angeles California FJCC 1971

6 OLBA－実験的ブロック解析システム

大須賀節雄、山内平行、東京大学宇宙航空研究所

情報処理 vol 12, №2, Feb. 1971

7 SKETCHPAD; A man-machine graphical communication system

by I. E. Sutherland Lincoln Laboratory, M. I. T.

Lexington Mass, SJCC 1963

8 A multilevel modeling structure for interactive graphic design

by H. B. Baskin and S. P. Morse

IBM SYST J №3 & №4, 1968

9 コンピュータ・グラフィックス(Ⅱ)

大須賀節雄　東京大学宇宙航空研究所

10 An experimental display programming language for the PDP－10 computer

by William M. Newman Information Research Laboratory

University of Utah October, 1969

11 Multi－function graphics for a large computer system
by Carl Christensen and Elliot N. Pinson Bell Telephone Laboratories, Incorporated Murray Hill, New Jersey
FJCC, 1967

12 GRAF: Graphic Additions to FORTRAN
by A. Hurwitz and J. P. Citron
IBM Los Angeles Scientific Center Los Angeles, California
SJCC, 1967

13 GPL/I－A PL/I extension for computer graphics
by David N. Smith Boeing Computer Services Company Inc., Wichita, Kansas
SJCC, 1971

14 AIDS－Advanced Interactive Display System
by T. R. Stack and S. T. Walker
National Security Agency, Fort George G. Meade, Maryland
SJCC, 1971

15 Un software de base pour consoles graphique Bulletin de L′IRIA n° 8, 1971

16 Graphic language translation with a language indendent processor
by Ronald A. Morrison General Electric Company Chincinnati, Ohio
FJCC, 1967

17 A language for three-dimensional geometry
by P. G. Comba
IBM SYST J №3 & 4, 1968

18 A subroutine package for FORTRAN
by A. D. Rully
IBM SYST J №3 & 4, 1968

19 Modeling in three dimensions
by A. Appel
IBM SYST J №3 & 4, 1968

20 A system for interactive graphical programming
by William M. Newmman Harvard University Cambridge, Massachusetts
SJCC, 1968

21 コンピュータ・グラフィックスにおけるデータ構造の問題
古川康一　電子技術総合研究所　情報システム研究室

22 Compound data structure for computer aided design ; a survey
by J. C. Gray University of Cambriage, Cambridge, England
A. C. M 1967

23 The CORAL language and data structure

24 APL－A languagc for associative data handling in PL／I
by George G. Dodd Research Laboratories, General Motors Corporation, Warren, Michigan
FJCC, 1966

25 ASP－A ring implemented associaive structure package
.by C. A. Lang University Engineering Laboratory, Cambridge, England
A. C. M vol 11, №8, Angust, 1968

26 The LEAP language and data structure
by Paul D. Rovner Massachusetts Institute of Technology, Lincoln Laboratory Lexington

Massachusetts, USA
and Jerome A. Feldman Computer Science Department,
Stanford University Stanford, California, USA
IFIP, 1968

27 An Algol—based associative language
by Jerome A. Feldman Stanford University, stanford,
California
and Paul D. Rovner M. I. T. Lincoln Laboratory,
Lexington, Massachusetts
A. C. M. vol 12, №8, August 1969

28 Auxiliary—storage associative data structure for PL/I
by A. J. Symond
IBM SYST J №3 & 4, 1968

29 Description of a set—theoretic data structure
by David L. Childs University of Michigan Ann Arbor,
Michigan
FJCC, 1968

30 Data structures and techniques for remote computer graphics
by Ira W. Cotton Sperry Rand Corporation
Philadelphia, Pennsylvania
and Franks Greatorex, JR Adams Associates
Bedford, Massachusetts
FJCC, 1968

31 Scatter Storage Technique
by R. Morris
A. C. M vol 11, №1, January 1968

32 A ring structure processor for a small computer
by N. E. Wiseman and J. O. Hiles
The Computer Journal, Febrary 1968

33 A note on compiling display file from a data structure
by N. E. Wiseman
The Computer Journal, August 1968

34 EDSPについて
古川康一　電子技術総合研究所
情報処理学会　昭和45年度

35 The AED free storage package
by Douglas T. Ross massachusetts Institute of Technology, Cambridge, Mass.
A. C. M vol 10, №8, August, 1967

第2章　UNGL

第2章 UNGL

第 1 節 概 要

1－1. 概 要

UNGLは、グラフィック・オペレーティングシステムCGOS (Comprehensive Graphic Operating System) の下で、開発を進めている汎用グラフィック言語である。UNGLはFORTRANをベース言語とするプリプロセッサー形式の言語で、FORTRANに、次の様な面で機能的拡張を加えたものである。

- o グラフィック・データの導入とそれに関する処理機能
- o グラフィック・入出力機能
- o 割り込み処理に関する機能
- o 汎用データ・ストラクチャへのアクセスの手段

又設計にあたって特に留意した事は次の様な点であった。

(1) 汎用性

特定のアプリケーション向きの言語でなく、なるべく広範囲のアプリケーション・プログラムが実現出来る様な言語である事が望ましい。

(2) グラフィック・システムの持つ機能が十分発揮出来ること。

(3) マシン・インディペンデントであること。

グラフィックスのハードウェアー、ソフトウェアーがまだ流動的であることを考えると、仲々設定しにくい目標である。

少くとも、特定のハードウェアー、ソフトウェアの存在を前提とする様な機能はとり入れるべきではない。ごく一般的なグラフィック・システムで実現できる様な言語であることが望ましい。

(4) ハイレベルであること。

グラフィック言語に於ては特に、図形データ、関連データ、割り込み処理等、一般のプログラミング言語から見ると、新しい概念を必要とするものが多く、その扱いも複雑である。使いやすさを前提とするには、これ等の扱いに関する概念を明確にし、適当な表現形式を考える必要がある。

UNGLのこれらに対する扱いに関して、特徴的な点を以下に記す。

図形データの扱いに関して

① 図形データの概念が明確である。

図形データを表現するデータ・タイプとして新たに4つのデータ・タイプ、イメージ、イメージ・データ、図形変換マトリックス、論理化因子が設けられている。

これ等は、いずれも図形に関係する量を表現し、その概念上の区別も明確である。

② 3次元図形の扱いが可能

3次元イメージによる、3次元図形の扱いも可能である。

③ 図形操作が容易

図形操作を表わすいくつかのオペレータがありプログラム上で自由に図形操作を表現できる。

④ 表現形式が優れている。

複雑な図形も、イメージ・エクスプレッションの中に端的に表現される。

⑤ 画面管理の機能がある。

画面はフレームと云う単位で管理され、保存しておくことが出来る。

⑥ 各種の画面操作が可能

フレームに出力された図形に対して各種の操作を加えることができる。

割り込み処理に関して

① プログラム・ステートの概念で取り扱うことが出来る。

割り込み処理過程をプログラムステートの遷移と云う形式で表現することが出来る。又従来のステート・ダイヤグラム・アプローチでは、プログラム・ステートがプログラム作成時にスタティックに決定されてしまうのに対しUNGLでは、これを動的に再定義する事も可能である。又システムの保有する状態をステート0として扱うことにより、従来のON文による割り込み処理の概念もそのまま通用する形になっている。

② 非同期処理が可能である。

リエントラント構造を備えた、割り込み処理ルーティンに対しては、非同期処理が認められる。

データ・ストラクチャ操作に関して

① LEAPタイプの連想記憶機構へのアクセスが可能である。

② 論理化因子により、図形構造と問題用データ構造の関連がつけられる様になっている。

又この種のデータ構造を採用した動機については、次の様に考えている。

o 一般に、汎用のデータストラクチャに於ては、ハッシュ技法に基本を置いた、連想記憶方式が、効率、2次記憶への拡長性と云う点で最も優れているとされていること。

o 原理的な面では確立されているが、技術的な面では未だ検討の余地があり、開発的意義がある。

1－2. CGOSについて

ＣＧＯＳはＳ４５～Ｓ４６にかけて当センターで開発されたグラフィック・オペレーティングシステムである。

詳しくは、４６年度報告書「ディスプレイ・システムの研究開発」に紹介されているので、ここでは、そのハードウェア構成、ソフトウェア構成についての簡単な説明にとどめる。

① ハードウェア構成

全体は、ＨＩＴＡＣ－８４００、ＨＩＴＡＣ－８８１１ の２台の処理装置から構成され、データ交換装置を介して両プロセッサー間の情報のやりとりが可能になっている。

〔図１－１〕にその構成を示す。

ディスプレー装置には、ライトペン、ファンクションキー、ダイヤルが附属している。

② ソフトウェア構成

ＣＧＯＳに於ける各プログラムの関連を〔図１－２〕に示し以下にその機能について簡単にふれる。

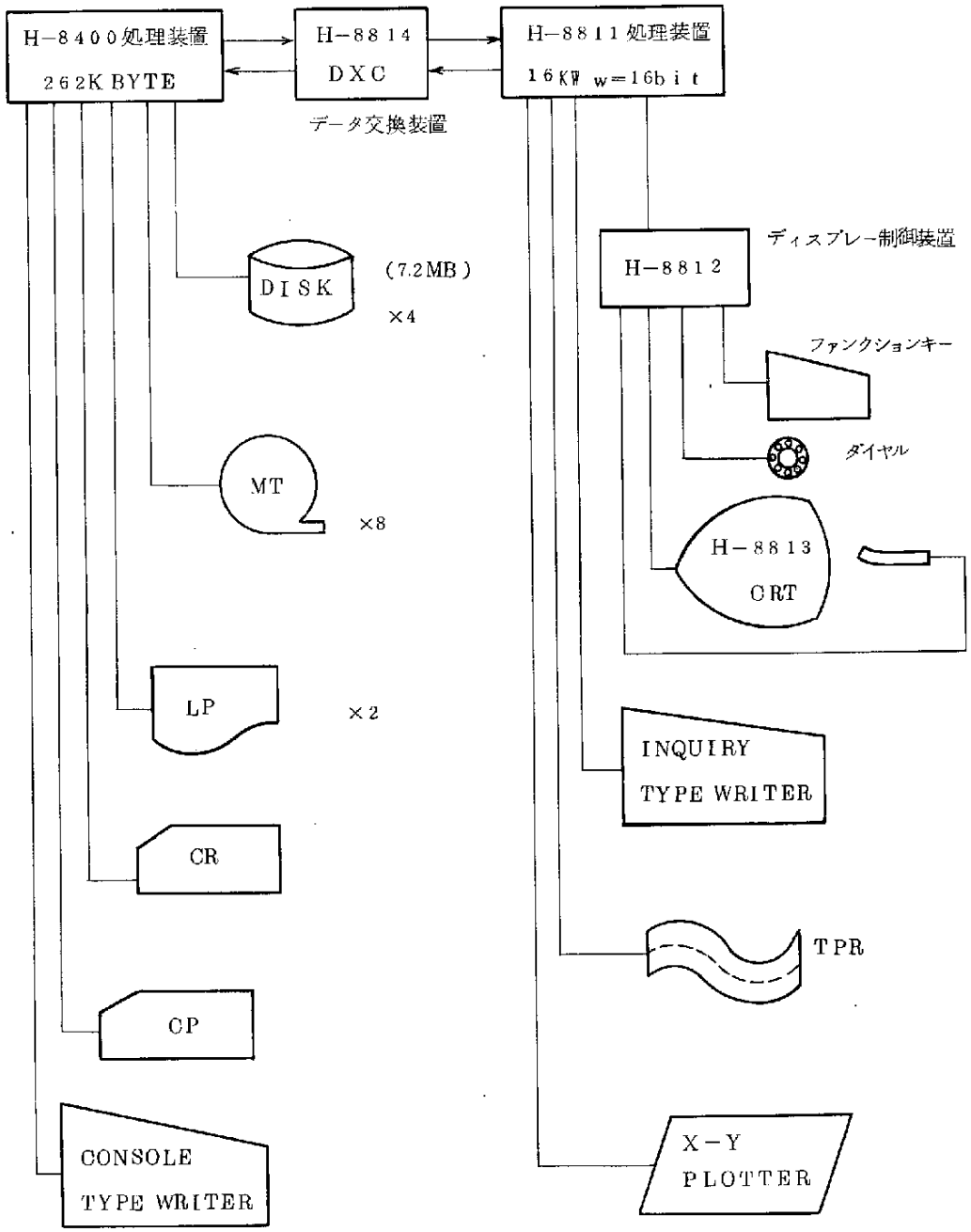
o ＧＪＭＯＮ （Graphic Job Monitor）

ＧＪＭＯＮは Ｈ－8400 ＥＸＥＣＵＴＩＶＥの下に常駐し次の機能を果す。Ｈ－８４００ Ｈ－８８１１間の相互データ転送、ＣＲＴからのユーザ・プログラムの起動およびモニタリング、グラフィック・アプリケーション・プログラムとの相互データ転送、グラフィック・コマンドに対する処理。

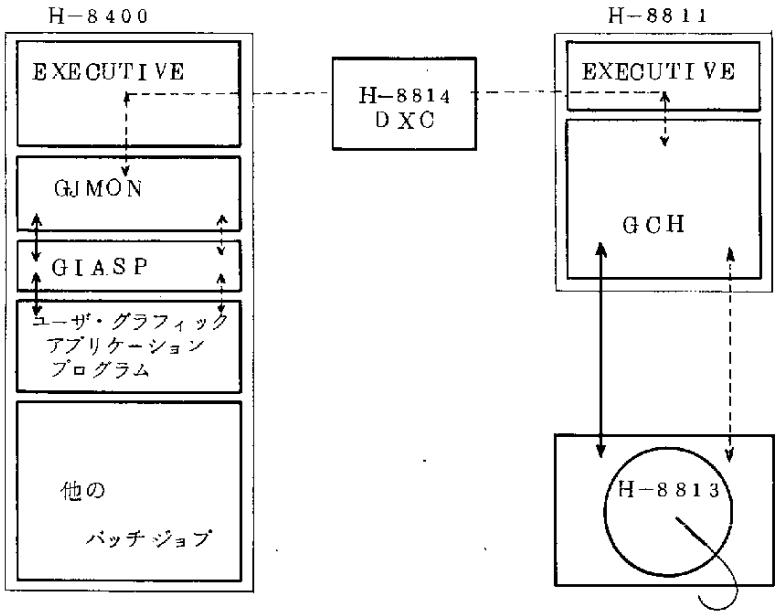
o ＧＣＨ （Graphic Communication Handler）

ＧＣＨは Ｈ－8811 ＥＸＥＣＵＴＩＶＥの下に常駐し次の機能を果す。

Ｈ－８８１１ 、Ｈ－８４００ 間の相互データ転送、システム・レスポンスの表示、図形データから図形コマンドへのコンパイリングおよび図形表示、各種割込みの処理。



〔図1−1〕 ハードウェア構成



GJMON:Graphic Job Monitor

GCH :Graphic Communication Handler

GIASP:Graphic Interaction Analysis Subroutine

−−−−→ データ・フロー

⟶ コントロール・フロー

〔図1−2〕 ソフトウェア構成

第２節　UNGLの機能

２－１．図形データの扱いに関する機能

【グラフィック・データ】

ＵＮＧＬが扱うグラフィック・データには、４つのデータ・タイプ、イメージ、イメージ・データ、図形変換マトリックス、論理化因子がある。

イメージは、高度な図形の概念を表わし、構造化、図形操作の対象となる。

イメージ・データはイメージよりさらに基本的な図形データで、イメージの実データを定義するのに使用される。図形変換マトリックスは、各種の図形変換を行う時に必要となる変換量（回転量、移動量等）を表わす。

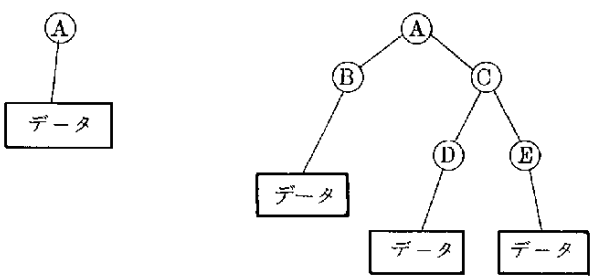
論理化因子は、図形の非幾何学的な性格（論理名、輝度等）を表わすデータ量として使用される。

又これらのグラフィック・データ間に適用されるいくつかのグラフィック・オペレータがあり、図形操作、図形の構造化はグラフィック・データ、グラフィック・オペレータによるイメージ・エクスプレッションと云う形式で表現される。

(1)　イメージ

イメージは構造化の対象となる図形要素（図形のまとまり）を表わす。

イメージの実体（中味の実際の図形データ）は、基本的には次に述べるイメージ・データによって定義されるが、これを既に定義された他のイメージを引用して階層的に記述することも出来る。（〔図２－１〕はイメージＡのプリミティヴな表現と階層的な表現の違いを図示したものである）



〔図２－１〕

しかしユーザは特にこの様な階層構造を意識する必要はない。後述するイメージ・エクスプレッションの中に暗にこの様な構造が存在するわけである。

この様に、イメージはサブピクチャーとか図形モデルとか云われるものに近い概念を表わすと考えればよい。又イメージには2次元図形を表わす2次元イメージと3次元図形を表わす3次元イメージがあるが、話が複雑になるので後者については一番最後に説明することにする。

① イメージの宣言

イメージはユニークな識別名を持つ。その構成規則はFORTRANの変数名と同様である。

イメージ名は宣言文で与えなくてはならない。

次の例で TREE,RIVER はそれぞれイメージである。

〔例〕

IMAGE*2 TREE , RIVER

(注) *2は2次元イメージである事を示す。同様に3次元イメージの場合は*3とする。

(2) イメージ・データ

前述した様にイメージ・データはイメージの実データを定義するための図形データである。これにはやはり2次元イメージ・データと3次元イメージ・データがあるが、後者については3次元イメージの項でふれる。イメージ・データの一般形式は次の様に与えられる。

一般形式 : 'G·FUNCTION , G·FUNCTION,…… '

ここでG·FUNCTION はグラフィック・ファンクションのことで、これにはLINEと.TEXTがある。

① LINE ファンクション

LINEファンクションは次の2つの形式で使用され、点列を結ぶ直線を表現する。

形式1

LINE(x_0,y_0,x_1 ,y_1 …… x_n,y_n)

始点(x_0,y_0) から始めて、点列(x_1 y_1) ……(x_n,y_n) を結ぶ直線を表わす。

形式2

LINE (x,y,i,j,k) (x,yは配列名である) iは初期値、jは終値、kはきざみを表わす。

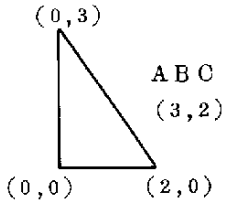
② ＴＥＸＴ

次の形式で、文字ストリングを表現するのに使用される。

ＴＥＸＴ（'文字ストリング',xo,yo,TYPE）

（xo,yo）の位置から、ＴＹＰＥで示された型式（大文字、小文字）に従って文字ストリングを書く。

例えば〔図２－２〕に示された図形は、イメージＰＩＣＴＵＲＥ として次の様に定義される。




```
PICTURE <= ' LINE( 0,0,2,0,3,0,0,0, ),
           TEXT('ABC', 3,2,1,) '
```

〔図２－２〕

(3) グラフィック・オペレータ

グラフィック・データ間で定義される次の６つのグラフィック・オペレータがある。

① ＋　　併合操作を表わすオペレータ

② <=　　代入操作を表わすオペレータ

③ ・TRS・　　変換（回転、移動、スケーリング）操作を表わすオペレータ

④ ・PERS・　　変換（透視投影）操作を表わすオペレータ

⑤ ・ORTH・　　変換（平行投影）操作を表わすオペレータ

⑥ ｜　　論理的なグルーピング操作を表わすオペレータ

又これ等の間の優先関係は次の様に定義されている。

オペレータ	優先度
.TRS.,.PERS.,.ORTH., ｜	高
＋	低

これらのオペレータのうち特に④、⑤は３次元図形データのみを対象としている。これについては３次元イメージの項で説明することにする。

又③、⑥についてはそれぞれ次の変換マトリックス、論理化因子の項で説明する。

ここでは最も基本的な①，②について説明する。

o　グラフィック・オペレータ「＋」とイメージ代入文について

＋オペレータはイメージの併合を表わす。

例えば２つのイメージA，Bが定義されている時A＋Bは一つのイメージを表わし、それは、A，Bの表わす両図形を併合した図形を表現している。

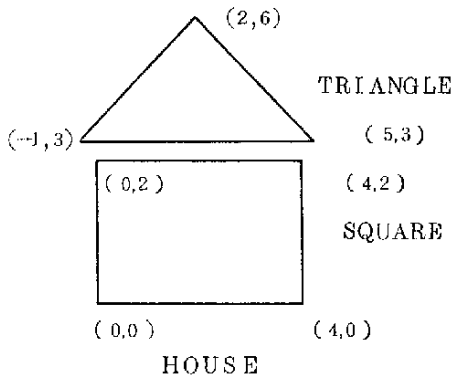
これを新たにイメージCとして定義するには次の形式のイメージ・アサインメント・ステートメントが使用される。

C <= A＋B

次の例はこれ等の関係をより具体的に説明したものである。

＜例＞

イメージHOUSEが、イメージTRIANGLE、イメージSQUAREから構成される時これは次の様に表現される。



〔図２－３〕

TRIANGLE <= ' LINE (－1, 3, 5, 3, 2, 6, －1, 3) '

SQUARE <= ' LINE (0, 0, 4, 0, 4, 2, 0, 2, 0, 0) '

HOUSE <= TRIANGLE＋SQUARE

(4)　図形変換マトリックス

a．図形変換マトリックスは、イメージに対して各種の図形変換を行う時に、その変換量を表わす図形データとして使用される。これには、次の３種類のマトリックスがある。

①　座標変換マトリックス

座標変換マトリックスは、図形の回転、移動、スケーリングに必要な情報を蓄えている。

座標変換マトリックスをイメージに作用させるには．ＴＲＳ．オペレータを使用する。

② 透視変換マトリックス

3次元図形を平面に透視投象する時に必要な情報を蓄えている。

3次元イメージに対して．PERS．オペレータで作用させる。

③ 平行投影マトリックス

3次元図形を平面に平行投影する時に必要な情報を蓄えている。

3次元イメージに対して．ORTH．オペレータで作用させる。

この内、②、③は3次元イメージの項で説明することにし、ここでは①の座標変換マトリックスについて説明しよう。

b．座標変換マトリックス

① 座標変換マトリックスの表現形式

座標変換マトリックスは次の形式で表現される。（xo，yo，θ，Sx，Sy．）

xo，yo …… Sy 等は位置パラメータで、各々は次の情報を表わす。

xo ： x方向の移動量

yo ： y方向の移動量

θ ： 原点を中心とする回転角

Sx ： x方向のスケール値

Sy ： y方向のスケール値

② 座標変換マトリックスによる図形変換

座標変換マトリックスは．ＴＲＳ．オペレータによりイメージに作用しこれに図形変換を施す。

変換の一般的な過程を、次のイメージ代入文を例に説明しよう。

Ａ，Ｂをそれぞれイメージとすると、代入文

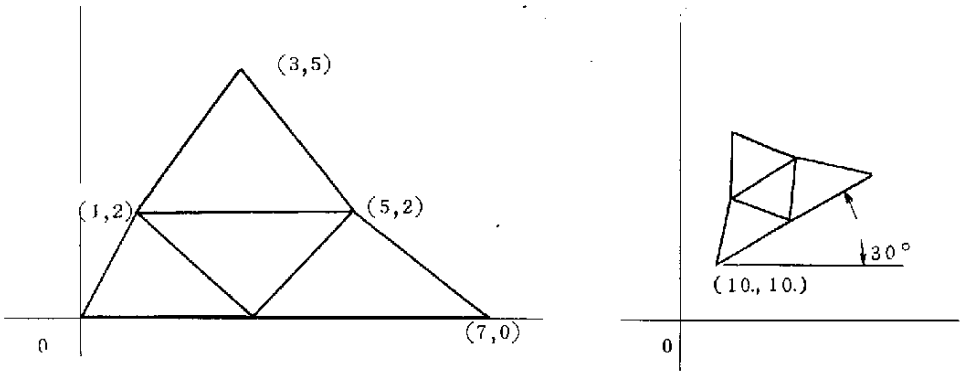
Ｂ<= Ａ．TRS.（x，y，o，Sx，Sy）

は、次の様に説明される。

イメージＡの表わす図形を、イメージＡの座標系で、x方向にSx倍、y方向にSy倍拡大し、原点を中心にθ回転させ、イメージＢの座標系の（xo，yo）に原点移動した図形をイメージＢとして定義する。

〔図２－４〕はこれ等の関係を具体的な例として示したものである。イメージＡが図の様に定義されている時イメージＢは次の様に定義される。

Ｂ<= Ａ.TRS.（10.，10.，30.，0.5，0.5）



イメージA　　イメージB

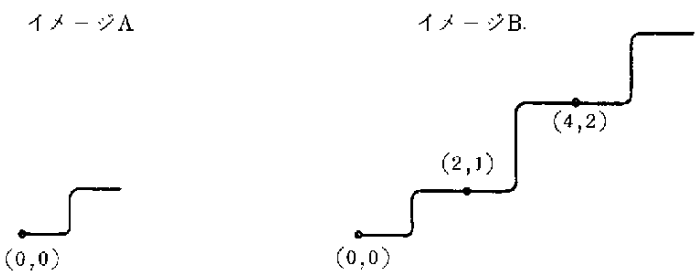
〔図２－４〕

③　座標変換マトリックスの省略時解釈

座標変換ストリックス（２次元）は、５つの位置パラメータで表現されるが、不要なパラメータの記述は煩雑さを増すので、適当にこれを省略することが出来る。

例えば単に（ｘ，ｙ）とするとこれは（ｘ，ｙ，０，１，１）と解釈される又、パラメータを省略をする時は単にデリミター　を記せば良い。

（例）　イメージBはイメージAを使って次の様に表現される。



〔図２－５〕

B <= A＋A．TRS．（2,1）＋A．TRS．（4,2）

(5)　イメージ・エクスプレッション

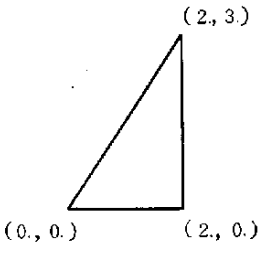
これまでに見てきたように、イメージは単にイメージ・データによって定義されるだけでな

く、他のイメージを図形モデルとして引用したり、これに適当な操作を加えたものを引用したりしながら次々に複雑なイメージを定義してゆくことが出来る。

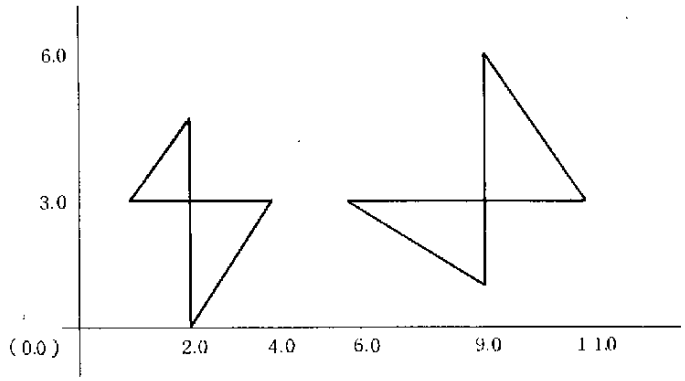
この様な、グラフィック・データとグラフィック・オペレータによるイメージの表現形式をイメージ・エクスプレッションと呼ぶ。

以下にイメージ・エクスプレッションのいくつかの例をあげる。

（例１）　既に定義されているＩＭＡＧＥＡ に、変換をほどこした図形を４つまとめてＩＭＡＧＥＢ と定義する。



イメージ　ＩＭＡＧＥＡ



イメージ　ＩＭＡＧＥＢ

〔図２－６〕

ＩＭＡＧＥＢは次のようなイメージ・エクスプレッションとして表現される。

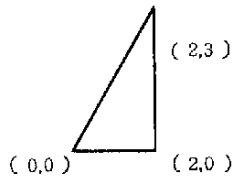
ＩＭＡＧＥＢ⇐　ＩＭＡＧＥＡ．ＴＲＳ．（1.0，3.0，0.0，0.5，0.5）

＋ＩＭＡＧＥＡ．ＴＲＳ．（4.0，3.0，180.0）

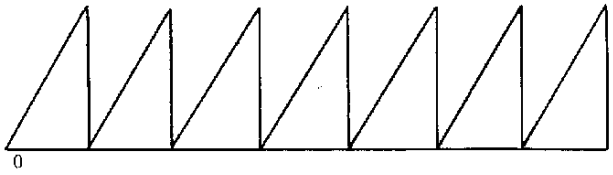
＋ＩＭＡＧＥＡ．ＴＲＳ．（9.0，6.0，－90.0，1.5，0.667）

＋ＩＭＡＧＥＡ．ＴＲＳ．（9.0，1.0，90.0）

（例２）　三角形　ＩＭＡＧＥＡ　を用いて、その連続図形　ＩＭＡＧＥＢを定義する例を示す。



イメージ　ＩＭＡＧＥＡ



イメージ　ＩＭＡＧＥＢ

〔図２－７〕

IMAGEBは次のように定義される。

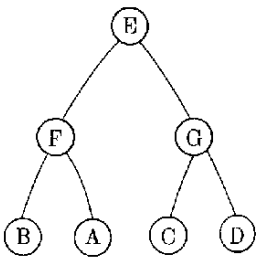
```
IMAGEA <=   'LINE( 0.0,2.0,3.0,0.)'
IMAGEB <=   IMAGEA
DO  100      I = 1,6
X(I)= 2 *  FLOAT(I)
IMAGEB <= IMAGEA . TRS . ( X(I) , Y(0)) + IMAGEB
100  CONTINUE
```

(6) 論理化因子

既に述べた様に、図形構造は、イメージ・エクスプレッションを通じて暗に定義されてゆく。例えば次の3つのイメージ代入文を実行する事により、各イメージの間に図の様な階層関係が生じるだろう。

（例）

```
E <=  F + G
F <=  A + B
G <=  C + D
```



（A～Gはイメージ）　　〔図2－8〕

ところで、一般にこの様なイメージ構造は、図形の幾何学的情報を取り出すには、十分であるが、表示を前提として考えた時には、まだ十分な情報を備えていない。

即ち、イメージの論理的なまとまりを指示する情報が欠けている。

UNGLでは、論理化因子、グラフィック・オペレータ '|' を設けることにより、イメージ・エクスプレッションの中で、任意にイメージの論理的なグルーピングが出来る様に考慮している。

a．論理化因子

論理化因子は、グラフィック・オペレータ '|' によりイメージ（イメージ・エクスプレッション）に付加され、それに対して論理名称を与える。（論理的なまとまりとする。）

論理化因子の表現形式は次の通りである。

論理化因子の一般形式　　<論理名、制御情報>

o　論理名

論理名は対象とするイメージに与える識別名で、ＣＲＴ画面上の図形は、この名称で管理される。

論理名の形式としては、整数、H定数、アイテム名（関連データ）が許される。

o　制御情報

制御情報は、この論理化因子によってグルーピングされる図形に対して与えられる、表示条件（ＣＲＴ制御情報になる。）である。

制御情報は、整数で記され、その値により次の様な制御情報が指示されたことになる。

制御情報の値	意　味
－３	高輝度でライトペンピック不可
－２	中輝度でライトペンピック不可
－１	低輝度でライトペンピック不可
０	ブランク
１	低輝度でライトペンピック可
２	中輝度でライトペンピック可
３	高輝度でライトペンピック可

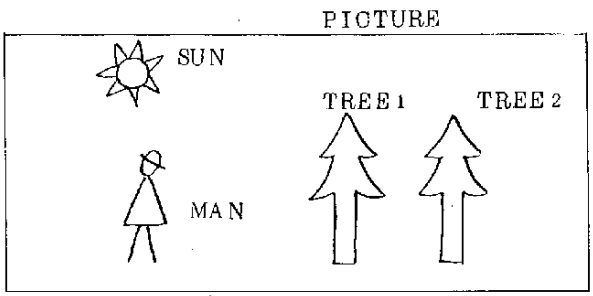
＊　省略時は２がとられる。

b．論理化因子の適用例

論理化因子が具体的にどの様に用いられるかを知るために、１つの例を紹介しよう。

<例１>　〔図２－９〕でイメージPICTURE が次の様に定義されていたとする。

PICTURE<= MAN＋SUN ＋TREE１ ＋ TREE２



〔図２－９〕

この時、次の３つの方法でこれをグルーピングした時、PICTURE が画面表示された時の違いについて述べる。

① PICTURE ⇐ (MAN + SUN+TREE１+TREE２) <１>

② PICTURE ⇐ (MAN + SUN)|<1>+(TREE1+TREE2)|<2>

③ PICTURE ⇐ MAN|<1>+SUN|<2>+TREE1|<3>+TREE2|<4>

どの場合もPICTURE が表示されると、〔図２－９〕の絵がＣＲＴに表示される。

①の場合はPICTURE 全体で一つの識別対象になっている即ちSUN、MAN、TREE１TREE２のどれが指摘されても、全く同じレスポンス'１'が返ってくる。

②では、MANとSUN，TREE１とTREE２が各々一まとまりで画面は２つの識別対象からなっている。

例えばMANかSUN が指摘される、レスポンス'１'が返り、一方TREE１かTREE２が指摘されるとレスポンス'2' が返ってくる。

③の場合はSUN、MAN，TREE１，TREE２が各々１つの識別対象であり、画面は４つの識別可能な図形から構成されている。

即ちMANが指適されると、レスポンス'1'が返り、同様にSUN，TREE１、TREE2に対してはレスポンス'2'，'3'，'4' が返される。

(7) ３次元イメージの扱い

３次元イメージも、イメージ・エクスプレッションに於て、２次元イメージと全く同じ扱いを受ける。

２次元イメージと異る点は、３次元イメージそのものは表示の対象とならないと云うことである。これを表示するために、透視変換又は、投影変換等の操作を通じて一度２次元イメージに変換しなくてはならない。

以下では、主として３次元イメージの変換操作について述べる。

a． ３次元イメージの回転、移動、スケーリング操作

３次元イメージの回転、移動、スケーリング操作は、２次元イメージの場合と同様、.TRS.オペレータと座標変換マトリックス(３次元)が用いられる。

座標変換マトリックス(３次元)の表現形式は次の通りである。

$(xo, yo, zo, \theta xy, \theta z, sx, sy, sx)$

変換の一般的な過程を次のイメージ代入文で説明する。

ここでIMAGEA，IMAGEB をそれぞれ３次元イメージとして、次の文を実行する。

IMAGEB⇐ IMAGEA.TRS.$(xo, yo, zo, \theta xy, \theta z, sx, sy, sz)$

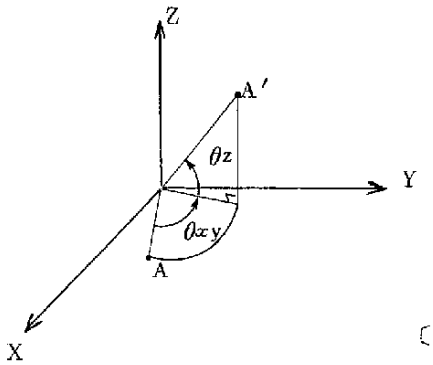
ＩＭＡＧＥＢ は次のように定義される。

① ＩＭＡＧＥＡ を ＩＭＡＧＥＡの座標系においてＸ方向に sx、Ｙ方向に sy，Ｚ方向に sz のスケールファクタをかける。

② さらに、ＸＹ平面において原点中心に θxy 回転させた後、xy 平面からＺ軸の正方向に θz 回転させる。

③ ＩＭＡＧＥＢ は、ＩＭＡＧＥＢの座標系で②の図形を（xo，yo，zo）にポジショニングされた図形として定義される。

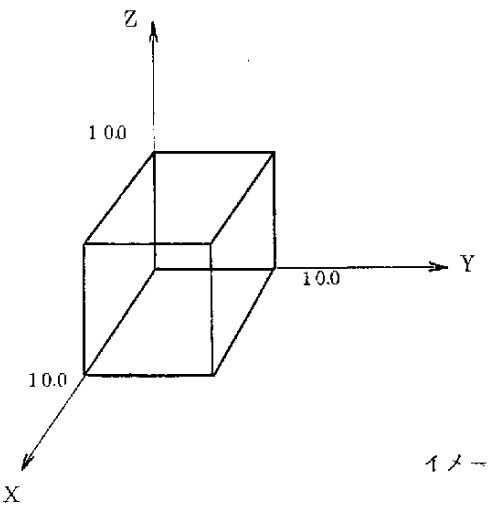
次に点Ａを点Ａ′に移動した時の θxy，θz の関係を図に示す。



〔図２－１０〕

次の図はこの具体的な例を示したものである。

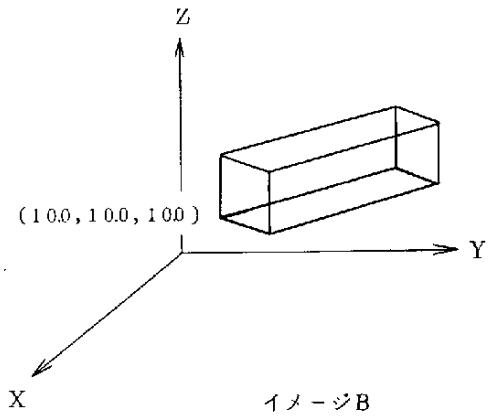
（例）



イメージ Ａ

IMAGE ＊3　　A ， B

B<=A．TRS．(10.0，10.0，10.0，45.0，0.0，0.5，2.0，1.0)



イメージB

〔図2－11〕

b．透視変換

3次元イメージに対して、グラフィック・オペレータ．PERS．と透視変換マトリックスを用いて透視変換を定義することが出来、結果は2次元イメージとなる。

投影面としてはXY平面、YZ平面、ZX平面のうちのいづれかが選べ、任意の視点からこの投影面に透視することができる。

透視変換マトリックスの表現形式は次のとおりである。

(x_0, y_0, z_0, F)

ここで($x0$，$y0$，$z0$　)は視点を表わし、Fは次のような値で投影面を表わす。

F　：　＝1のとき　　XY平面

＝2のとき　　YZ平面

＝3のとき　　ZX平面

次に透視変換の一般的な過程を次のイメージ代入文で説明する。

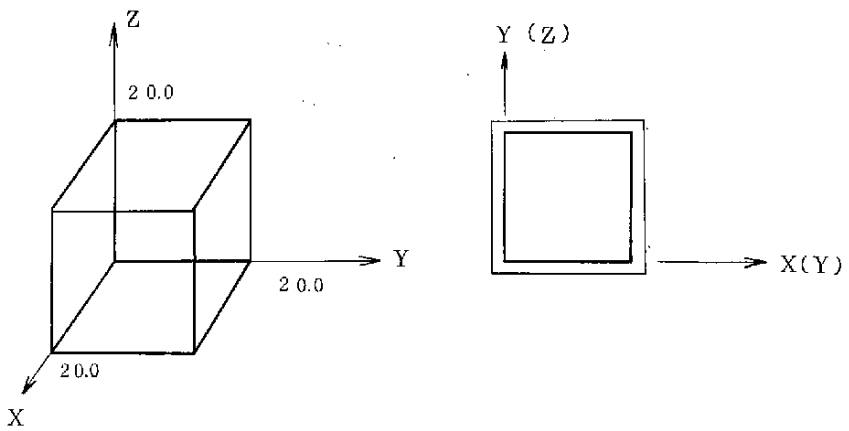
ここでIMAGEAは3次元イメージ、IMAGEBは2次元イメージとする。

IMAGEB<= IMAGEA．PERS．($x0$，$y0$，$z0$，F)

この文を実行することにより、IMAGEB は IMAGEA を視点($x0$，$y0$，$z0$)からFで指定された投影面に透視した図形として定義される。

次に示す例は次の代入文により３次元イメージであるＩＭＡＧＥＡを透視した図形をＩＭＡＧＥＢとしている。

（例）　ＩＭＡＧＥＢ⇐ＩＭＡＧＥＡ．ＴＲＳ．（１００．０，１０．０，１０．０，２）



〔図２－１２〕

ｃ．平行投影

３次元イメージに対して、グラフィック・オペレータ．ＯＲＴＨ．と平行投影マトリックスを用いて平行投影を定義する。投影面としては、ＸＹ平面、ＹＺ平面、ＺＸ平面のうちのいづれかが選べる。即ちこの機能は、ある３次元イメージの三面図を描く時などに利用できる。

平行投影マトリックスは次のような表現形式をとる。

（Ｆ）

ここでＦは投影面を表わし、次のような値をとる。

Ｆ：　＝１　　ＸＹ平面

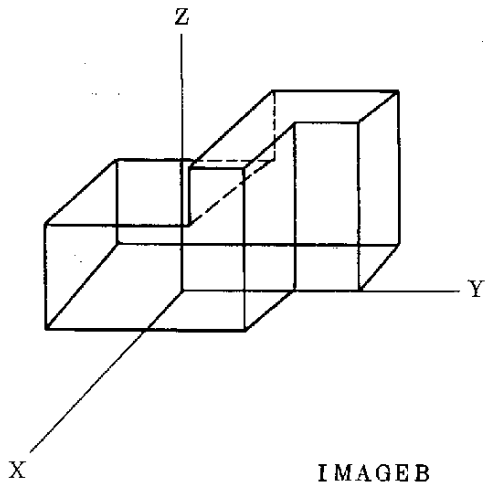
　　＝２　　ＹＺ平面

　　＝３　　ＺＸ平面

次に、平行投影の一般的な処理について次の代入文により説明する。ここでＩＭＡＧＥＡは２次元イメージ、ＩＭＡＧＥＢは３次元イメージとする。

ＩＭＡＧＥＡ⇐ＩＭＡＧＥＢ．ＯＲＴＨ．（Ｆ）

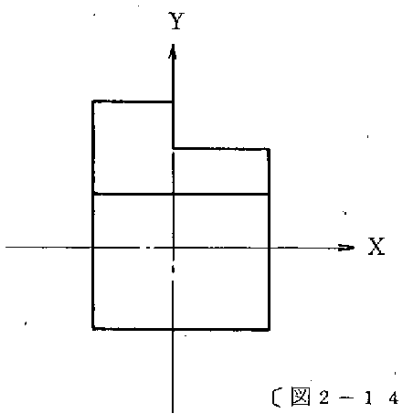
ここで〔図２－１０〕をＩＭＡＧＥＢ とすると〔図２－１１〕はこの平行投影マトリックスのＦの値が１のとき、〔図２－１２〕はＦの値が２のときのＩＭＡＧＥＡを表わしている。



ＩＭＡＧＥＢ

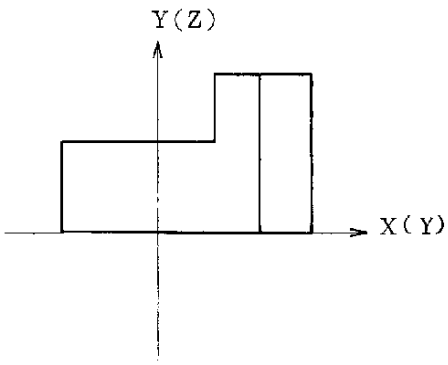
〔図２－１２〕

ＩＭＡＧＥＡ <= ＩＭＡＧＥＢ．ＯＲＴＨ．(1)



〔図２－１４〕

ＩＭＡＧＥＡ <= ＩＭＡＧＥＢ．ＯＲＴＨ．(2)



〔図２－１５〕

【グラフィック入出力】

(1) 表示データ・リストについて

表示は、イメージを対象にして行われる。

イメージを指定して出力ステートメントが実行されると、システムは、イメージ構造をたど

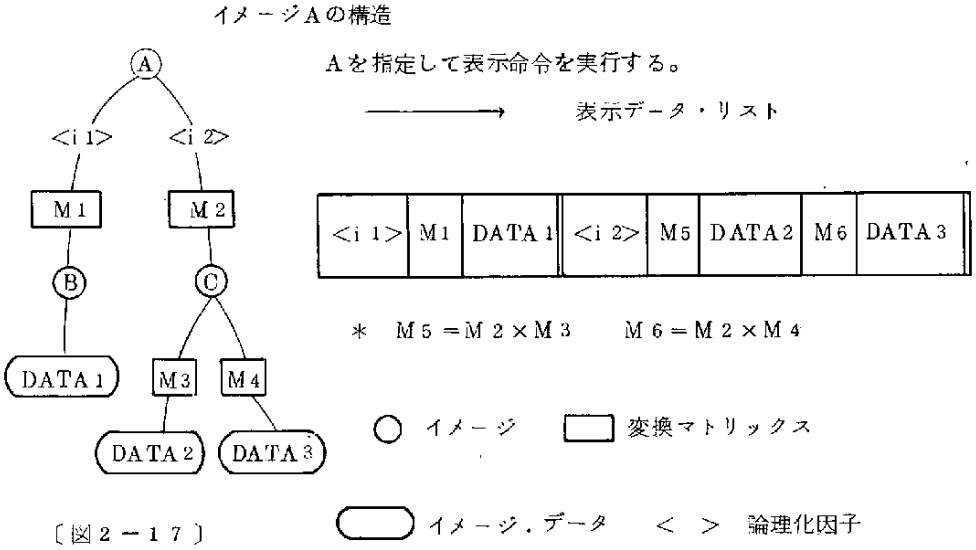
り、各種の図形変換を行いながら最終的に次の形式の表示用図形データを作成する。
これを表示データ・リストと呼ぶ。

論理化因子(i)	変換マトリックスM	図形データ	論理化因子(i')	変換マトリックスM'	

（表示データ・リストの概念図）

〔図２－１６〕

〔図２－１７〕はイメージ構造と、それから作成される表示データ・リストの関係を示したものである。



〔図２－１７〕

表示データ・リストは、まだ完全な表示用データではない。即ちこれを、表示コマンド・オーダに変換するまでには次の様な操作が残されている。

① 変換マトリックスと個々のデータの間の演算

② 論理化因子に従って、コリレーションリストと表示コマンドをジェネレートする。

２プロセッサー．システムの下では、これ等の操作をどちらのプロセッサーのロードとすることも可能であるが、計画としては、①、②を共にサテライトプロセッサーのロードシェアとする予定である。

(2) フレームの概念

イメージが図形として出力される媒体をフレームと呼ぶ。

フレームは０～ｎ（整数）で表わされる識別名を持ち、各々ＣＲＴ画面の１画面に相当する表示データを蓄えている。

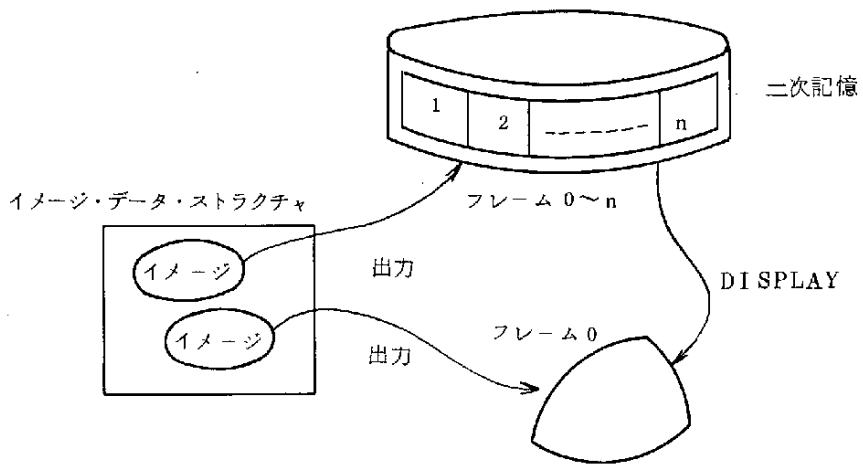
一担イメージが、フレームに出力されると、フレーム上の図形データは、論理化因子で指定されたまとまりで管理される表示データリストに変換され、イメージ構造は失われる。

従ってフレーム上の図形を識別するには、論理化因子で与えた論理名を用いなくてはならない。論理化因子で与えた論理名で識別される図形のまとまりをエンティティー（インスタンス）と呼ぶ。フレーム０はディスプレー・ファイルそのものに相当する。即ち、フレーム０に出力することは画面表示することである。フレーム０に対して為された図形の動的な変更に対してそのヒストリーを残すことは出来ない。

一方、フレーム１～ｎは、表示されない画面で、具体的にはストレージ・デバイスへの出力を意味する。

フレーム１～ｎに対して為された出力はパーマネントなものであり、必要な時点まで残して置くことが出来る。又、フレーム１～ｎを画面表示するためには、後述するＤＩＳＰＬＡＹステートメントが使用される。

（フレームの概念図）



〔図２－１８〕

(3) グラフィック入出力ステートメント

図形出力は、「あるイメージをあるフレームに出力する」と云う形式で行われる。

又フレームに出力された図形に対して、これをエンティティ単位で、修正を加える事も出来る。

具体的な修正動作として、置換、削除、制御情報の変更等がある。

a．イメージの出力

DRAW 文は、指定されたイメージを指定されたフレームに出力する動作を指示する。

一般形式は次のとおりである。

DRAW n，IM1，IM2，……〔，（ウィンドーイング・オプション）〕

n： 出力するフレームの名称 （0～1）

IM1，IM2…… ： 出力するイメージのリスト

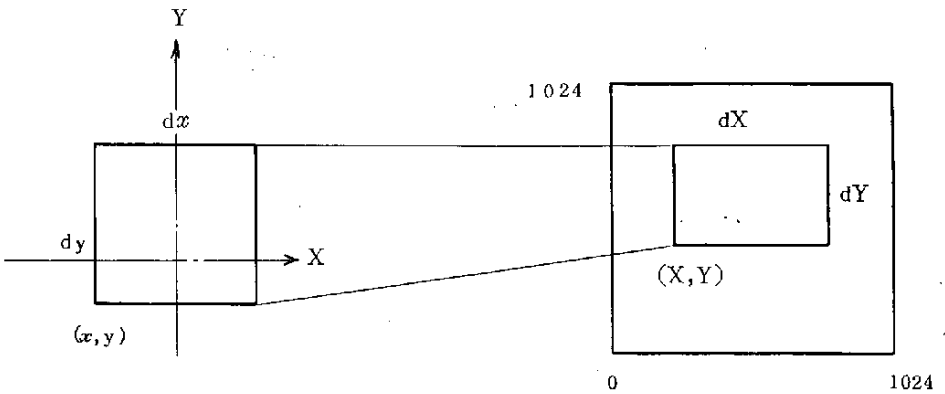
ウィンドイング・オプション：イメージをフレームに出力する時に、イメージの座標系とフレームの座標系との関係を与える。

ウィンドイングオペションの一般形式は次のとおりである。

（x，y，dx，dy，X，Y，dX，dY）

ここで（x，y，dx ，dy）はイメージの座標系の領域、（X，Y，dX，dY）はフレームの座標系の領域を表わし、指定されたイメージの領域をフレームの領域にスケーリングをして表示する。

これ等の関係を〔図2－19〕に示す。

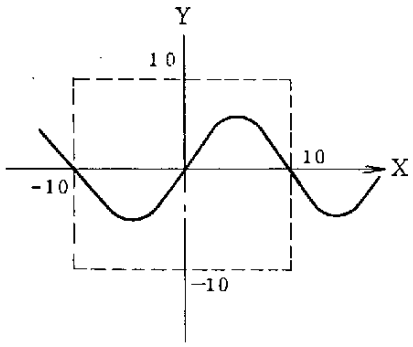


〔イメージの座標系〕 〔フレームの座標系〕

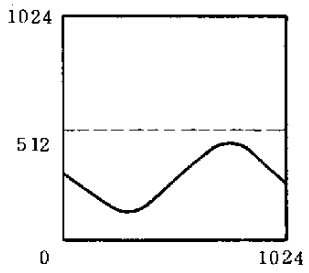
〔図 2－19〕

例えば、次の図のようにに定義されたＩＭＡＧＥＡにウインドイングを施して、フレームに出力すると、〔図２－２０〕のようになる。

ＤＲＡＷ　０，ＩＭＡＧＥＡ，（－１０．，－１０．，２０．，２０．，０，０，１０２４，５１２）



ＩＭＡＧＥＡの座標系



フレーム０の座標系

〔図２－２０〕

①　ウインドイング情報の省略時解釈

ウインドイング・オプションが省略されるとデフォルト値として次の値がとられる。

イメージの領域

（x，ｙ，ｄx，　ｄｙ）　　→　（０．，０．，１０２４．，１０２４．）

フレームの領域

（Ｘ，Ｙ，ｄＸ，ｄＹ）　→　（０，０，１０２４　，　１０２４）

次に、ウインドイング・オプションを全て省略した例を示す。〔図２－２１〕で示されるようにＩＭＡＧＥＡとＩＭＡＧＥＢ は既に定義されているものとし、次のＤＲＡＷステートメントを実行するとＣＲＴ画面には〔図２－２１〕のような図形が表示される

ＤＲＡＷ　０，　ＩＭＡＧＥＡ，　ＩＭＡＧＥＢ

この時のウインドイングオプションとして

（０．，０．，１０２４．，１０２４．，０，０，１０２４，１０２４）がとられる。

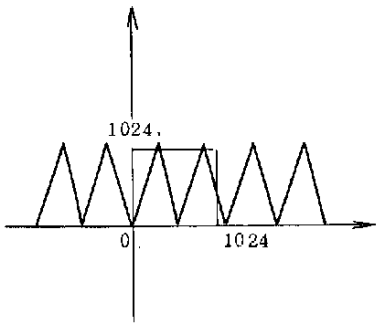
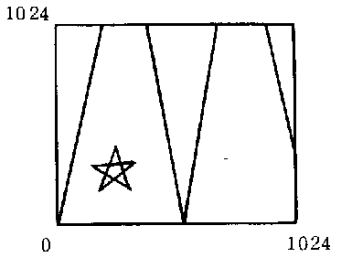
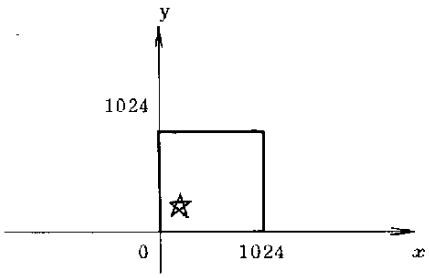


IMAGE Aの座標系



CRT画面



〔図2－21〕

IMAGE Bの座標系

② アニメーション

フレーム0に対する出力として、あるイメージをムーヴィングすることができる。図形表示のウインドイング情報は前で述べたような方法で与え、ムーヴィングのきざみ幅、回数を指定する。

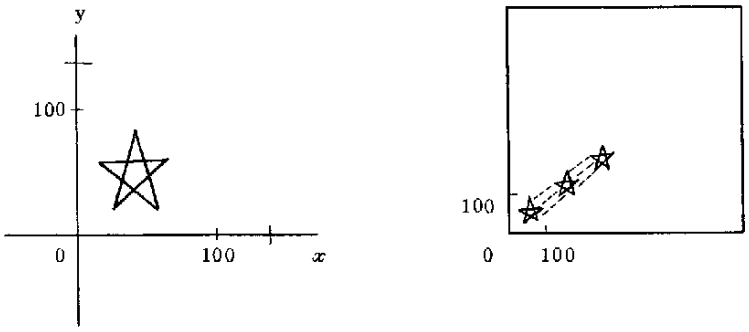
アニメーションはANIMATEステートメントで行う。例えば

ANIMATE IMAGEA (△x, △y, N)〔(x, y, dx, dy, X, Y, dX, dY)〕

はx方向△x、y方向△yのきざみ幅でN回動かすムーヴィングを行う。

次にイメージSTAR にアニメーションを施す例を示す。イメージSTAR は次のANIMATEステートメントにより、フレーム上の（100，100）の位置からx方向y方向のきざみ幅を5として40回移動させられる。

ANIMATE STAR （5，5，40）



イメージ STAR　　CRT画面

〔図2－22〕

(注) アニメーション完了はムーヴィング終了割り込みとして通知される。（割り込み処理の　項参照）

b．フレーム間の図形消去

フレームに描かれた図形はエンティティ単位に消去することができる。フレーム0を指定した場合はCRT画面上の図形が消え、フレーム1～nを指定した時は、そのフレーム内のそのフラグに対応するディスプレイ・オーダーが取り除かれる。

例えば次の様にすると

DELETE 0 ,<ITEM1>

ITEM1を名称とする図形が画面から消える。

(注) DISPLAY ステートメントでCRT画面上に表示された図形を消す時は、フレーム0を指定する。

例えば　フレーム7にITEM3のフラグをもつイメージが出力されていたとする。

DISPLAY 7

DELETE 0,<ITEM3>

これはＣＲＴ画面上のＩＴＥＭ３を消去したことになる。

DELETE 7,<ITEM3>

とすると二次記憶上のフレーム７の中のITEM3が消去される。

ｃ．制御情報の変更

一担フレームに出力された図形に対して、その制御情報を動的に変更することが出来る。

例えば

CONTROL n,<'FLAG',I>

とすると、フレームｎの中の、'ＦＬＡＧ'と云う名称を持つエンティティーの制御情報がＩに変更される。

ｄ．フレーム操作

二次記憶上のフレーム１～ｎをＣＲＴ画面上に表示することが出来る。これにはＤＩＳＰＬＡＹステートメントが使用される。

（例） DISPLAY K

フレームＫをＣＲＴ画面上に表示する。ＣＲＴ画面上になんらかの図形が表示されていても重ね書きをする形をとる。ＣＲＴ画面に表示された図形はすべてフレーム０の図形とみなし操作される。

ＣＲＴ画面をクリアする時には、ＲＥＳＥＴステートメントを用いる。

（例）

RESET

二次記憶上のフレーム１～ｎをクリアするには

RESET 4

のようにフレームＩＤを指定する。クリアされた領域はフリーストレッジとしてシステムに返される。

2－2 割り込み処理

ＵＮＧＬでは、次の２つの形式で割り込み処理過程を表現することが可能である。

① ＯＮステートメントにより直接システムに、割り込み動作に関する指示を与える。

② あらかじめ、プログラム・ステートを定義しておき、プログラム・ステートの遷移と云

う形式でこれを表現する。

①は通常の割り込み処理の概念にあてはまるもので、ＯＮ文により、割り込み制御に関する情報を登録する方法である。

②では、直接割り込み制御表を意識させずに、これをプログラム・ステートと云う概念で扱う方法である。

特に、後者は複雑な割り込み処理過程を表現する時に、その流れをマクロに把握出来ると云う意味で、優れた表現形式を与えるだろう。

(1) プログラム・ステート

割り込み制御表の内容によって性格ずけられるプログラムの状態をＵＮＧＬではプログラム・ステートと呼ぶ。

プログラム・ステートはユーザによって定義され、その遷移のコントロールを受ける。

ユーザ・プログラムがイニシェートされた時プログラム・ステートは「状態０」にあると云う。

「状態０」は、システムが定義するプログラム・ステートで、その初期状態に於て、あらゆる割り込み要因に対してＤｉｓａｂｌｅ の状態にある。

プログラム・ステートは、スタティクに定義され、プログラムの実行に先だってシステムに通知されるが、実行時にその内容を変更することも可能である。前述した①の形式は「状態０」を動的に再定義すると云う形式で実現されることになる。

(2) プログラム・ステートの定義

プログラム・ステートはＤＥＦＩＮＥ 文からＥＮＤＳ文の間で定義され、ＳＴＡＴＥ文により識別番号（ステート番号）が与えられる。一つのプログラム・ステートはＳＴＡＴＥ文から、次のＳＴＡＴＥ文又はＥＮＤＳ文の間で定義され、割り込み要因とその処理ルーティンの関係及び処理後のプログラム・ステートの遷移に関する情報が与えられる。２つのプログラム・ステート 状態１、状態２を定義する例を以下に示す。

```
（例） ＤＥＦＩＮＥＳ                    ……プログラム・ステートの定
                                              義の開始を示す
       ＳＴＡＴＥ １                     ……状態１の定義の開始を示す
          ＯＮ ＬＰＥＮ ＤＯ ＳＵＢ１，２
          ＯＮ ＴＲＣＭ ＤＯ ＳＵＢ２，１
       ＳＴＡＴＥ ２                     ……状態２の定義の開始を示す
          ＯＮ ＦＫＥＹ ＤＯ ＳＵＢ３，１
```

ＥＮＤＳ　　……プログラム・ステートの定義の終りを示す。

ＯＮ文は次の形式で、割り込み要因と処理ルーティンの関係及び割り込み処理ルーティンから復帰した時のプログラム・ステートの遷移に関する情報を定義する。

ＯＮ（割り込み要因）　ＤＯ　（処理ルーティン名）〔，ｎ〕

例えば、先の例に於て、「状態１」ではライトペン割り込み（ＬＰＥＮ）とトラックマーク割り込み（ＴＲＣＭ）が可能である。ライトペン割り込みに対しては処理ルーティンＳＵＢ１を実行して復帰後の状態を「状態２」に置く。又トラックマーク割り込み（ＴＲＣＭ）に対してはＳＵＢ２を実行し復帰後の状態を「状態１」に置く。

ここで割り込み要因は、次の６種のキーワードによって指定される。

ＬＰＥＮ	ライトペン割り込み。（図形の指摘）
ＴＲＣＭ	トラッキング・クロスによる割り込み。
ＦＫＥＹ	ファンクションキーによる割り込み。
ＭＶＥＤ	アニメーション完了を通知する割り込み。
ＤＩＡＬ	ダイヤル割り込み。
ＭＥＳＧ	メッセージ割り込み。

(3)　プログラム・ステートの遷移

割り込み処理過程の推移はプログラム・ステートの遷移と云う形式で表現される。

プログラム・ステートの遷移を指示する方法は２通りある。１つは前述のプログラム・ステートの定義の中でＯＮ文のオプションとして復帰後のプログラム・ステートを指示する方法である。

もう１つは実行文ＴＲＡＮＳＦＥＲによる方法である。これは、プログラム・ステートの遷意が動的に決定される様な場合に特に必要とされる。

例えば次の様な例を考えてみよう。

（例）　「あるプログラム・ステートでファンクションキー割り込みが発生し、その処理ルティンに制御が渡った。この時キー番号が１の時は復帰後の状態を「状態４」にそれ以外の場合は「状態５」に置く。」

これはＴＲＡＮＳＦＥＲ文に用いて次の様に実現される。

```
      IF（IFKEY－１）    10,20,10
10    TRANSFER  5
          ⋮
```

```
20  TRANSFER  4
    :
    RETURN          (注) IFKEY にはファンクションキー番号が格納されて
                         いる。
```

この様にＴＲＡＮＳＦＥＲ文を使用することにより、プログラム・ステートを実行時に自由に遷移させる事が出来る。

ここでＴＲＡＮＳＦＥＲ文の性格として特に注意しておかなくてはならないことがある。それはＴＲＡＮＳＦＥＲが実行される時の状態によってその効果が次の様に変わると云う事である。

① 割り込み状態でＴＲＡＮＳＦＥＲが実行された時は、復帰後にその状態に遷移する。

② 非割り込み状態でＴＲＡＮＳＦＥＲが実行された時は、その時点で状態が遷移する。

(4) プログラム・ステートの動的な変更

プログラム・ステートに関する情報は実行に先だってスタティックに定義されるが、これを実行時に動的に修正することも可能である。

これは、特に、プログラム・ステートの遷移と云う形式をとらずに割り込み処理過程を表現する時には必要となる機能である。

① ＯＮ 文

ＯＮ文は次の形式で指定された、プログラム・ステートの内容に修正を加える。

ＯＮ （割り込み要因） ＤＯ（処理ルーティン名）〔,Ｎ〕〔,Ｍ〕

Ｍは対象とするプログラム・ステートのステート番号である。割り込み要因、処理ルーティン名、Ｎ等については、ステート定義文のＯＮ文と同様である。

（例） ＳＴＡＴＥ１ が次の様に定義されていたとする。

```
DEFINES
   STATE1
      ON LPEN DO SUB1,2
      ON FKEY DO SUB2,3
ENDS
```

この時 ＯＮ ＬＰＥＮ ＤＯ ＳＵＢ４，５を実行すると、ステート１の内容は修正を受け、次の様に定義されたのと同じ効果を持つ。

```
DEFINES
      STATE1
```

```
    ON LPEN DO SUB4,5
    ON FKEY DO SUB2,3
  ENDS
```

ON文でステート・オプションMが省略された時は、「状態0」への割り込み処理登録を意味する。

これは、通常のON文の動作にあたるもので、プログラム・ステートを定義しないで、割り込み処理過程を表現する手段と考えることが出来る。

② LOCK文 , UNLOCK 文

LOCK文、UNLOCK文は次の形式で指定されたプログラム・ステートの割り込み抑制（禁止、許容）情報を修正する。

LOCK （割り込み要因）〔,M〕

UNLOCK （割り込み要因）〔,M〕

Mは対象とするプログラム・ステートの番号を示す。省略時の動作はON文の時と同じである。

（例）

```
    LOCK FKEY , 3
```

プログラム・ステート3に於て、ファンクションキー割り込みを禁止する。

```
  UNLOCK FKEY , 3
```

プログラム・ステート3に於て、禁止されていたファンクションキー割り込みを受け付け可の状態にする。

(5) 割り込みの処理形式

割り込みは、CRTを介した、ユーザ・インタラクションにより、非同期に発生する。UNGLでは、この様にして、発生する割り込みを次の2つの処理形式で扱う事が出来る。

① 非同期処理：ユーザプログラムを非割り込み状態で走っている時に、非同期に発生する割り込みを処理する。この時、割り込み制御の仕方はその時のプログラム・ステートに従う。又、制御が渡される割り込み処理ルーティンはリエントラント構造を備えていなくではならない。これはシステムにより判断され（リエントラント・ルーティンか否かと云う情報は、そのプログラムの先頭に記すことに約束しておく）もしリエントラント構造を備えていない場合は、そのルーティンには制御が渡らないようにコントロールされる。

（この時、その割り込みはキャンセルされることになる。）

② 同期処理

割り込みが発生するまで、プログラムをＷＡＩＴ状態に置き、この状態で発生する割り込みを処理する。これには、割り込みが発生した時に、プログラム・ステートに従ってこれを制御する形式と、ＷＡＩＴ状態に入る時に、受けつけ可能な割り込み要因を指定する形式の２つの形式がある。後者の場合は、割り込みが発生すると、その要因が通知され、直ちに制御がもどされるからその後の割り込み処理はユーザが管理しなくてはならない。

1. ＡＷＡＩＴ 文

ＡＷＡＩＴ文が実行されると、プログラムは割り込み待ちの状態に入る。割り込みが発生すると、システムはその時の（ＡＷＡＩＴ文を実行する直前）プログラム・ステートの内容に従ってこれを制御する。

割り込み処理ルーティンでＲＥＴＵＲＮ文が実行されると再びもとのＡＷＡＩＴ状態に復帰する。

この状態を脱出するためには割り込み処理ルーティンでＡＥＸＩＴ文が実行されなくてはならない。又、割り込み処理ルーティンの中でＡＷＡＩＴ文を出す事は出来ない。割り込み処理ルーティンの中で、割り込み待ちの状態を作り出すには次に述べるＡＴＡＫＥ文を使用しなくてはならない。

2. ＡＴＡＫＥ 文

ＡＷＡＩＴ文と同様、プログラムを割り込み待ちの状態におく。ＡＷＡＩＴ文の場合は、割り込みはプログラム・ステートに従って制御されるが、ＡＴＡＫＥ文では一時的にプログラム・ステートを無視し、ＡＴＡＫＥ文のオペランドで指定された形式に従ってこれを制御する。割り込みが発生すると再び以前のプログラム・ステートを復活しＡＴＡＫＥ文以下に制御が渡る。

ＡＴＡＫＥ分の一般形式は次の様に与えられる。

ＡＴＡＫＥ　ｎ，（要因１，要因２，……要因ｉ）

要因ｉ……　要因ｉは割り込み要因を示すキーワードである。

（例１）

ＡＴＡＫＥ　ｎ，（ＬＰＥＮ　，　ＦＫＥＹ　，　ＴＲＣＭ）

を実行するとプログラムは、割り込み待ちの状態に入る。この時、ＬＰＥＮ，ＦＫＥＹ，ＴＲＣＭがそれぞれ割り込み可であり、各々の割り込みに対してｎに１，２，３がアサインされる。

（例２）

前の例で、ＬＰＥＮ，ＦＫＥＹ，ＴＲＣＭ 割り込みに対してそれぞれ、割り込み処理ルーテ

ィン　ＳＵＢＬ，ＳＵＢＫ，ＳＵＢＴを呼び出す。

```
          :
          :
          :
     ATAKE  n,(LPEN,FKEY,TRCM)
     GO  TO  (96,97,98),n
96   CALL     SUBL
     GO  TO     100
97   CALL     SUBF
     GO  TO     100
98   CALL     SUBT
100       :
          :
          :
```

(6)　割り込み情報の明細

割り込みが発生した時、割り込み情報の詳細を知るためには、次のＳＰＥＣＩＦＹ文を使用する。

ＳＰＥＣＩＦＹ 文は次の２つの形式で使用される。

（形式１）　ＳＰＥＣＩＦＹ　配列名

（形式２）　ＳＰＥＣＩＦＹ　要因名、配列名

（形式１）はあらかじめユーザが確保した配列のアドレスを登録しておく方法で、以後、割り込みが発生すると割り込み情報はすべてこの配列に通知される。

（形式２）は割り込み発生後に、要因名と受け取り場所を指定してその都度そこに通知を依頼する方法である。

ａ．割り込み処理プログラムの例

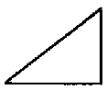
これまでに述べてきた割り込み処理機能が具体的にどの様な形で利用されるかを見る意味で、単純な割り込み処理プログラムの例を次に紹介する。

（例）

対象とする、インタラクティヴ・プロセスは次の様な過程からなる。

第１ステージ

○　トラッキング・マーク割り込みを待つ。



(0, 0)

○ トラッキング・マーク割り込みが入ったら、その位置情報を取り出し、定義しておいたイメージTRIANGLEの位置付け情報とする。

その位置にイメージTRIANGLEを表示し第2ステージに移る。

第2ステージ

○ トラッキング・マーク割り込みと、ライトペン割り込みを待つ。

○ トラッキング・マーク割り込みが入ったら、その位置に現在表示中のイメージTRIANGLEを移し、ステージ2を持続する。

○ ライトペン割り込みが入ったら、表示図形を消してステージ1にもどる。

この過程を次の3つの方法でプログラムを作成してみる。

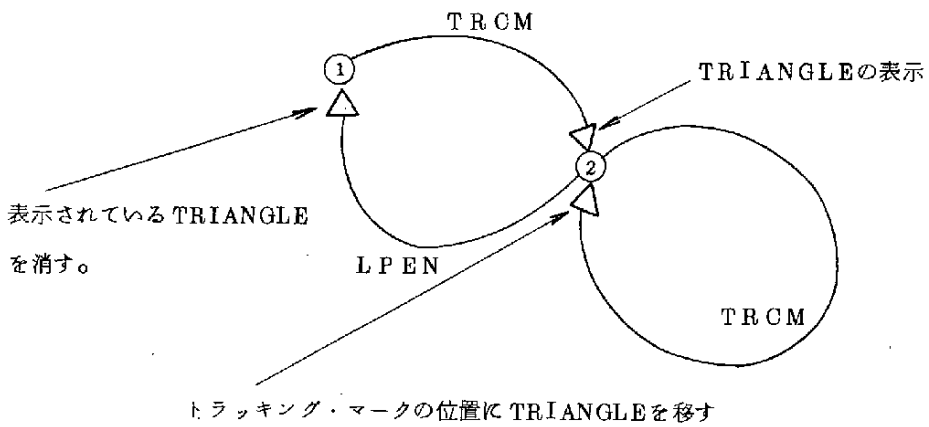
① プログラム・ステートの遷移で表現する方法

② ON文で表現する方法

③ ATAKE文で表現する方法

① プログラム・ステートによる表現

対象とする過程をステート・ダイヤグラムで表現すると次の様になる。



〔図2－23〕

以下にそのプログラム表現を記す。

```
DIMENSION  INF(20)
DEFINES
    STATE1
```

```
        ON TRCM DO SUBT , 2
    STATE2
        ON TRCM DO SUBT , 2
        ON LPEN DO SUBL , 1
 ENDS
      IMG<='LINE(0, 0, 2, 0, 2, 2, 0, 0)'
  TRIANG<=IMG·TRS·(X, Y)|<1>
 SPECIFY INF
 TRANSFER 1
 AWAIT
 STOP
 END
SUBROUTINE SUBT
 X = INF(2)
 Y = INF(3)
 DRAW 0, TRIANG
 RETURN
 END
SUBROUTINE SUBN
 DELETE 0, <1>
 RETURN
 END
```

(注) INFのCOMMON宣言は略してある。

② ON文で表現する方法

```
 DIMENSION INF(20)
  IMG<='LINE(0, 0, 2, 0, 2, 2, 0, 0)'
 TRIANG<=IMG·TRS·(X, Y)|<1>
 SPECIFY INF
 ON TRCM DO SUBT
 ON LPEN DO SUBL
```

```
    LOCK  LPEN
    AWAIT
    STOP
    END
  SUBROUTINE  SUBT
    X = INF (2)
    Y = INF (3)
    DRAW  0, TRIANG
    UNLOCK  LPEN
    RETURN
    END
  SUBROUTINE  SUBL
    DELETE  0, <1>
    LOCK  LPEN
    RETURN
    END
```

③ ATAKE文で表現する方法

```
      DIMENSION  INF (20)
      IMG = 'LINE (0, 0, 2, 0, 2, 2, 0, 0)
    TRIANG = IMG･TRS･ (X, Y) |<1>
      SPECIFY  INF
40  ATAKE  i, (TRCM)
    X = INF (2)
    Y = INF (3)
    DRAW  0, TRIANG
30  ATAKE  i, (TRCM, LPEN)
    GO TO (10, 20), i
10  X = INF (2)
    Y = INF (3)
    DRAW  0, TRIANG
```

```
      GO TO 30
20    DELETE  0, <1>
      GO TO 40
      STOP
      END
```

2－3　データ・ストラクチヤ操作

関連データの扱いに関しては、特に一般性を考慮してＬＥＡＰタイプの連想記憶機構を採用している。

言語機能も、ＳＥＴに関する機能が若干強化されていることを除いて大体ＬＥＡＰに準じている。

(1)　関連表現について

関連を表現してゆく上で基本となる、データ・タイプにアイテムとセットがある。

アイテムは、関連付けの対象となる最も基本的なデータ・タイプである。

アイテム間の関係は３つ組（オーダード・トリプル）を単位にして表現され、連想記憶に保存される。

一般に連想記憶に記憶された、関連３つ組に対しては、次の７つの形式の質問が可能である。（ＳＡＦと呼ばれる。）

ＳＡＦ（Simple Associative Form）

A・O≡V

A・O≡？

A・？≡V

A・？≡？

？・O≡V

？・？≡V

？・O≡？

これは、３つ組の順序性を考慮したすべての質問形式をつくしている。云いかえるならば連想記憶に表現された３つ組の関係は、構成要素のどれを手がかりにしても見出し得ると云う事が出来る。

この様に、関連表現に於て優れた力を持っていると云う事が、連想記憶に於ける１つの特徴で

あるが、さらに重要なことは、その記憶機構に対してユーザは全く気を配る必要が無いと云う事である。例えば従来のリング構造に於いては一般に「どのアイテムをどのアイテムのどのリングの何番目に挿入する」と云う様な内部構造を意識した関連の表現形式が必要とされたが、連想記憶機構では、単に関連の本質的な部分だけを述べれば十分である。

セットは、アイテムの集合を表現する。

アイテム間の関連は、単に個々のアイテムの間の関係として表現されるばかりでなくある性格のアイテムをグループ化したセットとして表現する事も出来る。

(2) アイテム

a，アイテムの表現形式

アイテムには、次の3つの表現形式がある。

① アイテム名標

アイテムに対してつけられたユニーク名をアイテム名標と呼ぶ。アイテムの最も基本的な表現形式である。（単にアイテムとも呼ぶ。）

② アイテム変数

アイテム変数は、アイテムの肩代わりをする変数的な役割りを果す。プログラミングはアイテム変数を利用して効果的に行うことが出来る。

③ ローカル変数

ローカル変数も、アイテム変数とほぼ同じ機能を果すが、後述する検索ステートメントＦＯＲＥＡＣＨ文のネストの中でのみ有効であると云う点が異る。

b，アイテムの宣言

アイテムは他のデータ・タイプと区別する意味で、宣言で与えなくてはならない。

＜例＞　ＰＯＩＮＴをアイテムとして宣言する。

```
ＩＴＥＭ　ＰＯＩＮＴ
```

c，アイテムのデータ

アイテムは、独自のデータを持つことが出来る。データ部の大きさは宣言時に与えられ、データの格納、検索は後述するファンクションを使用して行う事が出来る。

＜例＞　ＰＯＩＮＴをアイテムとして宣言し、そのデータ部として3語の領域を確保する。

```
ＩＴＥＭ　ＰＯＩＮＴ／３／
```

d，アイテムのアロケート

アイテムは次の2通りの方法でアロケートされる。

① 宣言されたアイテム

宣言で与えられたアイテムは実行に先立ってアロケートされる。

② 動的に生成するアイテム

ＡＬＬＯＣステートメントによりアロケートする。

ＡＬＬＯＣステートメントは実行時に外部から与えられた文字ストリングをアイテム名標として、アイテムをアロケートする。

＜例＞

ＡＬＬＯＣ　Ｚ，Ａ（１），４

Ｚはアイテム変数である。又

配列Ａには次の形式で文字ストリングが格納されている。

Ａ（１）	Ｐ　Ｏ　Ｉ　Ｎ	
Ａ（２）	Ｔ　１	

これは次の様にしたのと全く同じ効果を持つ。

ＩＴＥＭ　　ＰＯＩＮＴ１／４／

Ｚ　＝　ＰＯＩＮＴ１

アイテムはアロケートされると、アイテム・テーブルに登録される。アイテム・テーブルはアイテム名標とそのインターナルネームの対応を与えるものである。インターナルネームはアイテムの内部的な呼び名であり、アイテムに関する内部的な操作はすべてインターナルネームで行われることになる。

又アイテムのデータ部は、データ・テーブルに確保され、これはアイテムのインターナルネームから連想する事が出来る様になっている。

ｅ，アイテムの消去

アイテムの消去はＦＲＥＥ文で行われる。

ＦＲＥＥ　ＩＴＥＭ

を実行すると、そのアイテムはアイテム・テーブルから取り除かれ、データ部はフリーストレージに返却される。

(3) トリプル

アイテム間の関係は、３つ組（オーダード・トリプル）を単位として表現する。

３つ組を連想記憶に格納する命令にＭＡＫＥ文がある。

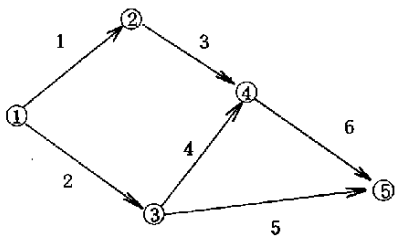
ＭＡＫＥ　《ＩＴＥＭ１，ＩＴＥＭ２，ＩＴＥＭ３》

ＩＴＥＭ１、ＩＴＥＭ２、ＩＴＥＭ３はそれぞれアイテムである。

関連表現の方法は全く任意である。要するに３つ組の集合として表現された関連にＳＡＦ形式の検索が可能であると云う事を念頭に於て後の操作に都合の良い様に表現すればよいわけである。

関連表現の例として次の様な例を考えてみよう。

＜例＞　次の図のようなネットワークに対して、３つ組を用いた関連表現の例をあげる。



〔図２－２４〕

〔表現１〕

各アクティビティをアイテムとし、さらにＳＵＣという後続アクティビティの意味をもつアイテムを設け、各アクティビティの後続アクティビティがどのアクティビティであるかという関連表現をする。

ＭＡＫＥ　《ＳＵＣ，ＡＣＴ１，ＡＣＴ３》

ＭＡＫＥ　《ＳＵＣ，ＡＣＴ２，ＡＣＴ４》

ＭＡＫＥ　《ＳＵＣ，ＡＣＴ２，ＡＣＴ５》

ＭＡＫＥ　《ＳＵＣ，ＡＣＴ３，ＡＣＴ６》

〔表現２〕

各ノード、各アクティビティをアイテムとし、各アクティビティと前後のノードに順序性をもたせ関連表現をする。

ＭＡＫＥ　《ＮＯＤＥ１，ＡＣＴ１，ＮＯＤＥ２》

ＭＡＫＥ　《ＮＯＤＥ２，ＡＣＴ３，ＮＯＤＥ４》

ＭＡＫＥ　《ＮＯＤＥ１，ＡＣＴ２，ＮＯＤＥ３》

ＭＡＫＥ　《ＮＯＤＥ３，ＡＣＴ４，ＮＯＤＥ４》

(4)　セット

セットはアイテムの集合を表わす。

セットにはいくつかの表現形式があるが最も基本的なものはセット名標である。セット名標

で表わされるセットは、セット・テーブルに実体を持つ。即ちセット・テーブルに実際にそのメンバーとするアイテムの集合を蓄えている。

この他のセットの表現形式として、集合演算子とセットによるセットの表現形式、アイテムとアイテムを関連演算子で結んだセットの表現形式等があるが、これ等はセット・テーブルには実体を持たない。そのエクスプレッションを評価した結果、システムの作業域に一時的に作り出されるだけである。即ちプログラムの実行過程で必要な時点で内部に創成され、必要がなくなると消滅する。

これ等のすべてのセットの表現形式を総称してセット・エクスプレッションと呼ぶ。

a　セット名標の宣言

セット名標は宣言で与えなくてはならない。その構成規則はアイテムと同様である。

＜例＞　ＳＥＴ　ＪＡＰＡＮ，ＧＥＲＭＡＮＹ

ＪＡＰＡＮ，ＧＥＲＭＡＮＹはそれぞれセット名標である。

b　セット・エクスプレッションについて

セット名標以外に、多くのセットの表現形式があるが、ここではその主なものについてのみ述べる。詳しくは文法の項を参照されたい。

①　集合演算子による表現

セット間の集合演算を定義する３つの演算子がある。

.－.　差　集　合

.＋.　和　集　合

.＊.　積　集　合

（演算子の間に優先順位はない。左から右へ評価される）

セットはこれ等の演算子による一つのエクスプレッションとして表現される。

＜例＞　ＳＥＴ１ .－. ＳＥＴ２

は１つのセットを表わしその内容はＳＥＴ１、ＳＥＴ２の差集合を表わしている。

②　関連演算子によるセット表現

関連演算子「・」、「，」はアイテム間で定義され次の条件を満たすようなアイテムの集合を表わす。

ＩＴＥＭ１・ＩＴＥＭ２　は《ＩＴＥＭ１，ＩＴＥＭ２，？》を満足する様なアイテムの集合。

ＩＴＥＭ１，ＩＴＥＭ２　は《ＩＴＥＭ１，？，ＩＴＥＭ２》を満足する様なアイテムの集合。

c　セットの生成と消去

宣言されたセット名標は、初期状態に於て空集合の状態にある。

セットにメンバーをつけ加えたり、取り除いたりするためには次に述べるＰＵＴステートメント、ＲＥＭＯＶＥステートメントを使用する。

① ＰＵＴステートメント

ＰＵＴステートメントは、指定されたセットにアイテム又はアイテムの集合をメンバーとしてつけ加える働きをする。一般形式は次の形をとる。

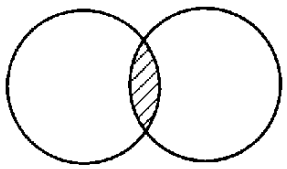
ＰＵＴ { アイテム / セット・エクスプレッション } ＩＮ セット名標

＜例＞ 次の例はセットＦＡＭＩＬＹにアイテムＴＯＭをメンバーとしてつけ加える例である。

```
PUT  TOM  IN  FAMILY
```

＜例２＞ 次の例はセットＣＬＡＳＳ１とセットＣＬＡＳＳ２の積集合をセットＣＬＡＳＳのメンバーとしてつけ加える例である。

ＣＬＡＳＳ１　　ＣＬＡＳＳ２



〔図２－２５〕

ＣＬＡＳＳ

```
PUT  CLASS1  *  CLASS2  IN  CLASS
```

② ＲＥＭＯＶＥステートメント

ＲＥＭＯＶＥステートメントはＰＵＴステートメントの逆の操作を意味する。即ち、指定されたセットから、アイテム又はアイテムの集合を取り除く働きをする。

一般形式は次の通りである。

ＲＥＭＯＶＥ { アイテム / セット・エクスプレッション } ＦＲＯＭ セット名標

＜例＞ セットＣＬＡＳＳのメンバーから、アイテムＪＩＭを取り除く

```
REMOVE  JIM  FROM  CLASS
```

(5) ファンクション

アイテム、セットに対して適用される各種のファンクションが用意されている。これ等は値として整数値又は論理値をとるものがあるが、その扱いは通常のＦＯＲＴＲＡＮのファン

クションと全く同じである。ここではその主なものについての説明にとどめる。詳しくは仕様の項を参照されたい。

a　アイテムに適用されるファンクション

①　ＩＳＴＲＰＬ

次の形式で使用され、連想記憶にトリプルが存在するか否かを調べる。

ＩＳＴＲＰＬ（ＩＴＥＭ１，ＩＴＥＭ２，ＩＴＥＭ３）

ＩＴＥＭ１ …… ＩＴＥＭ３はそれぞれアイテムである。もし、《ＩＴＥＭ１，ＩＴＥＭ２，ＩＴＥＭ３》のトリプルが存在すればＴＲＵＥ、存在しなければＦＡＬＳＥをそれぞれ関数値として返す。

②　ＩＤＡＴＡ

指定変数としても使用され、次の形式で、アイテムのデータを検索したり格納したりするために用いる。

ＩＤＡＴＡ（ＩＴＥＭ１，ｎ）

ＩＴＥＭ１は、アイテム、ｎは整定数又は整変数である。関数値は整数値をとる。

＜例１＞　アイテムのデータを検索する例

ＬＥＮＧＳ＝ＩＤＡＴＡ（ＴＲＥＥ，２）

ＬＥＮＧＳ（整変数）には、アイテムＴＲＥＥのデータ部の２番目のワードのデータが代入される。

＜例２＞　アイテムのデータ部にデータを格納する例

ＩＤＡＴＡ（ＴＲＥＥ，１）＝２＊Ｋ＋４

アイテムＴＲＥＥのデータ部の一番目のワードに２＊Ｋ＋４で評価される値を格納する。

b　セットに適用されるファンクション

①　ＩＳＳＢＳＴ

次の形式で使用され、指定されたセットが、指定されたセットのサブセットになっているか否かを調べる。

ＩＳＳＢＳＴ（ＳＥＴ１，ＳＥＴ２）

ＳＥＴ１、ＳＥＴ２はそれぞれセット・エクスプレッションである。もしＳＥＴ１がＳＥＴ２のサブセットであるならばＴＲＵＥそうでなければＦＡＬＳＥがそれぞれ関数値となる。

②　ＩＳＭＢＥＲ

次の形式で使用され、指定されたアイテムが指定されたセットのメンバーになっているか

否かを調べる。

ＩＳＭＢＥＲ（ＩＴＥＭ１，ＳＥＴ１）

ITEM1はアイテム、ＳＥＴ１は、セット・エクスプレッションである。

もしＩＴＥＭ１がＳＥＴ１のメンバーであるならばＴＲＵＥそうでなければＦＡＬＳＥがそれぞれ関数値として返される。

その他、セットのメンバーの数を調べるＩＳＮＭＢＲ、セット同志が等しいかどうかを調べるＩＳＥＱＳＴ等があるがこれ等については文法の項を参照されたい。

(6) 検索ステートメント

これまでに、アイテム間の関係が、トリプルあるいはセットと云う形で表現出来る事を述べてきた。

ここでは、この様にして表現された関連構造の中から必要とするアイテムをどの様な形で取り出すかという点について述べる。

一般に、アイテムの検索は、単にある条件を満たすアイテムを取り出すと云う形だけで終るものではない。通常取り出したアイテムを見るとかそれに何か操作を加えると云う必要性が必ずあるわけである。従って検索の形式としてはある条件を満たすアイテムに対して必要な操作が加えられる様な表現形式が便利である。

次に述べるＦＯＲＥＡＣＨ文は、こうした要望によくかなうものである。

FOR EACH 文の形式には次の２つの形式がある。

① ＦＯＲＥＡＣＨ　ローカル変数　ＩＮ　セット・エクスプレッション　ＤＯ　n

　プロセデュア　n　ＣＯＮＴＩＮＵＥ

nは端末文のステートメント番号である。

セット・エクスプレッションで評価されるセットの構成要素を１つづつローカル変数に割り当て１回ずつすべてのメンバーについてプロセデュアを実行する。プロセデュアは、求めたアイテムに対する操作を含む処理過程でFORTRAN ステートメントが記される。

<例>　　Ｉ＝０

　　ＦＯＲＥＡＣＨ　Ｚ　ＩＮ　ＳＯＮＳ　ＤＯ　１

　　Ｉ＝Ｉ＋ＩＤＡＴＡ（Ｚ，１）

　１　ＣＯＮＴＩＮＵＥ

上の例ではセットＳＯＮＳ　のメンバーがＺに１つ代入されるごとに次の代入文を実行するこの様にしてすべてのメンバーについてこの過程が実行され終るとCONTINUE 以下に制

御が渡る。この時Ｉにはセット SONSを構成するすべてのアイテムのデータの総和が求まることになる。

② FOREACH CONTEXT・SPEC DO n

プロセデュア

n CONTINUE

CONTEXT・SPEC としたところはオーダド・トリプルの１つ又は２つの構成要素がローカル変数で置き変えられたトリプルの形式が記され、アイテムの集合を表わす。

例えばZをローカル変数とすると《FRIEND,X,TOM》 Xに該当するいくつかのアイテムの集合が求まる。その集合に対してプロセデュアを一回づつくり返すことになる。

COTEXT・SPEC はさらに拡張された表現を認めているが詳しくは仕様の項を参照されたい。

<例>

FOREACH (COMP,FACE1,X) DO 1

1 PUT X IN WORK

COMP,FACE1はそれぞれアイテム、Xはローカル変数、WORKはセットである。

COMP・FACE1 = ? を満すアイテムを一つづつXに代入し一回ずつ次のＰＵＴ文を実行する。結果的にその集合がセットWORKのメンバーになる。

第３節　UNGLの仕様

３－１.　図形データ操作

(1)　グラフィック・データ

ＵＮＧＬでは、グラフィック・データ として、新たなデータタイプのイメージ、イメージ・データ、変換マトリックス、論理化因子を設ける。

ａ．イメージ

イメージは図形のまとまりを表わし、２次元イメージと３次元イメージがある。イメージは宣言文によって宣言される。又これらのＩＤはＦＯＲＴＲＡＮ変数構成規則に従う。

＜イメージＩＤ＞：：＝＜イメージ２ＩＤ＞｜＜イメージ３ＩＤ＞

ｂ．イメージ・データ

イメージ・データはイメージの実データを定義するもので、グラフィック・ファンクションで表現される。イメージ・データには、２次元イメージ、３次元イメージに対応する２次元イメージ・データ、３次元イメージ・データがある。

＜イメージ・データ＞：：＝＜イメージ２データ＞｜＜イメージ３データ＞

＜イメージｒデータ＞：：＝'＜イメージｒデータリスト＞'

＜イメージｒデータリスト＞：：＝＜ｒグラフィック・ファンクション＞

｜＜イメージｒデータリスト＞，＜ｒグラフィック・ファンクション＞

＜２グラフィック・ファンクション＞：：＝＜ＬＩＮＥ２ファンクション＞｜＜ＴＥＸＴファンクション＞

＜３グラフィック・ファンクション＞：：＝＜ＬＩＮＥ３ファンクッション＞

グラフィック・ファンクションに関して次に説明する。

①　２次元グラフィック・ファンクションにはＬＩＮＥとＴＥＸＴがあり、一般形は次のとおりである。

○　ＬＩＮＥ２（X_1，Y_1，X_2，Y_2，……X_N，Y_N）

Ｘｉ，Ｙｉ　：　点の座標値　（実数、実変数）

（X_1，Y_2）、（X_2，Y_2）　……　（X_N，Y_N）を結ぶ線を表わす。

○　ＬＩＮＥ２（Ｘ，Ｙ，Ｉ，Ｊ，Ｋ）

Ｘ，Ｙ　：　点のｘ座標、ｙ座標の値が格納されている配列の配列名

Ｉ，Ｊ，Ｋ：　配列のＩ番目からＪおきにＫ番目までのｘ座標、ｙ座標をもつ点を結ぶ線を表わす。　（整数、整変数）

(X(I), Y(I)), (X(I+J), Y(I+J)), (X(I+2J), Y(I+2J)), ……
…… (X(K'), Y(K')) を結ぶ線を表わす。

ここでK' はK−J<K'≦Kをみたす値をとる。

○ TEXT(Xo, Yo, MOJI)

Xo, Yo : 文字を描き始める点 (実数、実変数)

N : 文字数を表わし、正の時は大文字
負の時は小文字となる。(整数、整変数)

MOJI : 描く文字列を表わす。(ホラリスコンスタント、配列名)

(Xo, Yo) の点から指定された文字列を描く、文字は常にX軸と平行に描かれる。

② 3次元グラフィック・ファンクションにはLINEがあり、一般形は次のとおりである。

○ LINE3(X_1, Y_1, Z_1, X_2, Y_2, Z_2, …… X_N, Y_N, Z_N)

Xi, Yi, Zi : 点の座標値(実数、実変数)

(X_1, Y_1, Z_1), (X_2, Y_2, Z_2), …… (X_N, Y_N, Z_N)を結ぶ直線を表わす。

○ LINE3(X, Y, Z, I, J, K)

X, Y, Z : 点のx座標、y座標、z座標の値が格納されている配列の配列名

I, J, K : 配列のI番目からJおきにK番目までの点を指定する。(整数、整変数)

(X(I), Y(I), Z(I)), (X(I+J), Y(I+J), Z(I+J)) ……
…… (X(K'), Y(K'), Z(K')), を結ぶ線を表わす。

ここで

K' はK−J<K'≦Kをみたす値をとる。

c. 変換マトリックス

変換マトリックスは、イメージに対して座標変換、透視変換、平行投影を施すためのデータを与えるグラフィック・データである。

<変換マトリックス>::=<座標変換2マトリックス>
| <座標変換3マトリックス>
| <透視変換マトリックス>
| <平行投影マトリックス>

① 座標変換２マトリックス

一般形

（Xo，Yo，θ，Sx，Sy）

Xo ： x方向の平行移動量

Yo ： y方向 〃

θ ： 原点中心に反時計まわりの回転角度

Sx ： x方向のスケールファクタ

Sy ： y方向 〃

意味

２次元イメージをx方向にSx、y方向にSy倍し、原点中心にθ回転した後に（Xo，Yo）にポジショニングをするためのデータである。

② 座標変換３マトリックス

一般形

（Xo，Yo，Zo，θxy，θz，Sx，Sy，Sz）

Xo ： x方向の平行移動量（実数値）

Yo ： y方向 〃 （ 〃 ）

Zo ： z方向 〃 （ 〃 ）

θxy ： xy平面と平行な面で原点中心に反時計まわりの回転角度（実数値）

θz ： さらにxy平面と平行な面からZ軸の正方向への回転角度（実数値）

Sx ： x方向のスケールファクタ （実数値）

Sy ： y方向 〃 （ 〃 ）

Sz ： z方向 〃 （ 〃 ）

意味

３次元イメージをx方向にSx，y方向にSy，z 方向にSz 倍し、原点中心にxy平面と平行にθxy 回転させ、さらにxy 平面と平行な面からZ軸の正方向への角度θz だけ回転させる。そして（Xo，Yo，Zo） にポジショニングをする。これらの座標変換を行うためのデータである。

③ 透視変換マトリックス

一般形

（Xo，Yo，Zo，F）

Xo ： 視点のx座標　（実数値）

Yo ：　〃　y座標　（　〃　）

Zo ：　〃　z座標　（　〃　）

F　：＝1のとき　xy平面を投影面とする。

＝2　　yz平面　〃

＝3　　zx平面　〃

意　味

3次元イメージに対し、視点（Xo，Yo，Zo）からFで指定された投影面に透視するためのデータである。

④　平行投影マトリックス

一般形

（F）

F：＝1のとき xy 平面を投影面とする。

＝2　　yz 平面　〃

＝3　　zx 平面　〃

意　味

3次元イメージに対し、Fで指定された投影面に平行投影するためのデータである。

d．論理化因子

いくつかのイメージを論理的にグルーピングをし、それに対する識別名、CRT画面に表示される時の制御情報を与えるグラフィック・データである。

＜論理化因子＞：：＝＜＜論理名＞，＜制御情報＞＞

＜論理名＞：：＝＜アイテムID＞｜＜アイテム変数ID＞｜＜整定数＞

｜＜整変数＞｜＜ホラリスコンスタント＞

＜制御情報＞：：＝－3｜－2｜－1｜0｜1｜2｜3｜＜整変数＞

ここで制御情報は次のように決められている。

＜制御情報＞：　＝－3のとき　高輝度でライトペンピック不可

＝－2　中輝度　〃

＝－1　低輝度　〃

＝　0　ブランク

＝　1　低輝度でライトペンピック可

＝２　　中輝度でライトペンピック可

＝３　　高輝度　　〃

尚デフォルト値は「２」で高輝度、ライトペンピック可とする。

(2)　イメージ・エクスプレション

イメージの構造化、イメージに対する幾何学操作は、イメージ・エクスプレションという形式で定義される。

次にイメージ・エクスプレションをＢＮＦで記述する。

＜イメージγプライマリ＞：：＝＜イメージγＩＤ＞｜＜イメージγデータ＞

｜＜イメージγエクスプレション＞

＜イメージγファクタ＞　：：＝＜イメージγプライマリ＞

｜＜イメージγファクタ＞＋＜イメージγプライマリ＞

＜イメージ２ターム＞　：：＝＜イメージ２ファクタ＞

｜＜イメージ２ターム＞．ＴＲＳ．＜座標変換２マトリックス＞

｜＜イメージ３ターム＞．ＰＥＲＳ．＜透視変換マトリックス＞

｜＜イメージ３ターム＞．ＯＲＴＨ．＜平行投影マトリックス＞

＜イメージ３ターム＞：：＝＜イメージ３ファクタ＞

｜＜イメージ３ターム＞．ＴＲＳ．＜座標変換３マトリックス＞

＜イメージγエクスプレション＞：：＝＜イメージγターム＞

｜＜イメージγエクスプレション＞｜＜論理化因子＞

(3)　グラフィック・オペレータ

イメージに対して幾何学的操作を行うためにグラフィック・オペレータがある。

＜グラフィックオペレータ＞：：＝＜＝｜＋｜－｜．ＴＲＳ．｜．ＰＥＲＳ．｜．ＯＲＴＨ．

それぞれ次のような機能をもつ。

オペレータ	機　　能
<=	イメージを定義する。
+	イメージのインクルージョンを表わす。
/	論理化因子の結合
.TRS.	イメージの平行移動、拡大・縮小、回転を表わす
.PERS.	イメージの透視変換を表わす
.ORTH.	イメージの平行投影を表わす

(4) グラフィックステートメント

① イメージ宣言ステートメント

o IMAGE ステートメント

構 成

<IMAGEステートメント>::=IMAGE*2<イメージIDリスト>

/IMAGE*3<イメージIDリスト>

<イメージIDリスト>::=<イメージID>/<イメージID>,<イメージIDリスト>

機 能

IMAGE*2, IMAGE*3はそれぞれ<イメージIDリスト>で指定されたグラフィックデータが2次元イメージ、3次元イメージであることを宣言している。

② イメージ代入ステートメント

構 成

<イメージ代入ステートメント>::=<イメージγID><=<イメージγエクスプレション>

機 能

<イメージγエクスプレション>で表現されたイメージを<イメージγID>で指定されたイメージとして定義する。

(5) グラフィック入出力ステートメント

イメージをディスプレイ画面、あるいは二次記憶上のフレームに出力し、さらにそのイメージに操作を加えると共に、フレーム操作を行うためのステートメントである。

① DRAW ステートメント

構 成

＜DRAWステートメント＞：：＝DRAW＜フレーム番号＞，＜イメージ２ＩＤリスト＞

（＜ウインドイング・オプション＞）

｜DRAW＜フレーム番号＞，＜イメージ２ＩＤリスト＞

＜フレーム番号＞：：＝０｜１｜２｜……｜９９

＜イメージ２ＩＤリスト＞：：＝＜イメージ２ＩＤ＞｜＜イメージ２ＩＤ＞，

＜イメージ２ＩＤリスト＞

＜ウインドイング・オプション＞：：＝＜イメージ領域＞，＜フレーム領域＞

｜＜イメージ領域＞

＜イメージ領域＞：：＝＜始点のx座標＞，＜始点のｙ座標＞，＜△x＞，＜△ｙ＞

＜フレーム領域＞：：＝＜始点のＸ座標＞，＜始点のＹ座標＞，＜△Ｘ＞，＜△Ｙ＞

機 能

＜フレーム番号＞で指定されたフレームに＜イメージ２ＩＤリスト＞で指定されたイメージを出力する。フレームが０のときはＣＲＴ画面、その他は二次記憶上のフレームを示す。そのときイメージ系の座標系において＜始点のx座標＞から＜△x＞，＜始点のy座標＞から＜△y＞で囲まれた四辺形の領域内のイメージを、フレーム系の座標系＜始点のＸ座標＞から＜△Ｘ＞、＜始点のＹ座標＞から＜△Ｙ）で囲まれた四辺形の領域にスケーリングを施して出力する。尚イメージ領域の指定は実数値で、フレーム領域の指定は整数値で行う。イメージ領域、フレーム領域のデフォルト値はそれぞれ（0.，0.，1024.，1024.）（0，0，1024，1024）とする。

② ＡＮＩＭＡＴＥ ステートメント

構 成

＜ＡＮＩＭＡＴＥ ステートメント＞：：＝ＡＮＩＭＡＴＥ＜イメージ２ＩＤリスト＞

（＜ムーヴィング情報＞）

｜ＡＮＩＭＡＴＥ＜イメージ２ＩＤリスト＞

（＜ムーヴィング情報＞）

（＜ウインドイング・オプション＞）

＜ムーヴィング情報＞：：＝＜△x'＞，＜△ｙ′＞，＜回数＞

機 能

フレーム０ （ＣＲＴ画面）にのみ出力が可能であり、＜イメージ２ＩＤリスト＞で指定さ

れたイメージを　DRAW ステートメントと同様にウインドイングを施しx方向のきざみ幅<△x′>、y方向のきざみ幅<△y′>で、<回数>で指定された回数をムーヴィングする。

③　DELETE ステートメント

構　成

<DELETE　ステートメント>：：＝DELETE<フレーム番号>，<論理化因子>

機　能

<フレーム番号>で指定されたフレーム内の<論理化因子>で指定されたフラグを持つ図形を消去する。

④　CONTROL　ステートメント

構　成

<CONTROL　ステートメント>：：＝CONTROL　<フレーム番号>，<論理化因子>

機　能

<フレーム番号>で指定されたフレーム内の<論理化因子>で指定されたフラグの制御情報をこの<論理化因子>で指定した制御情報に変更する。

⑤　DISPLAY　ステートメント

構　成

<DISPLAY　ステートメント>：：＝DISPLAY<フレーム番号>

機　能

<フレーム番号>で指定された二次記憶上のフレームをCRT画面に表示する。従ってフレーム0を指定しても意味はない。

⑥　RESET　ステートメント

構　成

<RESET　ステートメント>：：＝RESET<フレーム番号>

機　能

<フレーム番号>で指定されたフレームをクリアする。フレーム0の時はCRT画面が消え、他の時には二次記憶上のフレーム領域をフリー・ストレイジに返す。

3－2.　割り込み処理

(1)　割り込み要因

UNGL では、割り込み要因を示す6種のキーワードを設けている。

<割り込み要因名>：：＝ＬＰＥＮ｜ＴＲＣＭ｜ＦＫＥＹ｜ＭＶＥＤ｜ＤＩＡＬ｜ＭＥＳＧ

以下にキーワードと割り込み要因の対応関係を示す。

割り込み要因名	割　り　込　み
ＬＰＥＮ	ライトペンによる割り込み
ＴＲＣＭ	トラッキング・クロスによる割り込み
ＦＫＥＹ	ファンクションキーによる割り込み
ＭＶＥＤ	アニメーション完了を通知する割り込み
ＤＩＡＬ	ダイヤルによる割り込み
ＭＥＳＧ	メッセージによる割り込み

(2)　ステートメント

割り込み処理のために次のようなステートメントが用意されている。

ａ．プログラム・ステート定義文

プログラム・ステートはＤＥＦＩＮＥＳ文からＥＮＤＳ文の間で定義されＳＴＡＴＥ文によりステート番号が与えられる。一つのプログラム・ステートはＳＴＡＴＥ文から次のＳＴＡＴＥ文、又はＥＮＤＳ文の間で定義され、割り込み要因とその処理ルーチンの関係、及び処理後のプログラム・ステートの遷移に関する情報をＯＮ文で与える。

①　ＤＥＦＩＮＥＳステートメント

構　成

<ＤＥＦＩＮＥＳステートメント>：：＝ＤＥＦＩＮＥＳ

機　能

プログラム・ステートの定義の開始を示す。

②　ＥＮＤＳ　ステートメント

構　成

<ＥＮＤＳステートメント>：：＝ＥＮＤＳ

機　能

プログラム・ステートの定義の終りを示す。

③　ＳＴＡＴＥ　ステートメント

構 成

＜STATEステートメント＞：：＝STATE＜ステート番号＞

＜ステート番号＞：：＝＜正整数＞

機 能

DEFINES ステートメントからENDSステートメントの間でのみ意味をもち、＜ステート番号＞で与えられたプログラム・ステートの定義の開始を示す。

④ ONステートメント

構 成

＜ONステートメント＞：：＝ON＜割り込み要因名＞DO＜処理ルーチン名＞，＜ステート番号＞

機 能

DEFINESステートメントからENDSステートメントの間で定義され、＜割り込み要因名＞で指定された割り込み要因の割り込み処理登録をし、さらに割り込み処理ルーチンから復帰した時は＜ステート番号＞で指定されたプログラム・ステートとなることを定義する。尚＜ステート番号＞が省略された時は元のプログラム・ステートのままである。

b. プログラム・ステートの遷移文

① TRANSFER ステートメント

構 成

＜TRANSFER ステートメント＞：：＝TRANSFER ＜ステート番号＞

機 能

＜ステート番号＞で指定されたプログラムステートに遷移させる。

c. プログラムステートの変更文

① ONステートメント

構 成

＜ONステートメント＞：：＝ON＜割り込み要因名＞DO＜処理ルーチン名＞，＜ステート番号－1＞，＜ステート番号－2＞

機 能

＜ステート番号－1＞で指定されたプログラム・ステート内の＜割り込み要因名＞で指定された割り込み要因に対する処理ルーチンを変更する。又はその割り込み要因が定義されていない時は新たに、割り込み処理登録をする。 ＜ステート番号－2＞はa.－④のONステート

メントと同様である。又＜ステート番号－１＞が省略された時はステート０の割り込み処理登録をする。

② ＬＯＣＫステートメント

構 成

＜ＬＯＣＫステートメント＞：：＝ＬＯＣＫ＜割り込み要因名＞，＜ステート番号＞：
ＬＯＣＫ＜割り込み要因名＞

機 能

＜ステート番号＞で指定されたプログラムステート内の＜割り込み要因名＞で指定された割り込みの禁止をする。＜ステート番号＞が省略された時はステート０を意味する。

③ ＵＮＬＯＣＫ ステートメント

構 成

＜ＵＮＬＯＣＫステートメント＞：：＝ＵＮＬＯＣＫ＜割り込み要因名＞，＜ステート番号＞
ＵＮＬＯＣＫ＜割り込み要因名＞

機 能

＜ステート番号＞で指定されたプログラムステート内の＜割り込み要因名＞で指定された割り込みの禁止を解除する。＜ステート番号＞が省略された時はステート０を意味する。

ｄ．同期処理に関するステートメント

① ＡＷＡＩＴ ステートメント

構 成

＜ＡＷＡＩＴステートメント＞：：＝ＡＷＡＩＴ

機 能

割り込みが発生するまでプログラムをＷＡＩＴ状態に置く。

② ＡＴＡＫＥ ステートメント

構 成

＜ＡＴＡＫＥ ステートメント＞：：＝ＡＴＡＫＥ＜整変数＞，（＜割り込み要因リスト＞）
＜割り込み要因リスト＞：：＝＜割り込み要因名＞｜＜割り込み要因名＞，＜割り込み要因リスト＞

機 能

ＡＴＡＫＥステートメントが実行されるとプログラムは割り込み待ちの状態に入る。この

時、＜割り込み要因リスト＞で指定された割り込みが可能となり、割り込みが発生すると、その割り込み要因が＜割り込み要因名＞の何番目にリストされた要因であるかを＜整変数＞で指定された変数にアサインし、ＡＴＡＫＥに続く文に制御を移す。

③ ＡＥＸＩＴ ステートメント

構 成

＜ＡＥＸＩＴステートメント＞：：＝ＡＥＸＩＴ

機 能

割り込みを解除し、ＡＷＡＩＴで指定されたＷＡＩＴ状態からぬけ出す。

３－３ データ・ストラクチャ操作

(1) 関連データ

ＵＮＧＬでは関連データとして、新たなデータ・タイプのアイテム、アイテム変数、ローカル変数、セットを設ける。

a．アイテム・エクスプレション

アイテムは関連づけの基本要素であり、次のような表現形式がある。アイテムはそれぞれ宣言文によって宣言される。又これらのＩＤはＦＯＲＴＲＡＮの変数構成規則に従う。

① アイテムＩＤ

アイテムのユニーク名でインターナルネームをもつ。

② アイテム変数ＩＤ

アイテム変数は検索の時などにアイテムの肩代りをする。

③ ローカル変数ＩＤ

ローカル変数は ＦＯＲＥＡＣＨ ステートメント内で使用されるアイテム変数である。

④ アレイアイテムＩＤ

配列をなすアイテムで、ＩＤはＦＯＲＴＲＡＮの配列構成規則に従う。

これらの関係をＢＮＦで記述すると次のようになる。

＜アイテム・エクスプレション＞：：＝＜アイテムＩＤ＞｜＜アイテム変数ＩＤ＞｜＜ローカル変数ＩＤ＞｜＜アレイアイテムＩＤ＞

b セット・エクスプレション

セットは、順序性をもたないアイテムの集合であり、次のような表現形式がある。

① 宣言されたセットＩＤ

セットＩＤはＦＯＲＴＲＡＮの変数構成規則に従う。

② 予約語 ＡＬＬ

アイテム・エクスプレションのリスト

④ 関連演算子により表現されるセット

⑤ セット間の和集合、差集合、積集合のセット

これらの関連をＢＮＦで記述すると次のようになる。

＜セット・プライマリ＞：：＝＜セットＩＤ＞｜ＡＬＬ｜＜アイテム・エクスプレション・リスト＞
｜＜ディライブド・セット＞

＜アイテム・エクスプレション・リスト＞：：＝＜アイテム・エクスプレション＞
｜＜アイテム・エクスプレション＞，
＜アイテム・エクスプレション・リスト＞

＜ディライブド・セット＞：：＝＜アイアム・エクスプレション＞＜関連演算子＞
＜アイテム・エクスプレション＞

＜関連演算子＞：：＝　　・｜，

＜セット・ファクタ＞：：＝＜セット・プライマリ＞｜＜セット・ファクタ＞．－．
＜セット・プライマリ＞

＜セット・ターム＞：：＝＜セット・ファクタ＞｜＜セット・ターム＞．＋．＜セット・ファクタ＞

＜セット・エクスプレション＞：：＝＜セット・ターム＞｜＜セット・エクスプレション＞．＊．
＜セット・ターム＞

ｃ．トリプル

トリプルはアイテムの関連を表現する。

＜トリプル＞：：＝《＜アイテム・エクスプレション＞，＜アイアム・エクスプレション＞；
＜アイテム・エクスプレション＞》

⑵ ステートメント

ａ．宣言ステートメント

関連データのデータ・タイプを宣言する。

① ＩＴＥＭステートメント

構 成

＜ＩＴＥＭ ステートメント＞：：＝ＩＴＥＭ＜アイテム・プライマリＩＤリスト＞

＜アイテム・プライマリＩＤリスト＞：：＝＜アイテム・プライマリＩＤ＞

| <アイテム・プライマリＩＤ>，<アイテム・プライマリ
ＩＤリスト>

<アイテム・プライマリＩＤ>：：＝<アイテムＩＤ>／<データ長>／|<アイテムＩＤ>
|<アレイ・アイテムＩＤ>／<データ長>／|<アレイ・アイテム>、

<データ長>：：＝<整定数>|<整変数>

機 能

<アイテム・プライマリＩＤリスト>で指定された関連データがアイテムであることを宣言し、<データ長>でそのアイテムが持つデータの長さを指定する。データ長を省略した時はデータは持たないものとする。

② ＩTEMVAR ステートメント

構 成

<ＩTEMVAR ステートメント>：：＝ＩTEMVAR <アイテム変数ＩＤリスト>

<アイテム変数ＩＤリスト>：：＝<アイテム変数ＩＤ>
|<アイテム変数ＩＤ>，<アイテム変数ＩＤリスト>

機 能

<アイテム変数ＩＤリスト>で指定された関連データがアイテム変数であることを宣言する。

③ ＳＥＴステートメント

構 成

<ＳＥＴステートメント>：：＝ＳＥＴ<セットＩＤリスト>

<セットＩＤリスト>：：＝<セットＩＤ>
|<セットＩＤ>，<セットＩＤリスト>

機 能

<セットＩＤリスト>で指定された関連データがセットであることを宣言する。

b．アイテムの創成・消去に関するステートメント

① ALLOCATE ステートメント

構 成

<ALLOCATE ステートメント>：：＝ＡＬＬＯＣＡＴＥ<アイテム変数ＩＤ>，<配列名>，<データ長>

機 能

<配列名>で指定された配列に格納されている頭６文字をアイテムＩＤとし、<データ長>

で指定されたデータ長のアイテムを動的にアロケートし、＜アイテム変数ＩＤ＞で指定されたアイテム変数に、そのアイテムのインターナル・ネームがアサインされる。

② ＦＲＥＥ ステートメント

構 成

＜ＦＲＥＥステートメント＞：：＝ＦＲＥＥ＜アイテム・エクスプレション＞

機 能

＜アイテム・エクスプレション＞で指定されたアイテムを消去する。このアイテムが存在しない時は無視する。

ｃ．トリプルの創成・消去に関するステートメント

① ＭＡＫＥステートメント

構 成

＜ＭＡＫＥステートメント＞：：＝ＭＡＫＥ ＜トリプル＞

機 能

＜トリプル＞で指定されたトリプルを創成する。このトリプルの各要素となるアイテムがアロケートされていない時には、このステートメントを実行するに先だってアロケートする。

② ＥＲＡＳＥ ステートメント

構 成

＜ＥＲＡＳＥステートメント＞：：＝ＥＲＡＳＥ＜トリプル＞

機 能

＜トリプル＞で指定されたトリプルを消去する。このトリプルが存在しない時は無視する。

ｄ．セットのメンバ変更に関するステートメント

① ＰＵＴステートメント

構 成

＜ＰＵＴステートメント＞：：＝ＰＵＴ＜アイテム・エクスプレション＞ＩＮ＜セットＩＤ＞
| ＰＵＴ＜セット・エクスプレション＞ＩＮ＜セットＩＤ＞

機 能

＜アイテム・エクスプレション＞で指定されたアイテム、あるいは＜セット・エクスプレション＞で評価されたセットのメンバである全てのアイテムを＜セットＩＤ＞で指定されたセットにメンバとして加える。

② ＲＥＭＯＶＥ ステートメント

構 成

＜REMOVE ステートメント＞：：＝REMOVE ＜アイテム・エクスプレション＞
FROM ＜セットＩＤ＞
| REMOVE ＜セット・エクスプレション＞
FROM ＜セットＩＤ＞

機 能

＜アイテム・エクスプレション＞で指定されたアイテム、あるいは＜セット・エクスプレション＞で評価されたセットのメンバである全てのアイテムを＜セットＩＤ＞で指定されたセットから取り除く。

e．検索処理ステートメント

① FOREACH ステートメント（セットに関するもの）

構 成

＜FOREACH ステートメント＞：：＝FOREACH ＜ローカル変数ＩＤ＞ＩＮ＜セット・エクスプレション＞ＤＯ＜ステートメント番号＞

＜CONTINUE ステートメント＞：：＝＜ステートメント番号＞CONTINUE

実際には次のように使用される。

＜ FOREACH ステートメント＞
.....
.....
..... } ユーザプロセデュア
＜CONTINUE ステートメント＞

機 能

＜セット・エクスプレション＞で評価されたセットのメンバーを一つづつローカル変数にわりあて、一回づつすべてのメンバについてプロセデュアを実行する。プロセデュアは求めたアイテムに対する操作を含む処理過程でFORTRAN ステートメントが記され、＜FOREACH ステートメント＞で与えたステートメント番号をもつ＜CONTINUE ステートメント＞で区切られている。尚 ＦＯＲＥＡＣＨステートメントのネスティングは許されている。

② FOREACH ステートメント （トリプルに関するもの）

構 成

＜ FOREACHステートメント＞：：＝ FOREACH ＜条件文＞DO ＜ステートメント

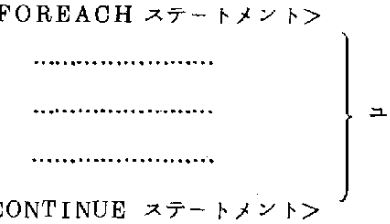
番号>

<条件文>：：=<トリプル>｜<条件文>・AND・<トリプル>

｜<条件文>・OR・<トリプル>

<CONTINUE ステートメント>：：=<ステートメント番号>CONTINUE

実際には次のように使用される。



機 能

<条件文>のトリプルは次の二つの制約をもつ。

o 少なくとも一つ以上のオペランドがローカル変数でなければならない。

o トリプルの要素が3つともローカル変数であってはならない。

そして<条件文>をみたすアイテム全てに対し、一回づつ、プロセデュアを実行する。プロセデュアは (1) FOREACH ステートメントと同様。

尚、FOREACH ステートメントのネスティングは許されている。

f. インターナル・ネームを求めるステートメント

① CONVERT ステートメント

構 成

<CONVERT ステートメント>：：=CONVERT <アイテム変数>，<配列名>

機 能

<配列名>で指定された配列内の頭6文字で表わされるアイテムIDのインターナル・ネームを<アイテム変数>で指定されたアイテム変数にアサインする。

(3) ファンクション

検索のために次のようなファンクションが用意されている。

a. アイテムに関するファンクション

① ISITEM ファンクション

構 成

<ISITEM ファンクション>：：=ISITEM(<アイテム・エクスプレション>)

機 能

<アイテム・エクスプレション>で指定されたアイテムが存在すればTRUE 、しなければFALSE が関数値となる。

② ISMBER ファンクション

構 成

<ISMBER ファンクション>：：＝ISMBER （<アイテム・エクスプレション>，<セット・エクスプレション>）

機 能

<アイテム・エクスプレション>で指定されたアイテムが<セット・エクスプレション>で評価されたセットのメンバであれば TRUE 、そうでなければFALSE が関数値となる。

③ DATA ファンクション

構 成

<DATA ファンクション>：：＝<IDATA ファンクション>｜<RDATA ファンクション>｜<LDATA ファンクション>

<IDATA ファンクション>：：＝IDATA(<アイテム・エクスプレション>，<セル番号>)

<RDATA ファンクション>：：＝RDATA(<アイテム・エクスプレション>，<セル番号>)

<LDATA ファンクション>：：＝LDATA(<アイテム・エクスプレション>，<セル番号>)

<セル番号>：：＝<整定数>｜<整変数>

機 能

<アイテム・エクスプレション>で指定されたアイテムの<セル番号>番目のデータをそれぞれ整数、実数、ロジカル・タイプで関数値として得る。データを格納する時は擬変数として使用される。

b．セットに関するファンクション

① ISNMBR ファンクション

構 成

<ISNMBR ファンクション>：：＝ISNMBR(<セット・エクスプレション>)

機 能

<セット・エクスプレション>で評価されたセットのメンバ数が関数値となる（整数）

② ISSBST ファンクション

構 成

<ISSBST ファクション>：：＝ISSBST(<セット・エクスプレション－1>，<セット・エクス

プレション－2＞）

機 能

＜セット・エクスプレション－1＞で評価されたセットが＜セット・エクスプレション－2＞で評価されたセットのサブセットであればＴＲＵＥ、そうでなければＦＡＬＳＥが関数値となる。

③ ISEQST ファンクション

構 成

＜ISEQST ファンクション＞：：＝ISEQST(＜セット・エクスプレション－1＞,
＜セット・エクスプレション－2＞)

機 能

＜セット・エクスプレション－1＞で評価されたセットと＜セット・エクスプレション－2＞で評価されたセットのメンバが等しければTRUE、そうでなければＦＡＬＳＥが関数値となる。

c．トリプルに関するファンクション

① ISTRPL ファンクション

構 成

＜ISTRPL ファンクション＞：：＝ISTRPL(＜アイテム・エクスプレション＞,
＜アイテム・エクスプレション＞,
＜アイテム・エクスプレション＞)

機 能

(＜アイテム・エクスプレション＞,＜アイテム・エクスプレション＞,＜アイテム・エクスプレション＞)で指定されるトリプルが存在すればTRUE、しなければＦＡＬＳＥが関数値となる。

昭和４７年３月発行

発行所　財団法人 日本情報処理開発センター
東京都港区芝公園３丁目５番８号
機械振興会館内
TEL （４３４）８２１１（代表）

印刷所　株式会社 三 州 社
東京都港区芝大門１丁目１番21号
TEL （４３３）１４８１（代表）

46-S002