ソフトウェア工学　52－21
（1987・2・14）

複数のプロセッサを持つ画像処理システムのための言語について

出口 光一郎
（山形大学工学部情報工学科）

複数のプロセッサを持つ画像処理システム上での、種々の画像処理演算を定義し実行するための、会話形の言語を開発した。

一般に画像処理では、画像データに対して、単独では単純な演算処理を段階的に組み合せることで、画像解析が行われる。対象としている画像処理システムでは、その基本演算を得意、不得意(できない)とする複数のプロセッサを持つ。このとき、どの演算をどのプロセッサで実行するか、また、その実行プロセッサがシステムの開発に応じて交代してもそれをユーザに意識させないことが、使い勝手の上で大切である。また、画像処理では演算の対象となる単位(画像)のデータサイズが大きいので、メモリの割り振り等が厄介であり、画像の置場所を意識しなくてすむとよい。本言語では特にこの2点を重視し、関数形の記法を採用することで、これらを解決している。

画像処理のみならず、複数のプロセッサを持ち機能の分散を計るシステムが増えており、本言語での処理方式はそれらのシステムの上での言語の開発にも応用できる。

An Image Processing Language for the System Having Multiple Image Processors

Koichiro DEGUCHI
Department of Information Engineering
Yamagata University
Yonezawa, 992 Japan

An image processing language was developed to define and operate several types of image processing on the system having multiple image processors. For image processing, it is common to apply combined simple basic operations on image data step by step. For these basic operations, each of the component processor of the system has its characteristics that it has good or weak capability to operate on itself. So that the each basic operations must be distributed to the most suitable processors, respectively. By this language, each of the operations is expressed by using function, and the sequence of the operations is described with combinations of the functional expressions. The user can define complex image processings by combining the simple functions and, at that time, he need not mind to specify which processor operates each of the component operations.

1. はじめに

複数のプロセッサを持つ画像処理システム上での、種々の画像処理演算を定義し実行するための、会話形の言語を開発した。

一般に画像処理では、画像データに対して、単独では単純な演算処理を段階的に組み合せることで、画像解析が行われる 1),2)。対象としている画像処理システムでは、その基本演算を得意、不得意(できない)とする複数のプロセッサを持つ。このとき、どの演算をどのプロセッサで実行するか、また、その実行プロセッサがシステムの開発に応じて交代してもそれをユーザに意識させないことが、使い勝手の上で大切である。また、画像処理では演算の対象となる単位(画像)のデータサイズが大きいので、メモリの割り振り等が厄介であり、画像の置場所を意識しなくてすむとよい。

本言語では特にこの２点を重視し、関数形の記法を採用することで、これらを解決している。そして、ユーザが関数を合成していくことで、複雑な画像処理操作を実現できるなど、ユーザ自身の使用環境を容易に拡張していくことができる。

2. 画像処理システムと画像処理プログラムの開発

図１に、本言語の開発の対象としている画像処理システムの概略を示す。システムは基本的に次の３つの画像処理プロセッサを持つ。

(1)ホスト(ミニコンピュータHP9000)

本言語処理系が動作し、また、システム全体の制御を行う。ホスト上での画像処理は高級言語のプログラムによって行われる。

(2)高速画像処理装置(ＩＰ)

画像処理ＬＳＩ T9506 を用いた高速画像処理演算を行う。マイクロプログラムにより動作する。

(3)ビデオメモリプロセッサ（ＶＭ）

ＩＴＶからの画像入力機能を持つビデオメモリであるが、画像の入出力の際に濃度変換、加減算、二値化、擬似カラー化などの処理を行うことができる。

これらのプロセッサの画像処理の能力と特徴を表１に示す。すなわち、このシステムでは、画像処理にとって、簡単な固定された処理を高速で行う（ＶＭ）、複雑な処理を高速で行うがプログラム開発がめんどう（ＩＰ）、処理はこれらに比べては遅いがプログラミングが容易（ホスト）という３つのタイプのプロセッサを持つことになる。

表１　画像処理システム上のプロセッサ

プロセッサ	処理速度	プログラミング
ホスト (HP9000)	低速 (汎用ミニコン)	容易 (unix + 高級言語)
ＩＰ (T9506)	高速 (画像処理専用LSI)	容易でない (マイクロプログラム)
ＶＭ (ビデオメモリ)	高速 (専用ハードウェア)	固定された処理のみをホストからのコマンドで実行

一般に画像処理では、フィルタリング、二値化、輪郭抽出、などの単独では単純な一単位の処理（以下、画像処理プリミティブと呼ぶ）を順次段階的に組み合せることで一連の処理を行う。本システムでは、それぞれのプリミティブをそれぞれのプロセッサが担う。ＩＰでのプリミティブの実行は、プリミティブが指定される毎に、その処理を実行するマイクロプログラムがホストからＩＰにロードされ、起動される。ＶＭでの処理はホストからのコマンドで起動される。

本システムでの最終目標としての実行形態は、画像処理演算はＩＰ上で行い、ホストがそれを制御する形であり、入出力に伴う簡単な処理はＶＭ上で行う。しかし、ＩＰのためのマイクロプログラムの生成は容易ではなく、アルゴリズムを十分にホスト上の高級言語で練っておく必要がある。すなわち、本システム上の画像処理プログラムの開発は図２のようになる。

(1)で画像処理のアルゴリズム自体の開発、試行を行った処理を、(2)では、基本プリミティブ単位に分割する。それらが未だ存在しないプリミティブであれば、それぞれをどのプロセッサが担うか決定し、本言語のプリミティブとしてインタフェースを合わせる。そして、これらを、既に存在するプリミティブとも組み合せて、画像処理プリミティブ列として実現する(3)。有用なプリミティブ列には新たに名前を付してマクロとして用いることの出来るようにする(4)。よく使うものであれば、そのマクロを一体化したプリミティブ

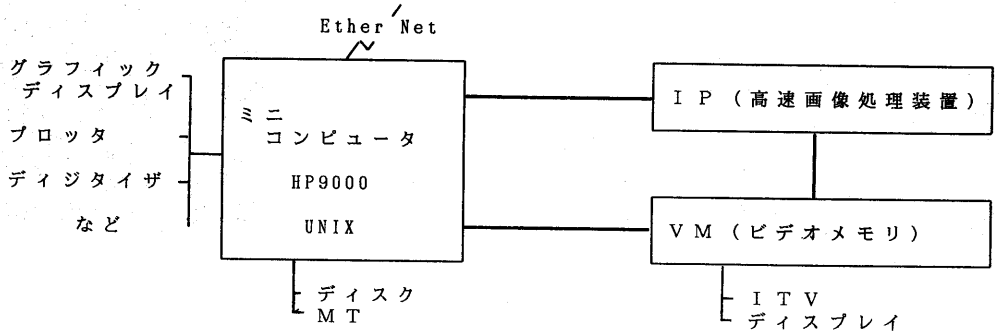


図１　本言語の対象としている画像処理システムの構成。

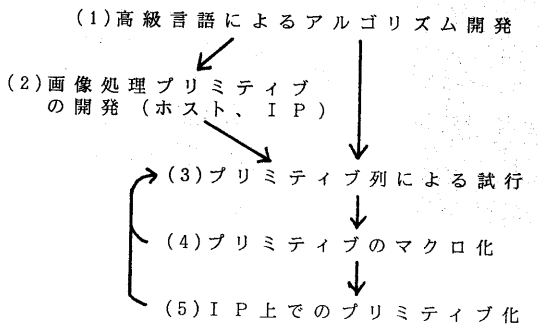


図２　本システムでの画像処理プログラムの開発手順

に落し、高速化する(5)。

本システムの様に複数のプロセッサを持つシステムでは、このように、段階的にマクロ化していく記述を用いてプロトタイプのテストを行い、動作を確認しながらプリミティブの割り当てと高速バージョンへの「差し替え」を行うというプログラム開発の仕方が効果的である。そして、このとき、どのプロセッサがそのプリミティブを担っているか、この差し替えが何時行われたかをユーザが意識しなくてよいことが重要であると考えている。システムの開発者の側でも、この「差し替え」を用いることで、トップダウンでもボトムアップでも、さらに「思い付き順」でもシステムの改良ができる。

3.関数形言語による画像処理アルゴリズムの記述

3.1 関数による画像処理の記述

前述の様に、多くの画像処理は基本的な処理演算を段階的に組み合せることで実現される。これは、画像処理の過程が、前処理 －＞ 特徴抽出 －＞ 特徴の解析、とデータを順次抽象化していくことに対応する 2)。このそれぞれのステップを細分し、いろいろな処理に共通するプリミティブとして登録しておくことは、汎用の画像処理システムとして有用であるばかりでなく、それを用いて画像処理の過程をプログラムすることで、アルゴリズムのよい記述ともなる。

そして、このプリミティブを関数として、すなわち、画像データあるいは画像特徴を別の画像データ、特徴に変換するものとしてとらえると、上記の逐次的な画像処理の過程は、与えられた画像に対して次つぎと関数を適用することとして表現できる。

このような考え方から開発した本言語での簡単な記述の例を図３に示す。(a)では、GIRLやL_GIRLは画像に、LEVELはある値に付けた名前、display,laplacian,thrshld は処理を表わす関数名である。(b)では、PARTICLE が画像、HIST は配列、LEVEL は同じく値である。histogram と valley が 関数である。このように、画像Ｉに処理演算 func を施した結果は、

func (I)

で表わす。このとき、funcはそれ自体の型に応じた値を返す。したがって、例えば順次 func1, func2, func3 という「画像型」の関数を画像 IN に施し、出力画像 OUT を得る場合、演算は、

OUT = func3(func2(func1(IN)))

と記すことができる。

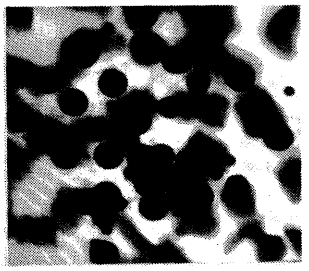
画像処理においては記憶の単位が一般に画像という大量データとなり、その管理がやっかいである。現在、広く出回っているＦＯＲＴＲＡＮによる画像処理サブルーチンパッケージ（ＳＰＩＤＥＲ，ＬＡＲＳＹＳなど）では、サブルーチン SUB1, SUB2, SUB3 を続けて適用する場合、それ

display(GIRL)

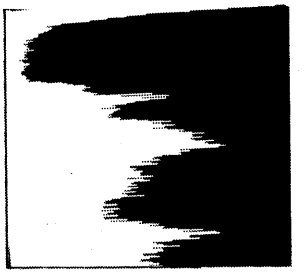
display(L_GIRL
=laplacian(GIRL))

display(thrshld(
L_GIRL,LEVEL))

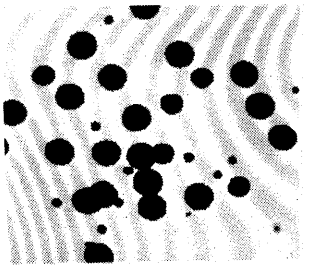
(a) 画像 GIRL にラプラシアン微分フィルタ (laplacian) をかけ、その結果を値 LEVEL をしきい値として二値化 (thrshld) する。



display(PARTICLE)



HIST=histogram(PARTICLE)



LEVEL = valley(HIST);
display(thrshld(PARTICLE,
LEVEL)

(b) 画像 PARTICLE のヒストグラムを求め (histogram)、その谷となる値 LEVEL を関数 valley で決定し、二値化する。

図３　本言語による画像処理の記述の例。

ぞれで入出力画像を明示しなくてはならないので、中間の作業領域が必要である。すなわち、

```
CALL SUB1( IN, W1 )
CALL SUB2( W1, W2 )
CALL SUB3( W2, OUT)
```

ここで、W1,W2 は単に中間結果であるが、名前と領域を与えねばならず、プログラムが煩雑であるのみならず、そのデータ量が大きいだけに効率が悪い。本言語の様な関数形の記述では、上記の各出力を明示しなくてよい。

さらに、本システムでは複数のプロセッサ間で画像を処理に伴い移動させねばならないので、後に必要な画像か、中間結果で消滅してよい画像なのかをきちんと管理することで処理時間のうえでも大きく効率が違う。

ただし、画像処理では、２つ以上の出力が必要な場合がある。関数からの出力は一般に１つであるので、いわば副作用のような形で多出力も許す。すなわち、図３の例で、display(L_GIRL = laplacian(GIRL)) は laplacian の出力画像に L_GIRL と名前を付けて保存すると共に、この代入自体は代入した画像 L_GIRL を値として持つことになり、次の関数 display の一引数となっている。

3.2 型

本言語では変数などすべての名前が型を持つ。関数にも型が定義される。画像処理では、ある処理はある特定の型のデータに対してのみ意味を持つので、型の整合をチェックすることで、無意味な演算の定義を避けることができ、プログラムの誤りを容易に検出できる。型の変換は、関数の適用によってのみ可能である。図３(b) の例で、histgram(IN)は、画像 IN の濃淡ヒストグラムを作る関数であり、「画像型」の引数をとる「配列型」の関数で、出力を「配列型」データ HIST に代入している。また、 valley は 「配列型」を引数にし、「値型」を返す。

3.3 マクロ関数

一連の画像処理操作に名前を付け、新しい合成関数として定義することが出来る。これをマクロ

LADY　　range(LADY)

図４　マクロ関数 range の実行例。

関数と呼ぶ。定義は、後述のシステムコマンドを用いる。例えば、max は最大値フィルタ、min は最小値フィルタ演算の関数であるとき、最大値フィルタの出力から最小値フィルタの出力を差し引くことでレンジフィルタを得るので、

```
.def  range( $x )
    sub( max( $x ), min( $x ))
```

と定義する($は仮引数を表わす)と、以後、

```
range( IN )
```

でレンジフィルタ演算が行える(図４)。マクロ関数は定義によってそのテキストが登録され、実行時に引用された段階で展開される。

前述の様に、有用なマクロ関数はシステム側でいつでも一体化して高速バージョンへ差し替える事が出来る。いつ差し替えられても構わない。また、マクロ関数での引用は何レベルでもネスティングしてよい。これらをユーザが意識しなくて良いことが重要である。

さらに、本言語では、関数自身を一種の値と考えて、ほかの関数の引数として用いることを許す、高階関数を定義することができる。画像処理の場合、例えば、平滑化といっても実は種々の処理がある。マクロ関数への合成の中間段階で、平滑化が必要であっても、どの平滑化処理を用いるかを実行時まで保留するといったことが、これによって可能である。すなわち、

```
.def   h_o_f( $X, $SM )
        func( $SM( $X ) )
```

と定義すると、average(), median() といった異なる平滑化関数を、h_o_f(IMAGE, average) とか h_o_f(IMAGE, median) というように引用することができる。

4．言語の仕様

4.1 プログラム

言語の構文の詳細は省略する。後述のシステムコマンドを除いて、プログラムは変数や型の宣言をする部分と実行文からなる。

実行文の基本単位は、関数の適用と結果の代入であり、

〔〈出力〉=〕関数(〈引数〉〔,〈引数〉〕・・・)

と表わす。この文を ; を用いていくつもつなげられる。結果が他の関数の引数となっているときは、そのデータを示すポインタがスタックを通して次の関数へ渡される。代入を伴わない関数の結果は捨てられる。すなわち、

```
OUT = func2( func1( IN ) )
```

の func1 の結果は、func2 に渡された後捨てられる。

```
OUT = func2( MID = func1( IN ) )
```

とすると、MID に中間結果が残される。代入文自体も値と型を返し、それらは代入された値自身と同じであるので、このようなことができる。

制御文も関数の形で与えられる。if, when, unless という名前の関数が用意されていて、

```
if( 〈条件〉, 〈関数〉, 〈関数〉 )
when( 〈条件〉, 〈関数〉 )
unless( 〈条件〉, 〈関数〉 )
```

というように使い、それぞれ引数である条件に応じて、同じく引数である関数が実行されたりされなかったりする。〈条件〉は、bool型変数またはそれに対する論理演算か、値型変数に対する比較のみである。

4.2 基本要素

名前 (name)

データや関数を登録し、参照するために、すべての基本要素は名前を持つ。すべての参照はポインタによることを基本にしており、名前の実体はポインタである。

式 (expression)

プログラムとして入力されるもののうちいくつかの予約語以外は式である。名前も式である。式は評価された時、式の値 (exprval) と式の型 (exprtype) を返す。式の値はすべて、式の型に応じた実体を指すポインタである。

型 (type)

主な型は表２のものがあり、それぞれの値は表のように対応する。

表２　式の型とその値

型	返す値の実体
1　pic（画像データ）	画像データの先頭へのポインタ
2　value（値）	その値へのポインタ
3　array（配列）	その配列へのポインタ
4　bool　（真、偽）	その値へのポインタ
5 以降 user_defined （ユーザ定義レコード）	そのレコードへのポインタ

新しく使用する変数は、すべて、

```
type  GIRL pic
type  HIST array[256]
```

のようにして、その型を宣言する。

このほかに型をユーザが定義することが出来、その仕方は、例えば、

```
type  pic_pair {
      pic      original;
      pic      filtered;
      }
```

とすると、以後、pic_pair型、および、現在は不完全な形でではあるが、pic_pair.original型、pic_pair.filtered型のデータを宣言することができる。

特殊変数

ハードウェアの構成上、いくつかの特殊な画像メモリにあらかじめ名前が付いている。これらは、画像データそのものではなくデータの在り場所に付けた名前ということになる。例えば、ITV_IN, VM01, IP_DM11, IP_DM21,・・・などで、ITV_INはテレビカメラから入力される（であろう）画像、VM_・・はビデオメモリ内の、IP_DM・・は IP 内の作業用のメモリであり、これらの名前の実体は、それぞれのメモリの先頭番地を差すポインタである。本言語の処理系では、データを表わすすべての変数がその実体としてデータへのポインタを持つので、このような特殊な変数も統一的に扱える。ただし、これらの引用や代入は、画像のそれぞれのメモリから、または、それぞれのメモリへの移動を伴う。

4.3 動作モード

システムには２つの動作モードがある。ダイレクトモードとバッチモードである。前者では、実行文の一行の入力が終了した時点で、入力行はプリミティブの呼出しに展開され、直ちに実行される。このモードは、一時的な画像の移動や処理に加え、現在の状態で次のステップの操作が正しく動くか（未定義のものが残っていないか）のチェックに用いることが出来る。

一方、バッチモードでは画像処理プログラムはプリミティブの呼出しに展開された後、実行はされずに"オブジェクト"とよばれるリストの末尾に追加される。オブジェクトは処理プリミティブを実行順に並べたリストである。オブジェクトの消去や一括実行は後述のシステムコマンドによってなされる。

この２つのモードを使い分けることで、一段一段結果を確かめながら処理を進め、しかもミスに対して後戻りして再計算をすることが可能になる。

4.4 システムコマンド

システムを管理するためのコマンドを持つ。それらは、一文字目が .で始まり、改行で実行される。表３に代表的なものを示す。どの時点で実行してもよい。前述のマクロ関数の定義なども、システムコマンドを用いて行う。

5．言語処理系

言語の処理系、特に実行の制御方式について述べる。構文の解析については省略する。処理系は、基本的には会話形のインタプリタであるが、実行

表３　主なシステムコマンド

.init	作業領域の初期化
.quit	終了
.{	object_list を消去、初期化
.}	object_list による実行
.exec	
.ls	現在定義されている名前の一覧を表示
.lsa,.lsl	詳しく表示
.def	マクロ関数を定義
.setp,.setv	ポインタ、値の定義
.defent	処理プリミティブのエントリを定義
.rm	名前を消去

表４　n_cell の属性

属性	内容と実体	登録の仕方
1 マクロ関数	関数列のテキスト。 n_ptr はそのテキストへのポインタ	.def name { 関数列 }
2 エントリ	ホスト上の関数。 n_ptr はその関数へのポインタ	.defent name ルーチン名
3 マイクロ	IPのためのマイクロプログラムのテキスト n_ptr はその先頭へのポインタ	.micro
4 ポインタ	n_ptr に定義された値が入る	.setp name expression
5 値	値を n_val に置き、それへのポインタが n_ptr に入る	.setv name expression
6 データ	画像を含め、何らかのデータに付けた名前。n_ptr にそれへのポインタが入る	.setn

＊ VM上での処理は、実際はホストからコマンドで起動されるので、2に含まれる。

時に”オブジェクト”を作りだし、再実行では構文解析はしない。

処理系は言語Ｃで書かれている。以下の説明においては、Ｃでの記法を用いる。

5.1 n_cell, name_list, ライブラリ

名前の参照のためのリストがライブラリである。n_cellはライブラリにおける記憶の単位である。各 n_cell は名前 (name) を持ち、名前は登録された順にリスト (name_list) を構成している。すなわち、name_list に登録された n_cell の全体がライブラリである。

n_cell は次の様な構造体である。

```
struct  n_cell  {
        struct   n_cell    *n_next;
        char               *n_name;
        long               *n_ptr;
        long                n_val;
        struct   envir     *n_env;
        int                 attr;
        int                 type;
    };
```

n_cell には６つの属性があり、メンバ attr で識別される。それらを表４に示す。n_cell として名前が登録されているものの実体が何なのか (どのような関数か、データか) を表わす。

したがって、それぞれの名前は、例えば、画像

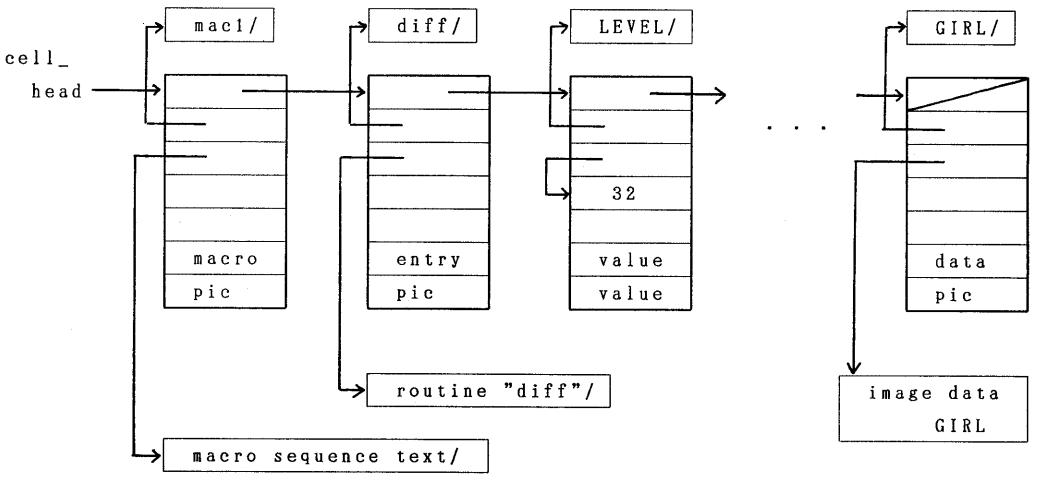


図５　name_list の構成例。

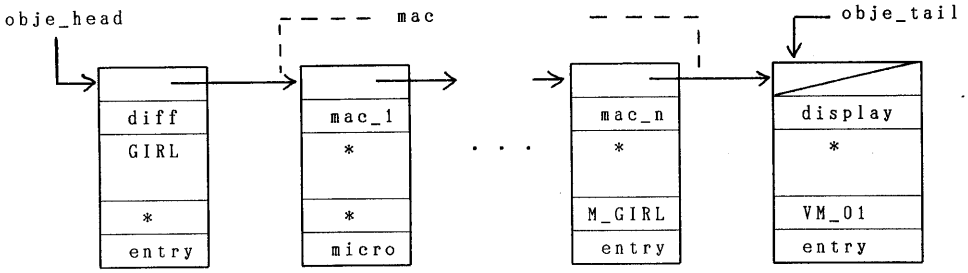


図６　object_list の構成例。 display(M_GIRL=mac(diff(GIRL))) を実行した場合。ここで、mac は mac_n(・・・mac_1( )) というマクロ関数である。

型のデータ、画像型の関数、配列型のマクロ関数といったように、型と属性を持つ。envir は関数の場合、引数の型などの実行時の環境を指定するデータの構造体である。

name_list は、

struct n_cell *cell_head,*cell_tail;

として、先頭と末尾を作ることで実現される。簡単な構成例を図５に示す。

このように、処理ルーチンとデータを共に名前のみで参照し、その実体へのポインタとして統一的に扱っている。この点が本言語の特徴であり、制御を容易にすると共に、拡張性を高めている。

5.2 o_cell, object_list, 実行

実行文から、連続した画像処理ルーチンのつながりである object_list が生成される。object_list は、一つのプリミティブに対応する o_cell から成るリストである。

実行文がスキャンされると予約語や名前が切り出され、返されたトークンに従って、名前の場合は name_list がたどられる。そして、その n_cell が見付かると、その名前の指すデータの参照や関数の実行が実際に行われる環境と、n_cell 自身の持つ環境との比較が行われ、整合のチェックがなされる。

マクロ関数はこの時点で展開される。このとき、マクロ関数の定義で用いている仮引数に対しては、”環境”を引き継いで埋め込むことも行われる。

o_cell は、次の構造を持つ。

```
struct  o_cell
    { struct  o_cell  *o_next;
      long            *exp_ptr;
      long            **o_par_list;
      long            *o_output;
      int              o_type;
    };
```

object_list の実現のためには次のポインタが用意される。

```
struct  o_cell  *obje_head, *obje_tail;
```

o_cell の各フィールドに各 n_cell や ”環境” から得た値をセットし、そのo_cellを object_list の末尾につなぐ。図６に、object_list の構成例を示す。 object_list は、二つの実行モード用に、二つ用意される。

実行モードがダイレクトモードであれば、続いてすぐに obje_head で指定された o_cell から順にリスト上の o_cell を取り出して実行され、obje_tail で止り、最後に object_list をクリアする。

バッチモードでは、実行開始のコマンド (.) または .exec) で、実行を開始する。object_list は、消去コマンドまで残される。

６．おわりに

複数のプロセッサを持つ、画像処理システムのための会話形の言語について述べた。画像処理をプリミティブの合成と考え、そのプリミティブを関数で記述することで、画像処理の過程が明解に記述できるようになった。 画像処理のみならず、複数のプロセッサを持ち機能の分散を計るシステムが増えており、本言語での処理方式はそれらのシステムの上での言語の開発のも応用できる。

より高度な言語とするためには、ブロック構造の導入とそのための制御文・条件文などの整備が必要である。また、関数も含めて型チェックを実現したが、型のより多様な定義やその参照については、まだ考察が必要である。

参考文献

1) M.Duff edt., Languages and Architectures for Image Processing, Academic Press (1981)
2) 松山ほか、画像処理演算の複合的合成、情報処理学会研究会報告、86-CV-43 (1986)
3) P.Henderson(杉藤ほか訳),関数形プログラミング、日本コンピュータ協会(1985)