MEMOIRS OF SAGAMI
INSTITUTE OF TECHNOLOGY
Vol. 18, No. 1, 1984

既存言語における抽象化技法支援の一手法について

深　沢　良　彰*

An Approach to Support Abstraction Facilities in Existing Languages

Yoshiaki FUKAZAWA

An approach to support abstraction facilities is described.

In program design, abstraction techniques are very effective. But current familiar languages does not support various kinds of abstraction facilities. In order to add abstraction facilities to existing programming languages, a macro language MACLAM (A Macro Language for Abstraction Mechanisms) has been designed and implemented. MACLAM is a general-purpose and syntax-directed macro language, and attention of this language is focused on data as well as on control. The MACLAM processor is implemented as a complete pre-processor for various kinds of base language processors.

MACLAM supports three kinds of abstraction techniques: (1) procedural abstraction, (2) data abstraction, for which a user can define new data types, define associated operations, and protect them from illegal operations, and (3) syntactic abstraction to develop powerful mode of expressions and to give a method for sequencing arbitrary actions. MACLAM offers a procedure and some special functions for these abstraction techniques.

The purpose of this paper is to illustrate the utilities of abstractions supported by MACLAM and to provide an informal introduction to MACLAM with some examples.

1. はじめに

プログラム言語設計に関する研究において，データ抽象化言語と超高級言語が注目を浴びている[25]。この両者に共通なのは，プログラムが“どのように”動作するかではなく，“何を”するかに重点を置いて記述しようとしていることである。データ抽象化言語は与えられた問題における抽象性の認識に従ってプログラムを設計していこうとする方法論[6,24]に基づいている。抽象データ型の概念は Simula 67[5] において，class として初めてインプリメントされた。それ以来，この概念は，CLU[14] における cluster, Alphard[18] における form, Concurrent Pascal[9] における class, その他多くの新しい言語[11,15,23]にインプリメントされてきている。

* 情報工学科　講師　1983 年 9 月 28 日受付

既存のプログラム言語にデータ抽象化機能を組み込むために，すでにいくつかの研究が行なわれている。これらは専用の前処理系を作成する[4,26]か現在使われているコンパイラを手直しする[1,27]かのいずれかの手法を採用している。

一方，数々のマクロ言語が 1960 年代半ばより開発されてきている[3]。高級言語が広く普及してからもなおマクロ機能は重要であり，マクロに基づく種々の高級言語拡張技法が発表されてきている[19]。

複雑なマクロ呼出し機能をもついくつかのマクロ言語[2,17,20,22]が発表されているが，その中で Leavenworth による構文マクロ[13]は特に重要である。その理由は，構文マクロでは基底言語の構文要素を用いて，基底言語の構文と意味を共に拡張できるからである。

この論文では，構文マクロの持つ機能により，いくつ

かの抽象化技法を支援することを提案する。このためにマクロ言語 MACLAM (MACro Language for Abstraction Mechanisms) を設計・試作した。MACLAM は汎用の構文マクロ言語である。MACLAM の処理系は多種の基底言語の処理系に対する完全前処理系として動作する。

次節で，MACLAM の目標とそのために必要な機能について概観する。続いて，MACLAM の簡単な紹介と MACLAM システムの現状について述べる。最後に MACLAM のマクロ機能によって与えられる抽象化技法の能力と効率について評価を行う。

2. MACLAM の目標と機能

MACLAM は，データの抽象化，構文の抽象化，および手続きの抽象化の 3 種の抽象化技法を支援することを主目標としている。

データ抽象化技法を支援するためには，次の 3 つの機能が必要不可欠である。

(a) 新しい抽象データ型を基底言語の組込みデータ型や他の抽象データ型を組み合わせて定義する機能。

(b) 新しい抽象データ型に対する演算を既存の演算の組合せにより定義する機能。

(c) 定義された抽象データ型変数を誤った演算から保護する機能。

MACLAM では，抽象データ型の定義，これらに対する演算の定義，これらの抽象データ型変数に対する実際の演算は，すべてマクロ呼出し文として表現される。実際に，データ抽象化技法をもたない言語を使ってプログラムを開発する時には，まずシステム・プログラマと呼ばれる熟練したプログラマがデータ抽象化定義マクロと呼ばれるマクロを MACLAM の機能を用いて定義する。このマクロ定義は，抽象データ型を指定された組込みデータ型へと変換するマクロ定義を生成する機能，抽象データ型に対する演算の定義を正しいサブルーチン定義や関数定義へと変換する機能，抽象データ型変数に対する演算を適切なサブルーチン呼出し文または関数呼出しへと変換するマクロ定義を生成する機能をもたねばならない。同時にこのデータ抽象化定義マクロは抽象データ型やその演算を定義するための構文も決定する。2.3 節でデータ抽象化定義マクロの機能の詳細とその例について述べる。

データ保護を行うには，MACLAM の処理系のような完全前処理系では，必らず何らかの記号表操作機能が必要である。MACLAM では，このためにいくつかの特殊なマクロ時の関数が用意されている。ただし，通常これらの関数はデータ抽象化定義マクロ中で使われる。よって，システム・プログラマのみが抽象データの保護に対して関心を払えば十分であり，一般のプログラマからは見えない。データ保護についての詳細な機構は 3.4 節で述べる。

以上のように，データ抽象化機能はデータ抽象化定義マクロによって基底言語中に組み込まれる。このデータ抽象化定義マクロによって，通常のプログラマは必要とするデータ型とその演算を最適な形式で定義し，使用することができる。この結果として，データ抽象化定義マクロは他のマクロ定義に比べて複雑になる。しかし，我々の経験では，抽象データ型は基底言語の形式と類似の形式で定義し，使用できるのが最も望ましい。よって各基底言語に対して，通常 1 つのデータ抽象化定義マクロがあれば十分である。ゆえに，その複雑さはさして問題とはならない。

構文の抽象化の目的は強力な表現形式や任意の文の順序付けを行うことである。PL/I に対して，REPEAT-UNTIL 文や CLU 風の iterator などを与えることが構文の抽象化の例である。

直構文マクロ処理系は本質的に構文抽象化機能を基底言語に対して与える。しかし，完全な直構文パーサは，どんな効率的なパーサを用いたとしても，かなりの処理時間を必要とする。そこで MACLAM では効率向上のために，マクロ呼出しの最初の単語は前置マクロ識別子と呼ばれるマクロごとに固定された単語でなければならない。

構文の抽象化のためには，あらかじめ基底言語の構文を定義しておかねばならない。MACLAM システムではそのために BNF 風の記述を採用している。

マクロ言語は本質的に開放型手続き (open procedure) の機能を与えることを目的としている。しかも通常のプログラム言語は手続きの抽象化をプロシージャやサブルーチンという形式で十分に支援している。よって MACLAM はマクロ展開以外に特別な手続きの抽象化用の機能をもっていない。

3. MACLAM の仕様

この論文の目的はプログラム作成時に MACLAM に

よって与えられる3種の抽象化技法の効用を述べ，MACLAM についての非形式的な紹介をすることである。ここでは MACLAM について完全に記述しようとはせず，抽象化技法支援用の機能に重点をしぼる。詳細については MACLAM マニュアル[16]を参照されたい。MACLAM の構文は PL/I と Pascal の構文の影響を受けている。また以下で例を記述するために，基底言語として PL/I を採用した。

```
%DEF <STATEMENT>
     REPEAT <:ST: %S-STATEMENT :> UNTIL %E-EXPRESSION ;
%BODY    LABEL1%INDEX :
              <:ST: %S :>
              IF %E
                   THEN GO TO LABEL2%INDEX ;
                   ELSE GO TO LABEL1%INDEX ;
         LABEL2%INDEX :
%DEFEND
```

(a)

```
    REPEAT   NEXT = NEXT + 1 ;
             MAX = MAX - 1 ;
             UNTIL NEXT > MAX ;
```

(b)

図 1 MACLAM プログラムの例
(a) マクロ定義
(b) マクロ定義 (a) に対するマクロ呼出し文

3.1 マクロの定義と呼出し

マクロ定義の例を図1(a)に示す。この例は IF-THEN-ELSE 文と GO TO 文を使って，REPEAT-UNTIL 型制御構造を定義している。即ち，構文の抽象化の一例である。

予約語 %ĐEF と %BOĐY との間の文字列をマクロ・テンプレートと呼ぶ。マクロ本体は %BOĐY から始まり，%ĐEFENĐ までで終る。

マクロ・テンプレートではそのマクロ定義の構文型と原型構造を定義する。構文型はこのマクロ定義がどのような構文要素に属するかを示す。この例の構文型は〈STATEMENT〉であり，このマクロ原型構造にマッチする文字列は文と通常呼ばれている構文要素に属することを示している。原型構造は前置マクロ識別子ではじまらねばならない。この例では REPEAT が前置マクロ識別子である。

1つのパラメータは % ではじめる。パラメータはその名前と属性からなっている。マクロ呼出し文中の引数によって，マクロ本体中の対応するパラメータ名の個所が置き換えられる。属性はあらかじめ BNF 風に定義されているか，他のマクロ定義のマクロ・テンプレートの

```
%DEF ABSTRACT DATA DEFINITION ;
       TYPE %TYPE-IDENTIFIER = %STRUCTURE-IDENTIFIER %ATTRIBUTE ;
    <:1: %LABEL-IDENTIFIER : PROC (:1:( %PARM1-IDENTIFIER
      <:2: , %PARM2-IDENTIFIER | :> ) :) (:2: %RETURNATTRIBUTE :) ;
        <:3: %STATEMENT-STATEMENT :>
         END  %LABEL2-IDENTIFIER ; :>
     END ABSTRACT DATA DEFINITION ;
%BODY
   "%DEF  DECLARE  %NAME-IDENTIFIER "  %TYPE ;
   "%BODY"
   %IF %STRUCTURE = 'ARRAY'
        %THEN  "DECLARE %NAME " %ATTRIBUTE ;
               "%VARIABLE ( %NAME , " %TYPE )
        %ELSE  %IF %STRUCTURE = 'STRUCTURE'
                    %THEN "DECLARE 1 %NAME " %ATTRIBUTE ;
                          "%VARIABLE ( %NAME , " %TYPE )
                    %ELSE "DECLARE %NAME " %ATTRIBUTE
                          "%VARIABLE ( %NAME , " %TYPE )
               %END
   %END
  "%DEFEND"
  %DECLARE %PARMCTR INTEGER INITIAL (1) ;
  <:1: %OPERATION ( %TYPE,%LABEL,%PARMCTR,%PARM1 )
    %PARMCTR = %PARMCTR + 1 ;
    %IF %PARM1 ¬= NULL
        %THEN "%DEF"  DECLARE %PARM1 "%PARMTYPE"
              "%BODY" DECLARE %PARM1 "%PARMTYPE"
                  "%OPERAND ( " %LABEL,%PARM1 ",%PARMTYPE )"
              "%DEFEND"
   %END
   <:2: %OPERATION(%TYPE,%LABEL,%PARMCTR,%PARM2)
        %PARMCTR = %PARMCTR + 1 ;
              "%DEF"  DECLARE %PARM2 "%PARMTYPE"
              "%BODY" DECLARE %PARM2 "%PARMTYPE"
                  "%OPERAND ( " %LABEL,%PARM2 ",%PARMTYPE )"
              "%DEFEND"
   :>
```

(A)
(C)
(D)
(B)
(C')
(D')

図 2 PL/I のサブセットに対するデータ抽象化定義マクロの例（一部）

構文型として定義されていなければならない。

中括弧 (EBCĐIC では <: と :> によって表わされる) は連続して1回以上括弧内が続く時に用いる。この連続するパラメータのマクロ・テンプレートとマクロ本体中の対応は一意な名前によって示される。この名前は左括弧の直後にはじまり，コロンで区切られる。

マクロ・テンプレート中で使用される他の制御構造としては，選択を表わす大括弧と多者択一を表わす垂直線 ("|") がある。(: と :) は大括弧の同義語として使用される。

一意的な記号を生成するために，マクロ時の関数 %INĐEX をマクロ本体中の記号に付加することができる。%INĐEX の機能は IBM OS/370 アセンブリ言語[12] における & SYSNĐX の機能と類似している。

図2に別なマクロ定義の一部を示す。MACLAM ではマクロ時の変数 (% で始まる英数字 31 文字以内) は %ĐECLARE 文中で必らず宣言しなければならない。%IF, %THEN 等はマクロ時の制御文である。また2重引用符はマクロ機能を抑制するために使われる。

このマクロ定義は，その使用法から，データ抽象化定義マクロと呼ばれる。データ抽象化定義マクロの機能および図2で使われている %OPERATION, %OPERANĐ, %VARIABLE 等の使用法は各々 3.2 節と 3.4 節で述べる。

3.2 データ抽象化定義マクロ

データ抽象化機能はデータ抽象化定義マクロによって基底言語に組み込まれる。システム・プログラマはデータ抽象化定義マクロを定義し，新しいデータ型とその演算の指定の方法を一般のプログラマに示す。図2は PL/I のサブセットに対するデータ抽象化定義マクロの例の一部である。

システム・プログラマは次のような機能を含むようにデーマ抽象化定義マクロを定義する必要がある。

(1) 定義されたデータ型を指定された組込みデータ型へ変換するマクロ定義を生成する機能。図2(A) 参照。この生成されたマクロ定義を宣言マクロと呼ぶ。宣言マクロは抽象データ型の変数が宣言されるたびに呼び出され，展開される。

(2) 抽象データ型変数に対する演算の定義を基底言語のサブルーチン定義または関数定義へ変換する機能。

(3) 抽象データ型変数に対する演算を適切なサブルーチン呼出し文または関数呼出しへと変換するマクロ定義を生成する機能。この生成されたマクロ定義は演算マクロと呼ばれ，抽象データ型の変数に対する演算において呼び出される。

(4) 抽象データ型変数を誤った演算から保護する機能。図2(B) 参照。データ保護に関する MACLAM の詳細な機能は 3.4 節で述べる。

データ抽象化定義マクロと生成されるマクロ定義との関係を図3に示す。この関係は 3.3 節で例を用いて再度説明する。図3における図番はそこでの例に対応している。なお，図3の矢印に付加された (1)～(3) は上記の

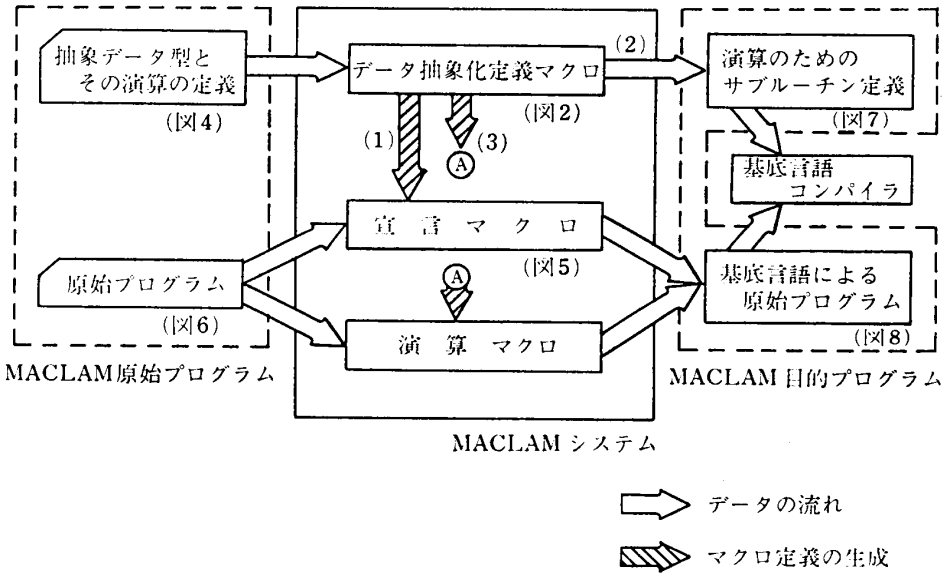

図 3　データ抽象化定義マクロによる抽象データ定義と原始プログラムの変換過程

機能に対応する。

データ抽象化のための仕様はこのデータ抽象化定義マクロの内容によって決定される。ゆえに，このデータ抽象化定義マクロ次第で種々のデータ抽象化技法を組み込むことができる。たとえば，もし必要なら，ユーザが定義した抽象データ型をパラメータとしてわたすようにすることさえ可能である。

ある種のエラー検出機構をデータ抽象化定義マクロに組み込むことができる。たとえば，基底言語のもつ全域的データ型（PL/I における EXTERNAL, Fortran における COMMON など）が抽象データ型定義の中に指定されていた時に警告メッセージを出すことなどが可能である。

3.3 抽象データ型の定義とその使用法

MACLAM のユーザはデータ抽象化定義マクロに従って抽象データ型を定義する。一例を図4に示す。この例では，図2のデータ抽象化定義マクロに従って，SET 型データとその演算を定義している。SET 型のデータ・オブジェクトは1ビットの配列で表わされ，1要素を1ビットに対応させている。定義されている演算は，FULL, ĐELETE, IN, PRINT の4種類である。たとえば，ĐELETE は2番目のパラメータで示される集合から最初のパラメータで示される要素を消去する。

```
ABSTRACT DATA DEFINITION ;
        TYPE  SET = ARRAY ( 0:1000 ) BIT ( 1 ) ;
    FULL : PROC ( SIEVE ) ;
           DCL  SIEVE      SET ;
           DCL  COUNTER    BIN FIXED ( 15,0 ) ;
        DO COUNTER = 0 TO 1000 ;
             SIEVE ( COUNTER ) = '1'B ;
        END ;
    END FULL ;
    DELETE : PROC ( NUMBER , SIEVE ) ;
             DCL  SIEVE       SET ;
             DCL  NUMBER      BIN FIXED ( 15,0 ) ;
          SIEVE ( NUMBER ) = '0'B ;
    END DELETE ;
    IN : PROC ( NUMBER , SIEVE ) RETURNS ( BIT ( 1 ) ) ;
         DCL  SIEVE      SET ;
         DCL  NUMBER     BIN FIXED ( 15,0 ) ;
      IF SIEVE ( NUMBER ) = '1'B
           THEN  RETURN ( '1'B ) ;
           ELSE  RETURN ( '0'B ) ;
    END IN ;
    PRINT : PROC ( SIEVE ) ;
            DCL  SIEVE      SET ;
            DCL  COUNTER    BIN FIXED ( 15,0 ) ;
         DO COUNTER = 0 TO 1000 ;
              IF SIEVE ( COUNTER ) = '1'B
                   THEN PUT LIST ( COUNTER ) ;
         END ;
    END PRINT ;
END ABSTRACT DATA DEFINITION ;
```

図 4 データ抽象化定義マクロ（図 2）に基づく抽象データの定義例

```
%DEF   DECLARE  %NAME-IDENTIFIER  SET ;
%BODY  DECLARE  %NAME  ( 0:1000 ) BIT ( 1 ) ;
%DEFEND
```

図 5 データ抽象化定義マクロ（図 2）により生成された宣言マクロの例

抽象データ型 SET が図4のように定義されると，図5のような宣言マクロが図2のデータ抽象化定義マクロによって生成される。この宣言マクロは抽象データ型 SET の宣言を組込みデータ型で配列とビット列の宣言に置き換える。抽象データ型変数に対する演算を対応するサブルーチン呼出し文または関数呼出しへと変換する演算マクロもこの宣言マクロと同程度に簡潔である。

図6に示した例はエラトステネスの篩として知られているアルゴリズムで素数を求めるプログラムである。この例では，図4で定義された SET 型変数と図1の REPEAT-UNTIL 文が使われている。

図4の抽象データ定義は図2にその一部を示したようなデータ抽象化定義マクロによって図7へと変換される。図6のプログラムは，データ抽象化定義マクロから生成された宣言マクロ（図 5）と演算マクロおよび図1 (a) の REPEAT-UNTIL マクロによって，図8のプロ

```
PRIME : PROC OPTIONS ( MAIN ) ;
        DCL  TARGET   BIN FIXED ( 15,0 ) ;
        DCL  NEXT     BIN FIXED ( 15,0 ) INIT ( 2 ) ;
        DCL  SIEVE    SET ;
     FULL ( SIEVE ) ;
     DELETE ( 0 , SIEVE ) ;
     DELETE ( 1 , SIEVE ) ;
     DO WHILE ( NEXT <= SQRT (1000) ) ;
          DO TARGET = 2*NEXT TO 1000 BY NEXT ;
               DELETE ( TARGET , SIEVE ) ;
          END ;
          REPEAT NEXT = NEXT + 1 ;
               UNTIL IN ( NEXT , SIEVE ) ;
     END ;
     PRINT ( SIEVE ) ;
END PRIME ;
```

図 6 抽象データ型 SET の変数を使用したプログラム例

```
FULL : PROC ( SIEVE ) ;
       DCL  SIEVE     ( 0:1000 ) BIT ( 1 ) ;
       DCL  COUNTER   BIN FIXED ( 15,0 ) ;
    DO COUNTER = 0 TO 1000 ;
         SIEVE ( COUNTER ) = '1'B ;
    END ;
END FULL ;
DELETE : PROC ( NUMBER , SIEVE ) ;
         DCL  SIEVE      ( 0:1000 ) BIT ( 1 ) ;
         DCL  NUMBER     BIN FIXED ( 15,0 ) ;
      SIEVE ( NUMBER ) = '0'B ;
END DELETE ;
IN : PROC ( NUMBER , SIEVE ) RETURNS ( BIT ( 1 ) ) ;
     DCL  SIEVE    ( 0:1000 ) BIT ( 1 ) ;
     DCL  NUMBER   BIN FIXED ( 15,0 ) ;
  IF SIEVE ( NUMBER ) = '1'B
       THEN RETURN ( '1'B ) ;
       ELSE RETURN ( '0'B ) ;
END IN ;
PRINT : PROC ( SIEVE ) ;
        DCL  SIEVE     ( 0:1000 ) BIT ( 1 ) ;
        DCL  COUNTER   BIN FIXED ( 15,0 ) ;
     DO COUNTER = 0 TO 1000 ;
          IF SIEVE ( COUNTER ) = '1'B
               THEN PUT LIST ( COUNTER ) ;
     END ;
END PRINT ;
```

図 7 図4の抽象データ定義例に対する MACLAM 目的プログラム

グラムへと変換される。図7および図8は普通の PL/I プログラムであり，PL/I コンパイラによって処理される。

3.4 抽象データの保護

抽象データ型変数を不正な演算から保護するために，MACLAM では2種類の保護テーブル（変数テーブルと関数テーブル）および5種類の保護関数（%OPERATION, %OPERANĐ, %VARIABLE, %FREEVAR, %SEARCH）を用いる。これらは MACLAM システムによって用意された，システム・プログラマのための抽象データ型とその演算であると考えることができる。これらのテーブルと関数との関係を図9に示す。

抽象データに対する不正な演算には，他の抽象データ型のために定義された演算によるものと，基底言語に組み込まれている演算によるものとの2種類がある。

```
PRIME : PROC OPTIONS ( MAIN ) ;
        DCL  TARGET   BIN FIXED ( 15,0 ) ;
        DCL  NEXT     BIN FIXED ( 15,0 ) INIT ( 2 ) ;
        DCL  SIEVE    ( 0:1000 ) BIT ( 1 ) ;
        DCL  IN       RETURNS ( BIT ( 1 ) ) ;
     CALL FULL ( SIEVE ) ;
     CALL DELETE ( 0 , SIEVE ) ;
     CALL DELETE ( 1 , SIEVE ) ;
     DO WHILE ( NEXT <= SQRT (1000) ) ;
        DO TARGET = 2*NEXT TO 1000 BY NEXT ;
           CALL DELETE ( TARGET , SIEVE ) ;
        END ;
        LABEL101 :
           NEXT = NEXT + 1 ;
           IF IN ( NEXT , SIEVE )
              THEN GO TO LABEL201 ;
              ELSE GO TO LABEL101 ;
        LABEL201 :
     END ;
     CALL PRINT ( SIEVE ) ;
END PRIME ;
```

図 8　図6のプログラム例に対する MACLAM 目的プログラム

抽象データ型変数を他の抽象データ型の演算から保護するために，次のような機構がシステム・プログラマによって組み込まれる。

変数テーブルが抽象データを不正な演算から保護するために使われる。マクロ時の特殊関数 %VARIABLE は抽象データ型であると宣言された変数を変数テーブルに登録するために使われる。演算テーブルは抽象データ型に対する演算がオペランドとして適切なデータ型の変数をもっているかどうかを調べるために使われる。%OPERATION と %OPERANĐ が，このテーブルに演算名と予想されるオペランドのデータ型を登録するために用意されている。システム・プログラマは，データ抽象化定義マクロにおいて，関数 %SEARCH を用いてオペランドのデータ型をチェックすることができる。

この保護機構の一部を図2(B)に示す。データ抽象化定義マクロが呼び出された時，即ち抽象データ型とその演算が定義された時，データ型名，演算名，パラメータ名はマクロ時の特殊関数 %OPERATION によって演算テーブルへ登録される。図2(C), (C′) 参照。続いて，図2(Đ) および (Đ′) において，あるマクロ定義が生成される。このマクロ定義は，その演算のパラメータが宣言された時に呼び出され，関数 %OPERANĐ によって，演算テーブルに，演算名，パラメータ名およびそのデータ型名を登録する。他の抽象データのために定義された演算からの抽象データの保護はこのようにして行われる。

MACLAM システムにおいては，抽象データ型変数

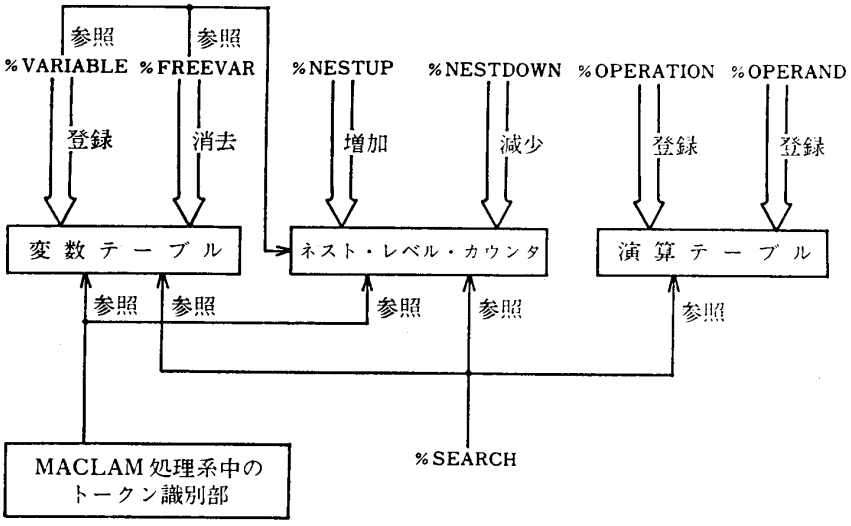

図 9　保護テーブルと関連する関数との関係

への演算はすべてマクロ呼出し文によって行なわれる。組込み演算からの抽象データの保護はこの性質を利用して行う。即ち，MACLAM の処理系によって切り出されたすべてのトークン（語彙的な基本要素）は変数テーブル中に定義された抽象データ型であると登録されているかどうかを調べられる。MACLAM 処理系中のこの検査を行う部分をトークン識別部と呼ぶ。なお，システム・プログラマは展開されたテキストを，抽象データ型変数に対する演算のオペランドの直後から再走査するようにデータ抽象化定義マクロを定義する必要がある。上記の機構を用いることによって，変数テーブルに登録されている変数（トークン）がトークン識別部によって切り出されたら，基底言語の組込み演算によって，保護エラーが引きおこされたことを認識できる。

以上の他に，ネスト・レベル・カウンタと2つのマクロ時関数 %NESTUP %NESTĐOWN とが変数の有効範囲の識別のために用意されている。この機能は MACLAM がブロック構造をもつ基底言語に対して適用される時にたいへん有用である。関数 %FREEVAR は現在のネスト・レベルにおけるすべての変数名およびその情報を変数テーブルから消去するために使われる。変数の有効範囲に関する特殊な規則をもたない基底言語 (Basic, Fortran 等) に対しては，この機能は不必要である。

いくつかの言語では，ある種のデータ型に対して，データ・チェックを完全には行なっていない。このような場合にも MACLAM システムを適用することによって，完全なチェックが可能になる。PL/I においてポインタが異なったデータ構造を指す場合などがその例である。この場合，ユーザはポインタとタグからなる新しいポインタ型を定義し，その演算においてタグをチェックすればよい。

4. MACLAM システムの構造

図 10 は MACLAM システムの概略である。
このシステムへの入力は次の3種に大別される。

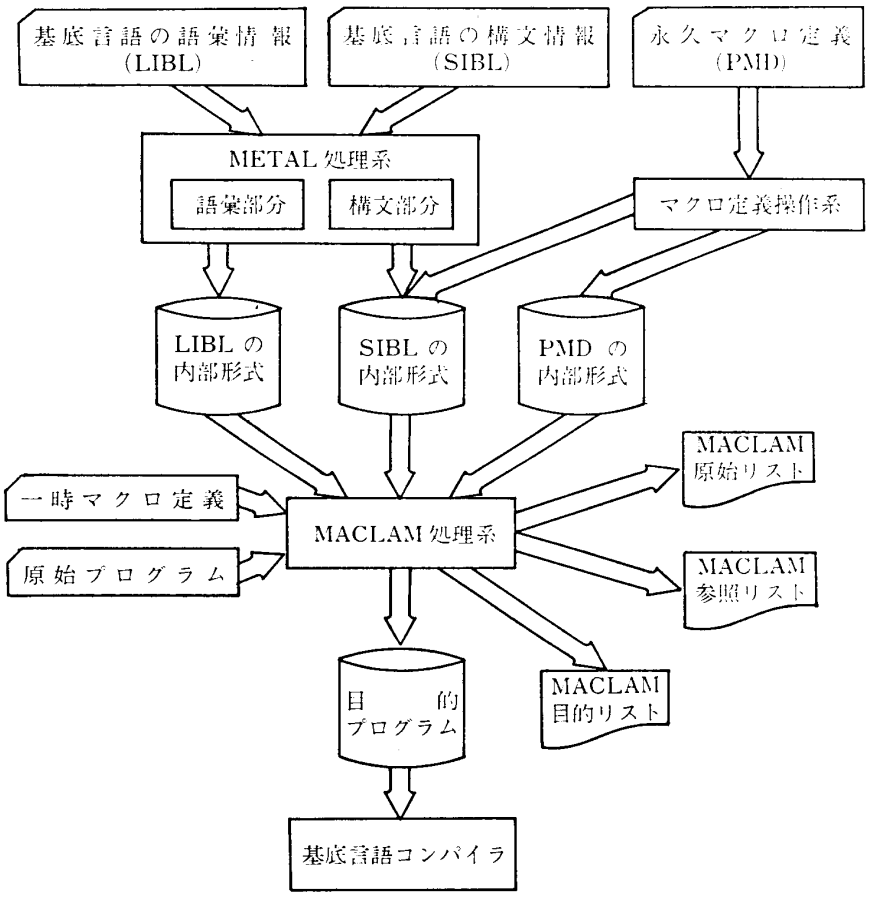

図 10 MACLAM システムの構造

(a) 基底言語の定義
(b) マクロ定義
(c) 種々の抽象化技法が用いられている原始プログラム

またこのシステムの出力は，次に基底言語の処理系により処理され，実行される MACLAM 目的プログラムと，原始リスト，参照リスト等の3種の出力リストである。

マクロ定義は，その使用法から，永久マクロ定義 (PMD) と一時マクロ定義 (TMD) に分類される。永久マクロ定義はデータ抽象化定義マクロや他の必要なシステム・マクロからなる。一方，一時マクロ定義とは現在処理されている原始プログラムにおいてのみ呼び出されるマクロ定義である。

処理の効率化のために，基底言語の情報は語彙情報 (LIBL) と構文情報 (SIBL) に分類される。基底言語の構文情報は BNF 風に指定される。語彙情報と構文情報は METAL (META Language) 処理系によって内部形式へと変換される。経験によれば，正しい文法定義はめったに書かれないので，METAL 処理系のほとんどの処理はさまざまなエラー状態に対して適切な診断メッセージを出すことに費されている。

MACLAM の処理系は原始プログラムとマクロ定義を走査し，基底言語の語彙情報に従ってトークンに分解する。トークン分けされた入力テキストは，構文情報を用いてマクロ呼出し文であるかどうか直構文的にチェックされる。

コンパイラの完全前処理系として動作し，基底言語の構文情報を利用する MACLAM 処理系のような処理系では，効率を軽視することができない。これはマクロ処理系と基底言語コンパイラにおいて二度構文解析が行なわれるからである。さらに，より柔軟な構文をマクロ呼出し文に許せば許すほど，より複雑な認識アルゴリズムが必要になる。

以上のトレード・オフを考え，前節で述べたようなマクロ呼出し文の記法が決められ，さらに次のような構文解析アルゴリズムが採用された。このアルゴリズムは，Earley のアルゴリズム[7]に基づいており，前置マクロ識別子と基底言語の語彙情報と構文情報の分離という2点を用いてさらに改良されている。

本システムを使用する際には，通常のプログラマは一時マクロを定義し，原始プログラムを作成するのみである。他の定義，即ち基底言語定義と永久マクロ定義はすべて，予めシステム・プログラマによって行なわれる。

MACLAM システムは早稲田大学の IBM 4341 システム上で，OS/CMS のもとに作成されている。なお，携帯性を考慮して，システム全体は PL/I で書かれている。

5. 評価と結論

使用経験から以下のような評価が得られた。

まず第1に，基底言語として PL/I を選んだ。その理由は，PL/I は強力ではあるがデータ抽象化機能が欠除しているからである。PL/I の構文定義[21]はたいへん複雑であるので，ある部分集合を採用した。この部分集合に対する構文情報 (SIBL) は約 280 行であり，データ抽象化定義マクロはたいへん複雑ではあるが，たった 70 行ほどである。このマクロを定義し，テストするのに約 0.5 人月が必要であった。MACLAM システムが図4と図6のプログラム例を処理するのに各々約 39 mS と 31 mS 必要であった。

我々の2番目の目標は MACLAM システムを Pascal に適用することであった。Pascal の構文は PL/I の構文に比べて，たいへん簡潔である。Pascal に対する SIBL は約 90 行であった。Pascal 用のデータ抽象化定義マクロは約3人日で定義できた。なお，その大きさは PL/I の場合とほぼ同一であった。このように容易にデータ抽象化マクロを定義できたことは，Pascal 用と PL/I 用のデータ抽象化定義マクロが本質的に同じであったことと，基底言語コンパイラと独立に定義できることに依存していると思われる。今後，種々の言語に対してより多くのデータ抽象化定義マクロを書くにつれて，必要な労力は少なくなっていくと思われる。

図6と等価な Pascal プログラムは MACLAM 処理系によって，20 mS で処理された。この結果から，処理時間のほとんどが直構文的な処理に使われていることがわかる。

我々は現在データ抽象化言語の処理系をもっていないので，以上の結果を比較することはできない。しかし，既存のプログラム言語にデータ抽象化機能を付加しようとするこのアプローチは，Ada 等のデータ抽象化言語が広く普及するまではたいへん有益であると思われる。また専用の前処理系を作ったり，使用中のコンパイラを修正したりすることより容易であろう。しかし，我々の

手法は他に比べて効率の点では不満が残るかもしれない。

同時にこれらの経験から，この方式をさらに追求していくためのいくつかの問題点も明らかになった。これらはマクロ言語のもつ本質的な性質によるものと，抽象データ型の限界によるものとに大別される。

MACLAM はこの種のマクロ言語（汎用の直構文マクロ言語）のもついくつかの欠点を克服できていない。それらのいくつかを以下に示す。

(1) マクロ呼出し文は前置マクロ識別子で始めなければならない。その結果，新しい抽象データ型変数に関して中置演算子は使用できない。ただし，この制限は処理の効率の点から導入されているだけである。

(2) MACLAM 処理系により展開された2つのテキスト間または原始テキストと展開されたテキスト間において，識別子（変数名，ラベル名など）の混同を認識できない。マクロ時の特殊関数 %INDEX が用意されているが，究極的には注意深くマクロを定義する以外に解決策はない。

(3) 基底言語コンパイラによって検出されたエラー・メッセージに含まれる文番号は原始プログラムの文番号と一致しない。ただし，この対応を示す参照リストは MACLAM の処理系によって与えられる。

(4) MACLAM の性質は基底言語の性質によって制限されてしまう。たとえば，MACLAM は分割コンパイルの機能をもたない基底言語に対して，この機能を支援することはできない。

MACLAM の他の欠点は，すべてのデータ抽象化言語に共通である。たとえば，定義されたデータ型は最終的に組込みデータ型の組合せによって表現可能でなければならない。

コンパイラ記述システムは我々のアプローチと深く関連している。確かに簡単なコンパイラをこのシステムによって生成することはできるであろう。しかし生成されたコンパイラは非効率的であるばかりでなく，コード生成と最適化に欠点をもつであろう。

強力な抽象化機能をもった新しいプログラム言語はたいへん有用であるが，長い間使われてきた親しみ深いプログラム言語に各種の抽象化機能を付加することはより利用価値の高いものであろう。

参考文献

1) Banatre, M. *et al.*, An Experience in Implementing Abstract Data Types, Softw. Pract. Exper., **11**, (1981), 315-320.
2) Brown, P. J., The ML/1 Macro Processor, Comm. ACM, **10**, 10 (1967), 618-623.
3) Brown, P.J., Macro Processors and Techniques for Portable Software, (1974), Jonn Wiley & Sons.
4) Burton, W., A FORTRAN Preprocessor to Support Encapsulated Data Abstraction Definitions, Comput. J., **22**, 4 (1979), 307-312.
5) Dahl. O.-J., Hierarchical Program Structures, in Structured Programming, (1972), Academic Press.
6) Dijkstra, E. W., Notes on Structured Programming, *ibid.*
7) Earley, J., An Efficient Context-free Parsing Algorithm, Comm. ACM, **13**, 2 (1970), 94-102.
8) Fukazawa, Y., Abstraction Mechanisms Supposed by a Macro Processor, Journal of Information Processing, **6**, 2 (1983), 59-65.
9) Hansen, P. B., The Programming Language Concurrent Pascal, IEEE Trans. on Soft. Eng., SE-1, **2**, (1975), 199-207.
10) Hoare, C. A. R., Notes on Data Structuring, *ibid.*
11) Ichbiah, J. D. *et al.*, Reference Manual for the Ada Programming Language, SIGPLAN Notices, **14**, 6 (June 1979).
12) IBM, OS/VS-DOS/VS-VM/370 Assembler Language, File No. S370-21, GC33-4010-4 (Jan. 1975).
13) Leavenworth, B. M., Syntax Macros and Extended Translation, Comm. ACM, **9**, 11 (1966), 790-793.
14) Liskov, B. *et al.*, Abstraction Mechanisms in CLU, Comm. ACM, **20**, 8 (1977), 564-576.
15) Musser, D. R., Abstract Data Type Specification in the AFFIRM System, IEEE Trans. on Soft. Eng., SE-6, **1** (1980), 24-31.
16) 野島 章，山縣 良，深沢良彰，MACLAM マニュアル，早稲田大学理工学部電気工学科 (1983).
17) Sassa, M., A Pattern Matching Macro Processor, Softw. Pract. Exper., **9** (1979), 439-456.
18) Shaw, M., Wulf, W. A., and London, R.L., Abstraction and Verification in Alphard: Defining and Specifying Iteration and Generators, Comm. ACM, **20**, 8 (1977), 553-564.
19) Solntseff, N. and Yezerki, A., A Survey of

Extensible Programming Languages, Ann. Rev. Auto. Program., **7**, 5 (1974), 267–307.
20) Strachey, C., A General Purpose Macrogenerator, Comput. J., **8**, 3 (1965), 225–241.
21) Urschler, G., Concrete Syntax of PL/I, IBM Technical Report TR 25.096 (1969).
22) Waite. W. M., The Mobile Programming System: STAGE 2, Comm. ACM, **13**, 7 (1970), 415–421.
23) Wegbreit, B., The Treatment of Date Types in EL 1, Comm. ACM, **17**, 5 (1974), 251–264.
24) Wirth, N., Program Development by Stepwise Refinement, Comm. ACM, **14**, 4 (1971), 221–227.
25) Wulf, W. A., Trends in the Design and Implementation of Programming Languages, IEEE Computer, **13**, 1 (1980), 913–927.
26) Young, S. J., Improving the Structure of Large Pascal Programs, Softw. Pract. Exper., **11**, (1981), 913–927.
27) Zelkowitz, M. V. and Larsen, H. J., Implementation of a Capability-Based Data Abstraction, IEEE Trans. on Soft. Eng., SE-4, 1 (1978), 56–64.