THE
IMP MACRO LANGUAGE
MANUAL

PREFACE

The IMP Macro Scheme is currently under development as an extension to the compile time facilities of the IMP language as defined in the ERCC manual, A Syntactic and Semantic definition of the IMP Language (reference 1). The Scheme was designed originally by Alan Freeman in 1969 (reference 2).

This manual is an introduction to the IMP Macro Language. It is designed to help the experienced IMP programmer use the powerful facilities provided, to extend the permitted input to the normal IMP compilers by writing macros and submitting them together with a program written in the extended IMP language to the IMP macro system.

The software itself is still experimental and is liable to change significantly as it develops. It has undergone substantial testing but is in no sense as solid as the IMP compiler. All faults with carefully documented evidence, should be sent to the author directly.

At present the macro scheme is only implemented on EMAS and there are no plans to make it more widely available.

The author would like to thank Anne Tweeddale who typed this manual.

CONTENTS

INTRODUCTION

The source program which is input to the macro scheme can be considered to be made up of two parts. The first part contains macro language statements which define the extensions to the standard IMP language which the user wishes to make. The second part is the user program written in the extended IMP language which may contain normal IMP statements and the newly defined extensions side by side.

The first part of the input, which specifies the new language phrases and statements using the specialised macro language, also defines their translation into valid statements in the standard IMP language. The extended language may range from a quite minor modification to a complete redefinition of the permitted input.

The following simple example shows the type of source program which is accepted by the macro scheme:

```
%MACRO: '%SET' A(NAME) '%TO' C(CONST)
  :$A$=$C$
  %ENDMACRO
%MACRO: '%ADD' A(NAME) '%TO' B(NAME)
  :$B$=$B$+$A$
  %ENDMACRO
%MACRO: '%DISPLAY' A(NAME)
  :WRITE($A$,1)
  %ENDMACRO
%BEGIN
%INTEGER I,J
%SET I %TO 1
%SET J %TO 3
%ADD I %TO J
%DISPLAY J
NEWLINE
%ENDOFPROGRAM
```

The Macro Scheme is a phrase structure oriented translator whose heart is a syntax analyser which recognises both instructions to the scheme itself and statements in the user defined input language. It translates statements in the extended language into IMP statements according to the rewrite rules specified by the user. The Scheme is available in two forms, a two-pass compiler (whose procedure name is IMPM) and a text to text translator (with procedure name MACRO IMP). The system operates by changing the input text to a suitable internal representation which is then used to manipulate strings in the extended language and convert them into strings in IMP. In the two pass compiler the resulting internal representation is passed to the compilation pass to produce a normal object program, but in the macro translator it is converted back into valid IMP text which may of course be presented subsequently to the standard IMP compiler.

IMP macros extend the current IMP syntax by adding new alternatives to existing phrases. These extensions are normally added on to the end of all the existing alternatives in the order in which they are specified but they may also be added to the front if so desired.

The result of a call on a macro is that in-line IMP is planted in the program at that point. If a macro extends phrase source statement, a sequence of IMP source statements may replace the macro call. If, however, the macro extends any other phrase, the effect is of an insertion of a sequence of IMP source statements in front of the statement containing the call.

SECTION 1 - IMP SYNTAX

Imp is a phrase structure language; that is, its syntax is governed by a series of rewrite rules. These rules are specified by a set of PHRASE DEFINITIONS. The basic unit of input to the IMP scheme is the statement or SOURCE STATEMENT (SS). Phrase 'source statement' is the generator of the language and contains a list of alternatives which uniquely define the language statement types which form the permitted input to the IMP system of compilers. Each alternative is defined in terms of other phrases and literals (i.e. actual IMP text). All other phrases similarly contain a list of alternatives which can be defined in terms of phrases, literals or built-in phrases. Built-in phrases are phrases which are recognised by means other than formal syntax analysis for the purposes of efficiency and convenient internal representation. The most prominent examples are alphanumeric identifiers (NAME), and constants (CONST).

Reference 1 contains a detailed description of the syntax and semantics of the IMP language, however, for the convenience of the macro programmer the syntax is reproduced below in the form that it is used in the macro scheme.

The conventions used in the specification of the syntax given below are as follows. A phrase definition is a phrase name preceded by the letter P or the letter V and followed by an equals sign which should be read as 'is defined as'. This is in turn followed by a list of the alternatives produced by that phrase separated by commas, which should be read as 'or', and terminated by a semi-colon. The null alternative is represented by the symbol 0 (zero). Note that phrase names are enclosed in round brackets and literals are enclosed in single quotes.

```
e.g.      P(ONOFF) = '%ON',
                     '%OFF';
```

The letter V is used in place of the letter P to introduce certain phrases known as SPECIAL PHRASES. Special phrases are known to the macro scheme and their names are held in a special dictionary. The scheme can only operate on these phrases when defining transformations; see Section 8 on elastic phrases.

```
P(+')=
    '+',
    '-',
    '¬',
    0;
P(PLUS')=
    '+',
    '-',
    0;
V(OPERAND)=
    (NAME)(APP)(ENAME'),
    (CONST)(IMPMULT),
    '('(EXPR)')',
    '!'(EXPR)'!';
P(IMPMULT)=
    (NAME)(APP)(ENAME'),
    0;
V(EXPR)=
    (LEAVE HOLE)(+')(OPERAND)(SET MARKER)(RESTOFEXPR);
V(RESTOFEXPR)=
    (OP)(OPERAND)(RESTOFEXPR),
    0;
V(APP)=
    '('(EXPR)(RESTOFAPP)')',
    0;
V(RESTOFAPP)=
    ','(EXPR)(RESTOFAPP),
    0;
V(OP)=
    '**',
    '+',
    '-',
    '||',
    '!',
    '*',
    '//',
    '/',
    '&',
    '>>',
    '<<',
    '.';
P(,')=
    ',',
    0;
P(%IU)=
    '%IF',
    '%UNLESS';
P(%WU)=
    '%WHILE',
    '%UNTIL';
P(CMARK)=
    '!',
    '%COMMENT';
```

```
V(TYPE)=
    '%INTEGER',
    '%REAL',
    '%BYTE' '%INTEGER',
    '%SHORT' '%INTEGER'
    '%LONG' '%REAL',
    '%STRING'(RESTOFUCS);
P(RT)=
    '%ROUTINE',
    (TYPE)(FM);
P(FM)=
    '%FN',
    '%MAP';
V(FPDEL)=
    (RT)(%NAME'),
    (TYPE)(%QNAME'),
    '%NAME',
    '%RECORD'(%ARRAY')'%NAME';
V(LABEL)=
    (N),
    (NAME);
P(ELABEL)=
    (N),
    (NAME)(APP);
P(%ARRAY')=
    '%ARRAY',
    0;
P(%NAME')=
    '%NAME',
    0;
P(%QNAME')=
    '%ARRAYNAME'
    '%NAME',
    0;
V(FPP)=
    '('(FPDEL)(NAMELIST)(RESTOFFPLIST)')',
    0;
V(RESTOFFPLIST)=
    (,')(FPDEL)(NAMELIST)(RESTOFFPLIST),
    0;
P(ENDLIST)=
    '%OFPROGRAM',
    '%OFFILE',
    '%OFLIST',
    '%OFMCODE',
    0;
P(%FORMAT')=
    '%FORMAT',
    0;
P(RSTMNT)=
    '%FORMAT'(NAME)'('(RFDEC)(RESTOFRFDECLN)')',
    '%SPEC'(LEAVE HOLE)(NAME)(ENAME'')(SET MARKER)'('(NAME)')',
    (LEAVE HOLE)(DECLN)(SET MARKER)'('(NAME)')';
V(SC)=
    (EXPR)(COMP)(EXPR)(RESTOFSC),
    '('(SC)(RESTOFCOND)')';
```

```
P(RESTOFSC)=
    (COMP)(EXPR),
    0;
V(RESTOFCOND)=
    '%AND'(SC)(RESTOFANDC),
    '%OR'(SC)(RESTOFORC),
    0;
V(RESTOFANDC)=
    '%AND'(SC)(RESTOFANDC),
    0;
V(RESTOFORC)=
    '%OR'(SC)(RESTOFORC),
    0;
P(RESTOFUI)=
    (ASSOP)(EXPR),
    0;
V(ASSOP)=
    '==',
    '=',
    '<-',
    '->';
P(%SPEC')=
    '%SPEC',
    0;
P(%LN)=
    '%LONG',
    '%NORMAL';
P(RESTOFBPLIST)=
    ','(EXPR)':'(EXPR)(RESTOFBPLIST),
    0;
P(CORP)=
    '%PERM'(TEXTTEXT),
    '%IOCP'(TEXTTEXT),
    '%MAINEP'(TEXTTEXT);
P(DECLN)=
    (%QNAME')(NAMELIST),
    '%ARRAY'(%FORMAT')(ADECLN);
P(ADECLN)=
    (NAMELIST)(BPAIR)(RESTOFARRAYLIST);
P(RESTOFARRAYLIST)=
    ','(ADECLN),
    0;
P(OWNDEC)=
    (LEAVE HOLE)(NAMELIST)(CONST')(SET MARKER)(RESTOFOWNDEC)(S),
    '%ARRAY'(NAME)(CBPAIR)(=')(CONSTLIST),
    (NAME)(CONST')(S);
P(RESTOFOWNDEC)=
    ','(LEAVE HOLE)(NAMELIST)(CONST')(SET MARKER)(RESTOFOWNDEC),
    0;
P(XOWN)=
    '%OWN',
    '%EXTERNAL',
    '%EXTRINSIC',
    '%CONST';
P(CBPAIR)=
    '('(PLUS')(ICONST)':'(PLUS')(ICONST)')';
```

```
P(RESTOFNLIST)=
    ','(N)(RESTOFNLIST),
    0;
P(FLIST)=
    (N)(RESTOFNLIST)'->'(LABEL)(RESTOFFLIST);
P(RESTOFFLIST)=
    ','(FLIST),
    0;
P(MON)=
    '%STOP',
    (N),
    0;
P(RESTOFSWLIST)=
    ','(NAMELIST)(CBPAIR)(RESTOFSWLIST),
    0;
V(COMP)=
    '='
    '>=',
    '>',
    '#',
    '<=',
    '<',
    '¬=',
    '->';
P(RESTOFSS1)=
    ':',
    (%IU)(SC)(RESTOFCOND)(S),
    (%WU)(SC)(RESTOFCOND)(S),
    (S);
V(RESTOFIU)=
    '%START',
    '%THENSTART',
    '%THEN'(LEAVE HOLE)(UI)(SET MARKER)(ELSE');
P(RESTOFWU)=
    '%CYCLE',
    '%THENCYCLE',
    '%THEN'(UI);
P(AUI)=
    '%AND'(UI),
    0;
V(ELSE')=
    '%ELSESTART',
    '%ELSE'(UI),
    0;
P(ENAME'')=
    ' '(NAME),
    0;
P(ENAME')=
    ' '(NAME)(APP)(ENAME'),
    0;
P(BPAIR)=
    (CBPAIR),
    '('(EXPR)':'(EXPR)(RESTOFBPLIST)')';
P(CONST')=
    '='(+')(CONST),
    0;
```

```
P(%SEX')=
    '%SYSTEM',
    '%EXTERNAL',
    '%DYNAMIC',
    0;
P(CYCPARM')=
    (NAME)(APP)(ENAME')'='(EXPR)','(EXPR)','(EXPR),
    0;
P(OWNSTMT)=
    (TYPE)(OWNDEC),
    '%RECORD'(NAMELIST)'('(NAME)')'(S),
    '%RECORD' '%ARRAY'(LEAVEHOLE)(NAME)(CBPAIR)(SETMARKER)'('(NAME)')'(S);
P(RESTOFRFDECLN)=
    ','(RFDEC)(RESTOFRFDECLN),
    0;
P(RFDEC)=
    (TYPE)(%QNAME')(NAMELIST),
    (TYPE)'%ARRAY'(NAMELIST)(CBPAIR)(RESTOFSWLIST),
    '%RECORD'(%ARRAY')'%NAME'(NAMELIST),
    '%RECORD'(NAMELIST)'('(NAME)')',
    '%RECORD' '%ARRAY'(LEAVEHOLE)(NAMELIST)(CBPAIR)(RESTOFSWLIST)
                      (SETMARKER)'('(NAME)')';
P(UCS)=
    (NAME)(DISP'),
    '<'(LABEL)'>',
    '*'(DISP');
P(DLB)=
    (N)'('(N)','(N)')',
    (UCS)'('(N)')';
P(DXB)=
    (UCS)(RESTOFUCS),
    (N)(RESTOFDXB);
P(RESTOFDXB)=
    '('(N)','(N)')',
    '('(N)')',
    0;
P(RESTOFSI)=
    ','(ICONST),
    0;
P(DB)=
    (UCS),
    (N)(RESTOFUCS);
P(RESTOFUCI)=
    (UCRR)(N)','(N),
    (UCRX)(N)','(DXB),
    (UCRS)(N)','(N)','(DB),
    (UCSI)(DB)(RESTOFSI),
    (UCSHIFT)(N)','(DB),
    (UCSS)(DLB)','(DB),
    (UCPD)(DLB)','(DLB),
    (UCSPEC)(N);
P(DISP')=
    '+'(N),
    '-'(N),
    0;
```

```
V(UCI)=
    '*'(N)','(@')(NAME)(APP)(ENAME'),
    '->' 'P'(N),
    (UCINM)(RESTOFUCI),
    'PUT '(ICONST),
    'USING*,'(N),
    'DROP'(N);
P(RESTOFUCS)=
    '('(N)')',
    0;
P(=')=
    '=',
    0;
P(@')=
    '@',
    0;
V(UI)=
    (LEAVE HOLE)(NAME)(APP)(ENAME')(SET MARKER)(RESTOFUI)(AUI),
    '->'(ELABEL),
    '%PRINTTEXT'(TEXTTEXT)(AUI),
    '%RETURN'
    '%RESULT'(ASSOP)(EXPR),
    '%MONITOR'(MON),
    '%STOP',
    '%EXIT';
V(SS)=
    (LEAVE HOLE)(UI)(SET MARKER)(RESTOFSS1),
    '%CYCLE'(CYCPARM')(S),
    '%REPEAT'(S),
    (N)':',
    (%IU)(LEAVE HOLE)(SC)(RESTOFCOND)(SET MARKER)(RESTOFIU)(S),
    (%WU)(LEAVE HOLE)(SC)(RESTOFCOND)(SET MARKER)(RESTOFWU)(S),
    (TYPE)(DECLN)(S),
    '%END'(ENDLIST)(S),
    (LEAVE HOLE)(%SEX')(RT)(SET MARKER)(%SPEC')(NAME)(FPP)(S),
    (CMARK)(TEXT)(S),
    '%REALS'(%LN)(S),
    '%BEGIN'(S),
    '%FAULT'(FLIST)(S),
    '%FINISH'(ELSE')(S),
    '%SWITCH'(NAMELIST)(CBPAIR)(RESTOFSWLIST)(S),
    '%LIST'(S),
    (XOWN)(OWNSTMT),
    '%SPEC'(NAME)(FPP)(S),
    '%SPECIAL' '%NAME'(NAME)(S),
    '%TRUSTED' '%PROGRAM'(S),
    '%REGISTER'(NAME)(RESTOFUCS)(S),
    '%MCODE'(S),
    '%SHORT' '%ROUTINE'(S),
    '*'(UCI)(S),
    '%CONTROL'(CONST)(S),
    '%DEFINE'(CORP)(S),
    '%RECORD'(RSTMNT)(S),
    (S),
    (LABEL) ':';
```

The built-in phrases are

NAME
ICONST
CONST
CONSTLIST
N
S
TEXT
TEXTTEXT
NAMELIST
LEAVEHOLE
SET MARKER
UCRR
UCRX
UCRS
UCSI
UCSHIFT
UCSS
UCPD
UCSPEC
UCINM

The macro scheme also has a built in alternative literal facility. This allows differing conventional hardware representations of a particular conceptional input symbol to be equivalenced. Whenever the macro scheme is searching for the master literal it accepts any of the allowed alternatives in its place. At present the scheme accepts the alternative '¬=' to the master literal '#'. Note that whichever alternative is recognised, it is always the master literal which is output.

SECTION 2 - MACRO DEFINITIONS

A macro definition consists of two parts; the macro specification and the macro body. The macro specification (MACRO SPEC), which is terminated by the first occurrence of a newline character, has the following form:

<front'> %MACRO <spec'> (<phrase being extended>): <new alternative to phrase>

The macro spec serves three purposes. Firstly it specifies which standard IMP phrase is being extended, whether the alternative is being attached before or after the normal alternatives and whether the macro body follows immediately or is described later. Secondly it describes in detail how to recognise a call on the macro, by specifying which text literals and which macro parameters to look for and in which order; that is, it completely specifies the new phrase alternative. Thirdly, it specifies both the name and type of the macro parameters, for subsequent use in the macro body.

THE MACRO SPEC

A macro spec is introduced by the word %MACRO optionally preceded by the word %FRONT and optionally followed by the word %SPEC. In the event of the word %FRONT occurring, the new alternative is added to the front of the phrase being extended (the ELASTIC PHRASE) and precedes all previous alternatives, otherwise the new alternative is added to the end of the phrase and follows all previous alternatives. In the event of the word %SPEC occurring, the new alternative is attached but no macro body is compiled at this point. The macro may be described later in the normal way when the macro body is connected with the appropriate alternative. The macro must not be executed unless the macro body has been described.

The name of the phrase being extended must follow the <spec'> option enclosed in round brackets, however, if the IMP phrase source statement, V(SS), is being extended this field may be omitted. This field is followed by a colon which merely acts as a separator. Everything which follows to the right of the colon up to the first newline character, specifies the new alternative to the elastic IMP phrase.

The elements which may be used in the specification of the new alternative are TEXT LITERALS and MACRO PARAMETERS.

LITERALS

Text Literals are sequences of characters enclosed in single quotes as in normal IMP character strings.

e.g. '%GOTO' , '('

Note that before being placed in the macro definition, literals are converted to language mode; that is, spaces are removed and underlines are inserted where

appropriate. (Underlining is specified as in the IMP language by the % character.) Two consecutive literals must be separated by a full stop since two consecutive single quotes represent one quote as in IMP strings.

e.g. '%READ'.'('

PARAMETERS

Macro parameters differ fundamentally from routine parameters and variables in IMP. They correspond, not to locations or numbers, but to character strings. A parameter is specified by giving a mnemonic name, by which it is to be identified within the macro body, followed by a description of its type enclosed in round brackets.

e.g. A(NAME), B(OPERAND)

A list of permitted types and their usage is given in Section 8.

The type refers to a definition in the IMP phrase structure. Each such definition refers to a possible set of character sequences. For example, phrase (NAME) corresponds to the set of all IMP identifiers, phrase (CONST) to character strings such as

```
23
X'1019'
M'AF'
B'1001100101101'
'NL'
```

Assigning a type to the parameter has the following effects:

It states first of all which character strings are permissible in place of the formal parameter in a macro call. For example, if the following macro is specified

```
%MACRO : '%OUTPUT' C(CONST)
```

then legitimate calls on the macro are

```
%OUTPUT 50
%OUTPUT M'ABCD'
```

but not, for example

```
%OUTPUT XYZ
```

because XYZ is a name and not an example of phrase (CONST).

Secondly it has the effect that during the description of the macro, the parameter may be used, enclosed in pound signs (£), whenever the IMP syntax permits the character string of the type assigned to the parameter. Taking the above example, since

```
B = 2
```

is a legitimate IMP statement, the 2 may be replaced by a parameter, thus:

```
B =£C£
```

C may also be used in any other context where constants are allowed.

This string will be substituted during the call in all text-generating statements whenever the corresponding formal parameter was used.

QUALIFICATION OF PHRASE NAMES AS PARAMETERS

A phrase name by itself is not a complete specification of the alternatives to be used in recognising the parameter since new alternatives may be added to the phrase concerned, later. The user may wish to exclude certain alternatives from the recognition process for the parameter. This is achieved by preceding the phrase name by one of the words %CURRENT, %THIS, or %BASIC. If %THIS is used then all alternatives are excluded which are subsequently added to the front of the phrase concerned. If %CURRENT is used all such alternatives including the alternative currently being defined (if this is being added to the same phrase) are excluded. If %BASIC is used all alternatives are excluded which have at any time been added to the front of the phrase concerned.

e.g.
```
%MACRO : '%PLUS' A(%CURRENT OPERAND)
%FRONTMACRO(UI) : '%DO' INSTRUCTION(%BASIC UI)
```

whence for example

```
%DO %DO X = X + 1
```

is not a valid statement in the extended language.

Further the subsequent declaration of the macro

```
%FRONT MACRO (OPERAND) : '%TWO'
```

would not make the following statement valid.

```
%PLUS %TWO
```

THE MACRO BODY

The second part of the definition of a macro is the macro body. The body of the macro contains a sequence of MACRO STATEMENTS which control the text which is generated to replace the macro call. The actual generation of text is effected by TEXT REPLACEMENT STATEMENTS and the order of execution of these statements is controlled by tests and jumps called MACRO CONTROL STATEMENTS. This control is exercised at compile time and affects the form of the final program, not its running. In particular it is to be noted that the EXECUTION of a macro statement occurs at compile time.

Section 4 describes all the macro statements which are provided in the macro scheme, and section 6 gives examples of how to write various types of macro.

THE ORDER OF SEARCH

Macro extensions to any IMP phrase are added on the end of all the existing alternatives in the order in which they are specified. If the compiler fails to recognise its input as an example of the phrase concerned it tries the macros.

For this reason, if a macro is defined such that an attempted call on it is recognised as some previous alternative or macro of the phrase, then it will not be called. For example, calls on

```
%MACRO : A(NAME) '=' B(NAME)
```

are strings like

```
FRED = JIM
AB1C2 = X
```

which will be recognised as assignment statements by the compiler and not calls on the macro.

Care should be taken to avoid defining two macros which start off the same way; even if the compiler is able to distinguish calls on the two, recognition time will be considerably increased.

Macros, of course, may be added to the front of existing phrase definitions if desired by preceding the entire spec with the word %FRONT. The resulting macro is added before the existing phrase alternatives and macros. It is sometimes necessary to use 'current' versions of a phrase in macro definitions: i.e. to exclude from the search any further front macros which may be added. Typical of this is the following macro to add a new operator %EXP to the existing ones:

```
%FRONT MACRO (OPERAND) : A (OPERAND) '%EXP' X (OPERAND)
```

This will result in the compiler going into a recursive loop when attempting to detect an (OPERAND) in its input.

To avoid this, the type description should be preceded by the word %CURRENT, which excludes from the search (for the corresponding actual parameter) all future front macros added to the phrase concerned.

```
%FRONT MACRO (OPERAND) : A (%CURRENT OPERAND) '%EXP' X (OPERAND)
```

SECTION 3 - PHRASE DEFINITIONS

The programmer need not confine himself in choice of macro parameters to those permitted by the basic IMP syntax. The macro scheme permits the user to synthesise new phrase definitions for use as parameters, and provides macro statements to break these down into constituent parts and examine their structure.

A phrase definition has the following form:

%PHRASE (<mnemonic identifier>) : <alternative list>

It is introduced by the word %PHRASE and is followed by a user-chosen name for the phrase being defined, enclosed in round brackets. The phrase name is followed by a colon which merely acts as a separator. Everything to the right of this colon up until the first newline which is not immediately preceded by a comma is taken as the alternative list. An alternative list is a sequence of ALTERNATIVES, separated by commas. An ALTERNATIVE is a sequence of LITERALS and/or PHRASES, or the word %NULL. A LITERAL is, as in macro specs, a sequence of characters enclosed in single quotes. A PHRASE is a phrase name enclosed in round brackets. It is not preceded, as in a macro spec, by a symbolic identifier.

```
e.g.  %PHRASE (ABC) : (NAME),(CONST)
      %PHRASE (NEWOP) : '+','-',
                        '*',
                        '/'
```

A user defined phrase may be used to define other phrases and to define macros exactly in the same way as an IMP phrase.

Of course, no user-defined phrase will be accepted as a parameter in a text-replacement statement since it is not defined in the IMP syntax (unless the statement is a macro call). Before use as parameters in text-replacement statements, user defined parameters must be broken down into components which are defined in the IMP syntax. (See section on macro statements).

The compiler attempts to identify an example of a given user-defined phrase in the input stream by testing each alternative in turn. For each alternative the items are tested against the input stream, in sequence, as follows:

Each literal is tested symbol by symbol against the characters of the input stream.

Each phrase is tested for in the input string again by testing each alternative in turn.

A null alternative is always accepted.

If an item in the alternative matches the input string, the compiler proceeds to search for the next item in the succeeding input characters. If an item fails to match, then the compiler abandons the attempt to match the current alternative, goes back to the position in the input stream it was at when starting the phrase, and tries the next alternative.

If an alternative is all matched against the input stream the phrase succeeds; if no alternative matches the phrase fails.

It should be noted that basic IMP phrase names used in user phrase definitions may be qualified by the prefixes %BASIC, %CURRENT and %THIS as described for macro definitions.

SECTION 4 - MACRO STATEMENTS

The macro body is a sequence of macro statements terminated by the statement

%ENDMACRO

Macro statements govern the generation of the substitute text. There are three types of macro statement: declaration, text replacement and control statements.

Macro declarations are declarations of local phrase variables which, when assigned to, may be used exactly as macro parameters which are simply phrase variables initialised by the macro call itself.

LOCAL PHRASE VARIABLES

The user declares further phrase variables using the statement

%LOCAL N(P)

where N is a mnemonic identifier for the variable and P is the name of a phrase.

GENERATED PHRASE VARIABLE

The user may generate private macro names uniquely for future use in the macro, by using the statement

%GENERATE N(P)

where N is a mnemonic identifier for the variable and P is the name of a phrase. This statement is necessary for the in-line substitution of unique labels. The name generated has the form PMNn where n is an unsigned integer starting at 1.

The actual generation of text in a macro is accomplished by text replacement statements of which there are two types.

GENERAL TEXT REPLACEMENT

The basic text replacement statement is a colon, followed by an IMP source statement.

:<source statement>

When the macro statement is executed, the source statement is presented to the compiler as the next effective statement for compilation.

e.g. : WRITE(A,3)

More than one macro statement may be placed on a line, separated by semi-colons, but each text replacement statement must have the initial colon. Failure to insert the colon will result in a syntax error, or in the statement being interpreted as a macro control statement.

e.g. : WRITE(A,3); :-> 2

Note, also, that labels are source statements in their own right. If, therefore, a label is placed on the same line as another SS, the later must be preceded by a second colon.

e.g. : FRED: : WRITE (A,4)

TEXT REPLACEMENT IN PHRASE MACROS

Another text replacement statement exists for macros replacing phrases other than Source Statements. This statement has the form

%RESULT = C

where C is a syntactically valid example of the phrase being replaced by the macro. For example, if a macro extends phrase unconditional instruction, V(UI), %RESULT=%STOP is valid but %RESULT=%ENDOFPROGRAM is not valid. When the statement is executed C replaces the macro call and the macro will exit. Exits from macros other than through a %RESULT statement are faulted unless the macro extends V(SS).

Exits from phrase macros other than through a result statement will be faulted. However, a phrase macro may generate source statements in the normal way, and these will be generated so as to precede the source statement containing the macro call.

PARAMETERS IN TEXT REPLACEMENT STATEMENTS

A parameter of the macro being defined may be used anywhere in a text replacement statement, enclosed in pound signs, where it is syntactically valid. For example, if the following macro is defined

%MACRO : '%DISPLAY' C(ICONST)

then a valid text replacement statement would be

: WRITE (£C£,2)

Some care must be exercised, however. In certain cases a parameter may be used in a context where it will not be accepted by the syntax, although it apparently could be. For example, since IMP allows integer labels, such as

2:

it might be thought that since 2 is a constant and A is a constant, one could legitimately write

However, A may represent more general character strings than are permitted in this context, e.g. M'29' or 'B', and is therefore disallowed.

When the macro is called, some character string will be supplied as actual parameter (e.g. in the example above, %DISPLAY 25 is a call on the macro in which 25 is the actual parameter.).

When the statement is executed, the parameter will be replaced by whatever character string it currently stands for.

PARAMETERS IN TEXT

In certain parts of the language, text is handled as it stands, without being analysed (e.g. strings and comments).

Macro variables may be used anywhere in text without any check being made on their syntactic validity; they will simply be replaced, when the statement is executed, by the text of the character string they stand for.

It should be noted that spaces in comments are thrown away: those in strings and %PRINTTEXT are not.

MACRO CALLS INSIDE MACROS

If a macro is used within a text replacement statement, it will be recognised but not called. Whenever the statement is executed, the macro will be activated.

In the case of nested and sequential macro calls in a single text replacement statement, the order of activation is left to right, innermost call first.

The macro currently being defined is not added to the syntax until (and unless) the description is complete, and can therefore not normally be called recursively. If it is necessary to make a recursive call, a %MACROSPEC for the same macro must be given previously.

The order in which text replacement statements are processed is governed by macro control statements, such as tests and label jumps, in a similar manner to the statements of an IMP program. This control is exercised at compile time and affects only the form of the final program, not its running.

LABELS

A macro label has the following form

LABEL)

Any macro statement may be labelled with an alphabetic identifier followed by a right bracket. The use of the right bracket instead of a colon is to avoid confusion owning to a superabundance of colons. Integer labels are not permitted.

e.g.

```
        :WRITE (X + 1,2)
LAB1) :WRITE (X - 1,2)
```

UNCONDITIONAL JUMPS

An unconditional jump to a label is written exactly as in IMP,

```
-> L
```

where L is a label. It causes the macro to commence execution at the labelled statement.

e.g.

```
      -> LAB2
LAB1) :NEWLINE
LAB2) :NEWPAGE
```

The next three statements are provided to allow the detailed analysis of user defined phrases which have been recognised in the call of a macro.

SWITCH ON CATEGORY NUMBER

If a user-defined phrase is being used as a parameter, it is possible to test the category number of the phrase; i.e. to ascertain which alternative of the phrase was satisfied by the actual parameter when the macro was called.

The statement has the form

```
-> (L1,L2,...Ln) P
```

where L1, ..., Ln are label identifiers and P is a parameter name. If alternative r of P is satisfied, control is transferred to the statement labelled by Lr. If r > n, an error is given and the macro exits.

UNCONDITIONAL RESOLUTION

Suppose a phrase P has been recognised in the input and the resultant analysis is represented by the phrase variable V. When P was recognised, one of its alternatives succeeded; the statement described above, switch on category number, can test which one. However, all the elements of this alternative were also recognised in the input, since P was recognised.

It is of interest to know what character strings were recognised for each of the phrases contained in this alternative. The macro scheme permits the user to break P down into component parts by assigning each of these to a phrase variable. The resolution statement has the following form

```
N -> <Resolution list>
```

where N is a phrase variable and the resolution list must parallel exactly the phrase alternative concerned, except that in place of each phrase, the name of a phrase variable must occur. Two consecutive phrase variables must be separated by a full stop.

e.g.

```
EXPR -> OP1 '+' OP2
LIST -> EL.LIST
```

The effect is to assign the components of the string currently represented by N, to each of the phrase variables in the resolution list, according to the analysis given to this string by the recognition procedure.

This statement will fault at execution time if the alternative of P represented by N does not check with the alternative specified by the resolution list.

The statement will fault at macro definition time if the resolution list corresponds to no alternative of P.

CONDITIONAL RESOLUTION

There are two forms of this statement

```
-> L %IF N -> <Resolution List>
-> L %UNLESS N -> <Resolution List>
```

where N is a phrase variable of type P, and L is a label.

The first statement performs the resolution, and transfers control to label L, if the alternative of P held in N matches that specified by the Resolution List.

The second statement transfers control to L only if the match does not occur. Otherwise the resolution is performed.

The statement is faulted at compile time if the Resolution List corresponds to no alternative of P.

The present implementation of the macro scheme has no arithmetic or general conditional statements. The following three statements provide a limited method of achieving two of the most useful facilities which would be provided by macro arithmetic, namely, flag setting and a simple counter.

FLAG SETTING

This facility is provided by the following statement

```
%SETFLAG F %TO N
```

where F specifies which of the five flags, numbered 1 to 5, maintained by the macro scheme is to be set and N specifies the value which is to be set. N must lie in the range 1 to 255.

F is in fact a global stack of depth 9. Consequently up to nine levels of each flag may be set at the one time. The flag is unset by specifying N as 0 and this pops up the next cell of the flag. Since the stack is global, flag values are retained between macro calls and this provides a useful communication between macros.

FLAG TESTING

The macro scheme also provides the following statement to test the value of a flag setting. The statement has two forms.

```
-> L %IF %TESTFLAG F = N
-> L %UNLESS %TESTFLAG F = N
```

where L is a label, F is one of the five flags and N is the value which the flag is to be tested against.

The first statement transfers control to the label L if the current stack setting of flag F is equal to N. The second statement only transfers control if the flag setting does not equal N.

COUNTERS

The macro scheme provides the following statement to make use of a macro flag as a simple counter variable. The statement has two forms

```
%INCFLAG F
%DECFLAG F
```

The first form increments the value of flag F by 1 and the second form decrements the value by 1.

Note that a macro flag which has been 'set' cannot be used as a counter.

SECTION 5 - GLOBAL MACRO INSTRUCTIONS

In addition to the basic macro and phrase definitions the macro scheme provides various other facilities.

MACRO LISTING

The special instructions

```
%MACROLISTON and
%MACROLISTOFF
```

are provided to aid the programmer in checking whether the desired macro expansions have in fact taken place. The effect of specifying %MACROLISTON is that after every subsequent macro call until macro listing is again suppressed by %MACROLISTOFF, the generated IMP text resulting from the call of that macro is listed. The macro scheme assumes the default condition to be %MACROLISTOFF.

COMMENT PRESERVATION

The macro scheme normally treats comments exactly as the IMP compiler does, that is, it lists them and moves on to the next input statement. This has the added effect in the translater version of the macro scheme that comments do not appear in the output source file. However the instruction

```
%LISTCOMMENTS
```

is provided to allow the programmer to specify that comments are to be preserved and not thrown away.

SEQUENCE NUMBERING

With operating systems such as EMAS with on-line disc based file systems and sophisticated editors, the sequence numbering of source programs is unnecessary and indeed undesirable. If, however, a user wishes to have a copy of a program on cards, or to transport macro translator output to computing installations with less sophisticated systems, the macro scheme provides the instruction

```
%SEQUENCE
```

to produce a sequenced card image output source file.

TRANSLATION TERMINATOR

The macro scheme, in line with the IMP compiler, normally takes the input terminator to be either of the IMP statements %ENDOFPROGRAM or %ENDOFFILE. This is obviously not adequate when the source input is a complete redefinition of the IMP language. To take care of this situation and to allow files of macros to be test compiled before being submitted complete with extended IMP program, the macro scheme provides the terminating instruction

```
%ENDJOB
```

SECTION 6 - WRITING MACROS

The following examples illustrate some of the information given in the previous sections.

6.1 Simple Text Substitution

```
Definition    Spec    %MACRO : '%EXPLAIN'

              Body        : %PRINTTEXT'
                          INTEGER BECAME NEGATIVE
                          '
                          %ENDMACRO

Call                  %EXPLAIN

Result                %PRINTTEXT'
                      INTEGER BECAME NEGATIVE
                      '
```

6.2 Substitution with Parameters

```
Definition    Spec    %MACRO : '%DISPLAY' A(NAME)

              Body        : NEWLINE
                          : %PRINTTEXT' £A£ = '
                          : WRITE (£A£,4)
                          : NEWLINE
                          %ENDMACRO

Call                  %DISPLAY B

Result                NEWLINE
                      %PRINTTEXT'B='
                      WRITE(B,4)
                      NEWLINE
```

6.3 Extension of a Phrase other than Source Statement

```
Definition    Spec    %MACRO (OPERAND) : '%BYTEMASK'

              Body        %RESULT=X'FF'
                          %ENDMACRO

Call                  B = I & %BYTEMASK

Result                B = I & 255
```

6.4 Use of Generate Instruction

```
Definition   Spec    %MACRO : '%RS(' A(NAME) ')'

             Body          %GENERATE B(NAME)
                           : £B£ :
                           : READ SYMBOL (£A£)
                           : -> £B£ %IF £A£ =NL %OR £A£ = ' '
                           %ENDMACRO

Call                       %RS(P)

Result                     PMN1:
                           READ SYMBOL(P)
                           -> PMN1 %IF P=NL %OR P=32
```

6.5 Use of a Front Macro to redefine an existing language facility

This macro provides run time label tracing

```
Definition   Spec    %FRONTMACRO : L(LABEL ':' S(SS)

             Body          : £L£ :
                           : NEWLINE
                           : %PRINTTEXT'**LABEL £L£ REACHED**'
                           : NEWLINE
                           : £S£
                           %ENDMACRO

Call                       ENTER: X=20

Result                     ENTER:
                           NEWLINE
                           %PRINTTEXT'**LABEL ENTER REACHED**'
                           NEWLINE
                           X=20
```

6.6 Use of User defined Phrases and resolution in the provision of Boolean Algebra

Definitions

```
%PHRASE (NLIST) : (NAME)',' (NLIST),(NAME)
%PHRASE (BOOLEAN) : '%TRUE', '%FALSE'
```

Spec

```
%MACRO : '%BOOLEAN' NL(NLIST)
```

Body

```
    %LOCAL N(NAME)
    A1)
    -> A2 %UNLESS NL -> N ',' NL
    : %INTEGER £N£
    -> A1
    A2)
    NL -> N
    : %INTEGER £N£
    %ENDMACRO
```

Spec

```
%FRONTMACRO (UI) : A(NAME) '=' B(BOOLEAN)
```

Body

```
    -> L3 %IF B -> '%FALSE'
    %RESULT = £A£ = 1
    L3)
    %RESULT = £A£ = 0
    %ENDMACRO
```

Call

```
%BOOLEAN BA,BB
BA = %TRUE
```

Result

```
%INTEGER BA
%INTEGER BB
BA = 1
```

SECTION 7 - MACRO FAULTS LIST

As the macro scheme processes a source program, it may encounter errors in the syntax or semantics of certain statements. If this occurs a fault message is printed.

MACRO DEFINITION FAULTS

Fault Number	
1	PHRASE NOT DEFINED
2	PHRASE DEFINED TWICE
3	NAME NOT SET
4	NAME SET TWICE
5	LABEL NOT SET
6	LABEL SET TWICE
7	LABEL OUT OF CONTEXT
8	NOT A LABEL TYPE
9	ATTEMPT TO EXTEND BUILT-IN PHRASE
10	RESOLUTION DOOMED
11	ATTEMPT TO RESOLVE SYSTEM PHRASE
12	GENERATE PARAMETER NOT NAME TYPE

MACRO EXECUTION FAULTS

Fault Number	Message
50	RESOLUTION FAILS
51	MACRO NOT DESCRIBED
52	RESULT NOT SPECIFIED
53	USER FLAG DEPTH > 10
54	NO LABEL FOR CATEGORY NO

SECTION 8 - ELASTIC IMP PHRASE LIST

The IMP phrases which the programmer is allowed to extend by adding IMP macros, are listed below:

Phrase	Meaning	Examples
APP	Actual Parameter Part (i.e. the bracketed parameter list following an array or function identifier)	(2) (X,Y,2)
ASSOP	Assignment Operator	==
COMP	Comparator	>=
ELSE'	Conditional Alternative	%ELSE %START
EXPR	An IMP expression	X + Y - 2
FPDEL	Formal Parameter Delimiter	%INTEGERNAME
FPP	Formal Parameter Part (i.e. the bracketed list of parameter descriptions following a routine or function declaration)	(%REAL X)
LABEL	An IMP Label	FRED
OP	An operator	+
OPERAND	A simple constant, IMP variable, array element, function call or a bracketed expression	29,X B(29,31) (X + Y - 2)
REST OF APP	Rest of Actual Parameter Part	,2
REST OF COND	Rest of Condition	%AND Y = 3
REST OF EXPR	Defines that part of an expression which follows the first operand.	+1 (in the expression X+1)
REST OF FPLIST	Rest of Formal Parameter Part	, %INTEGER I
REST OF IU	Rest of %IF or %UNLESS statement	%THEN I = 1
SC	Simple Condition	X < = Y
SS	Source Statement	%BEGIN
TYPE	An IMP variable declarator	%BYTEINTEGER
UCI	User Code Instruction	USING *, 5
UI	Unconditional Instruction	%RETURN

All of the above IMP phrases, together with the built in phrases listed below, may be used in user macro and phrase definitions.

Built-in Phrases	Meaning	Example
NAME	An IMP variable name	TOTAL
ICONST	An integer constant	-1
CONST	A general IMP constant	X'FF'
CONST LIST	A list of IMP constants	4,-7,0
N	An unsigned integer	5
S	A separator	;
TEXT	General Text	THIS IS A COMMENT
TEXT TEXT	Text within quotes	'IN QUOTES'
NAME LIST	A list of IMP names	AV,COUNT,I

SECTION 9 - ACCESSING THE MACRO SCHEME

The macro scheme can be accessed by anyone who is an accredited EMAS user by appending the library ERCC11.MIMPLIB.

The text to text translator version of the macro scheme is accessed through the call MACRO IMP e.g.

MACRO IMP (<extended IMP source>,<output source>,<listing>)

where

<extended IMP source> is the input program file containing the user defined phrases and macros

<output source> is the name of the file into which the translated source is to be entered

<listing> is the name of the file or pseudo-device to which the translator listing is to be given

The two-pass compiler version of the macro scheme is accessed by the procedure name IMPM e.g.

IMPM (<extended IMP source>,<object code>,<listing>)

where

<object code> is the name of the file into which the compiled object code is to be placed

i.e. The parameters to IMPM are exactly the same as those to IMP or IMPS.

The extended input source may be specified as the concatenation of two files, although there is a small increase in efficiency if input is combined in one file.

e.g. MACRO IMP (MACROS + PROGRAMA, MACOUT, .LP)

The main limitations of the present implementation of the macro scheme and the major known faults are listed below.

REFERENCES

1) A Syntactic and Semantic definition of the IMP Language by P.D. Stephens, First Edition, August 1974.

2) A Syntactic Macro Scheme by A. Freeman, a paper submitted as part requirement of a Diploma in Computer Science, University of Edinburgh, June 1969.