AN EXPRESSION ANALYSIS PACKAGE FOR REDUCE

J.A. van Hulzen and B.J.A. Hulshof

Twente University of Technology
Departments of Computer Science and Applied Mathematics
P.O. Box 217, 7500 AE Enschede
The Netherlands

Abstract

An expression analysis package for REDUCE 2 is presented. This package, completely written in Standard LISP, can be considered as an extension of the algebraic mode. It allows to interactively dismantled and/or modify the last output expression as it is desplayed or printed. An interface with the Code Optimizer makes on line construction of optimized FORTRAN programs possible.

1. Introduction

The pleasant variety of substitution facilities and flags for controlling output style in REDUCE 2 [6] are largely due to its former life as special purpose package for doing computations in high energy physics [4]. Local changes in displayed or printed expressions are not only discouraged by a possible discrepancy between external form and internal representation but also by the limited possibilities to accomplish this with the COEFF-routine. Yet, being able to interactively dismantle and/or modify presented output can be quite instrumental for a better understanding of the problem currently being studied. Substitution is of course in general more complicated than just modifying output (see for instance [5,3,8,2,11,16,1]). However, creating the possibility of user controlled deprival of past and future of a presented expression, delivers a carte blanche for mimicking it. Such an "unconditional surrender" can easily be embodied in REDUCE 2, once its output mechanism is understood. The internal representation of the expression, a so called standard quotient [12], is employed, subject to the current output controlling flag configuration, to temporarily construct a prefix expression, named !*OUTP, which is actually used to produce the output in the normal mathematical infix notation. Hence capturing !*OUTP is a harmless intervention, delivering a simple "hands on" instrument to dismantle, analyze or modify presented output. This is, for instance, possible by associating it with a global variable, say !*OUTP!*, via a special flag. An alternative is to store it on the property list of a user chosen identifier. Both possibilities are implemented. Once having this hard-copy the substitution problem is in fact reduced to text-editing. Since this editing is achieved by operating on a prefix expression the modified expression, or selected portions of it, can easily be printed or displayed or assigned to a variable via the normal REDUCE 2 mechanisms [12]. The implemented facilities, which can be considered as an extension of the algebraic mode, can be used in combination with the already available routines for code optimization [17,7,15], for instance to interactively construct optimized FORTRAN programs.
In section 2 a description of the expression analysis commands is given. These facilities are illustrated in section 3 via a sample session. Hearn's problem of substitution [5] is reconsidered in section 4. In section 5 the code and its structure are shortly discussed.

2. Description of the expression analysis commands

We distinguish between control-, analysis-, selection-, localization- and replacement commands. Special read-commands, associated with the different expression analysis commands, cause these

latter commands to be translated into proper procedure calls, allowing either to return NIL or a special meaningful value as a side effect of the procedure-execution.
Before describing the syntactic construction of the commands and their semantic effect we introduce some name and syntax conventions. In subsection 2.8 an additional number of error messages and warnings is listed.

2.1 Name conventions

<op> denotes any operator, i.e. either a "built-in" operator, like QUOTIENT, TIMES, PLUS, MINUS and SIN, or a user defined operator.

<id> denotes any scalar variable.

<opid> stands for an operator, with an arbitrary number of arguments, which is associated with (selected portions of) the expression currently being analysed by adding an extra "dimension", i.e. an extra argument.

<exp> denotes a legal REDUCE 2 expression, presented according to the current output-flag configuration.

<ilst> stands for a sequence of non-zero positive integers, separated by comma's, e.g. 1,3,4 for example.

<slst> denotes a sequence of operators T,F, and A (standing for term, factor and argument, respectively) with one non-zero positive integer argument, and separated by comma's. Such a user constructed sequence defines which part of an exp is subjected to user actions, e.g. T(1), F(1), A(2) stands for the second argument of the first factor of the first term.

2.2 Syntax conventions

An expression analysis command has a name in brackets <....> by which it is known and is defined by what follows the metasymbol ::= . Each command definition consists of one or more "alternatives", separated by the | metasymbol. Optional constructs are enclosed by the meta symbols [...]. Uppercase lexemes and punctation are used to describe key constructs in the command definitions, required for a successful execution. Lowercase lexemes enclosed in <...> are names of other defining rules. If more than one occurrence of an item would cause ambiguity these entities are extended with non-zero positive integers.
Hence our commands are defined by

```
<expression analysis command>:: =
                    <control command>|
                    <analysis command>|
                    <selection command>|
                    <localization command>|
                    <replacement command>
```

The following subsections are used to introduce and discuss these alternatives.

2.3 Control commands

```
<control command> ::=
        SETSEL;| SAVESEL <id>;| GETSEL <id>;|
        REMSEL <idlst>; | CODOPTSEL <idlst>;
<idlst>::= <id> | <id>, <idlst>
```

The command SETSEL; or a combination of the SAVESEL and the GETSEL command is used to capture the prefix expression !*OUTP, corresponding with the most recently presented output. Concequently both alternatives can be considered as flags allowing to enter the analysis-mode. SETSEL; causes the value of !*OUTP to be assigned to the global variable !*OUTP!*, which is used as a recognizer during the analysis. Hence, execution of a new SETSEL-command implies that the previously assigned value of !*OUTP is automatically lost. The command SAVESEL idf; however saves the value of !*OUTP, by storing it on the property list of idf as property of an indicator, named SELOUT. It remains there for later use as long as the command REMSEL idf; is not yet typed in. This command allows the removal of an arbitrary number of output copies saved on the property list of the scalar variables listed in its idlst. Access to the property list of a variable idf to retrieve the value of !*OUTP is accomplished with GETSEL idf;. This command causes the property of SELOUT to be assigned to !*OUTP!*.
The additional command CODOPTSEL idlist; can be used to achieve an interface with the code optimizer. The value of the elements of idlist, saved on on their property lists as property of SELOUT,

are handed over to the code optimizer in a fastread-fashion. As stated in [15], chapter 2 (and illustrated in chapter 4.4) this fastread facility causes assignment statements to be read without evaluating identifiers on the righthand-side and without simplyfing these expressions. By default the fastread option is off and input to the code optimizer undergoes the normal REDUCE 2 treatment, implying that the "hands on" aspects of the expression analysis would be disturbed (see section 4 for an illustrative example). Error messages are provided to inform the user that the selected output happens to represent an atom or when it is not stored, as expected, on a property list.

2.4 Analysis commands

<analysis command>::=
NARGS TOP; | NARGS <slst>;|
LOP TOP; | LOP <slst>;|
HIGHPOW <exp> [<into part>];|
LOWPOW <exp> [<into part>];
<into part> ::= INTO <slst>

The command NARGS TOP; delivers the number of arguments with respect to the Leading OPerator of the output, currently being analysed. Similarly the command NARGS slist; delivers the number of arguments of the LOP of that portion of the expression which is specified by slist. The leading operator itself is returned as a consequence of executing the LOP command. An error message is printed when the slst-part or the complete expression happens to be an atom or even does not exist. The command HIGHPOW exph; returns the highest integer power of exph, assuming exph is part of the output expression being analysed. The exponent search is however restricted to the top-level. This is justified by the possibility to accomplish a lower level search by extending the command with an into part, resulting in a HIGHPOW search in the correnponding slst part of the expression. The non existence of an integer-power of exp is indicated by an error message. The LOWPOW-command is similarly defined.

2.5 Selection commands

<selection command>::=
SELTRMS <ilst> <specification part>;|
SELFACS <ilst> <specification part>;|
SELEX <exp> <specification part>;|
SELCEX <exp> <specification part>;|
SELPOWS <exp> <name part>;
<specification part>::=
[<from part>] [<name part>]
<from part>::= FROM <slst>
<name part>::= NAME <opid>

The commands SELTRMS and SELFACS can be used in different ways.
The ilst determines which terms (or factors), in the usual left to right order, have to be selected from the output expression and to be combined together to form a new sum (or product) expres-
When top level extraction is assumed the from part can be omitted; lower level extraction however can be achieved by adding it. Adding a name part allows to assign the selected terms (or factors) to an operator of the opid-type. The extra integer argument is provided by the corresponding ilst-element. The result of the(se) assignment(s) is however not automatically printed or displayed, but demands an additional user action. Apart from the leading operator, the prefix representation of sums and products is identical. Consequently both SELTRMS and SELFACS require identical activities. Assuming a user, who also has the LOP-facility at his disposal, does not intend to collect some factors(terms) of a sum(product) the only test, which is implemented, is one to see if the leading operator is indeed a PLUS or TIMES. If it differs an error message will inform the user about the failure. Notice that this implies that a MINUS is not allowed either. But in view of the slst construction it can hardly be qualified as a deficiency to consider MINUS as a unary operator. Error messages are given when the ilst is omitted or when one of its elements is out of range, i.e. when it is greater than the NARGS of the corresponding slst in the form part, for instance. When this slst portion or the expression itself is an atom or does not exist this is also reported by an error message.

The commands SELEX and SELCEX provide the possibility to extract portions of an expression, say sexp. If the structure of sexp is assumed to be expl*exp2[+exp3] then, under certain restrictions, the command SELEX expl; returns expl*exp2 and the command SELCEX expl; only exp2. Hence we presume the leading operator of sexp to be PLUS or TIMES. As a consequence an error message is produced if this presumption is violated. The restrictions are imposed by the question: "How to interprete the word subexpression?" We consider sums as linear expressions and products as monomials, however not in indeterminates but in constructs of the exp-type. With such a sum we can associate a sign sequence (of its integer coefficients). Then a sum s2 is viewed as a subexpression of a sum s1 if the subsums of s1-s2 and s2, formed by those terms, which contain "generalized indeterminates", appearing in both s1-s2 and s2, have identical sign-sequences, e.g. expx+3*expy is a subexpression of expx+4*expy but not of expx+2*expy or expx-4*expy. A monomial m2 is a subexpression of a monomial m1 if m2 divides m1 exactly. Hence $expx^2*expy^2$ is a subexpression of $expx^2*expy^3$ but not of $expx^3*expy$. These restrictions imply that output presented by using a DIV and/or a RAT-flag causes ambiguity.
The use of a from part limits the SELEX and SELCEX activities to the corresponding slst part of the complete expression, again assuming that MINUS is a unary operator which have to be taken into account explicitly.
Extending these commands also with a name part causes the n different terms of exp2 to be assigned to the specified opid of this name part and this number n to be returned. The required ilst is assumed to be 1, 2, ...,n. Again it is the user's reponsibility to vizualize this list of coefficients Error messages are produced when the [slstpart of the] expression does not exist or is an atom.

The command SELPOWS sexp NAME opidf; can be used to collect all possible powers of sexp, occuring in the expression currently being analysed, using the opid-mechanism. The number of powers, which is found, is returned. It requires an additional user action to print or display this list of powers. Identical powers, occurring at different places in the output, are only counted once. It is obvious that the use of either the SELEX-or the SELCEX command suffices to vizualize the impact of one specific power. Omission of the name part is reported by an error message.

2.6 Localization commands

<localization command>::=
 LOCSUB <exp> NAME <id> <terminator>|
 TABLO <explst>;
<terminator>::= ;|$
<explst> ::= <exp>| <exp>, <explst>
The command LOCSUB lexp NAME idf $ returns the number n of occurrences of the subexpression lexp in the output, currently being analysed. In addition a numbered sequence of n slst's, identifying the different locations of lexp, is constructed. When a ; instead of a $ is used as terminator, then, instead of n, this sequence is presented. Each slst-item is given on a new line in the form i : slsti, where i=1, 2, ..., n. This information is essential for the replacement commands.
The LOCSUB-command applies an all-level search, which is almost equal to the earlier mentioned SELEX- and SELCEX strategy, albeit more extensive, as to include exp's functioning as operator argument. This extension has some implications for the structure of the slst's. When having, say ...SIN(exps) the last element of the corresponding slst is A(1). However the occurrence of exps in e^{exps} or 2^{exps} is denoted by an slst ending with A(2), since exps is now considered as the second argument of the operator EXPT, which stands for exponentiation.
Error messages are implemented to signal the absence of NAME or a wrong choise for the id. The role of this id is further explained, when discussing the replacement facilities.
The command TABLO explist; is mainly implemented to assist in developing an adequate strategy for substitution and code optimization. Assuming explist consists of n items, this command results in a list of n pairs of positive integers, separated by a colon. If i:j is such a pair then j denotes the number of occurrences of the i-th element of explist in the output currently being

analysed.

2.7 Replacement commands

<replacement command>::=
REPLACE <expl> BY <exp2> <localization part>;
<localization part>::=
|INTO <slst>| INLST <lcid>|INLST <lcid>(<ilst>)|
INLST -<lcid>(<ilst>)
<lcid>::= <id>

The localization part of the REPLACE command allows a variety of substitution approaches. Simply omitting it results in a toplevel parellel replacement of any occurrence of subexpression expl by exp2, i.e. repeated substitutions like suggested by REPLACE x BY x+1; do not take place. If the localization part has the form INLST lcid, where lcid is assumed to be previously introduced via LOCSUB expl NAME lcid; and applied to the same output expression, all occurrences of expl, as collected by LOCSUB and retrievable by the scalar variable name lcid, are replaced by exp2. This is only seemingly in accordance with a FOR ALL or a LET statement, since expl can be any legal REDUCE2 expression. The other alternatives for the localization part allow local substitutions. INTO slist-limits replacements to the portion of the output expression, which is specified by slist. INLST lcidf(ilist) defines replacements in those portions of the output expression, which are defined by the slst´s, generated by a previous LOCSUB expl NAME lcidf; command, and corresponding with the non-zero positive integers collected in ilist, i.e. a subset of all occurrences is employed for substitution. The last possibility defines a complementary action. Now replacements are done in all slst´s not mentioned in ilist. Assume 8 occurrences of expl are localized by LOCSUB expl NAME exp2. Then taking INLST exp2(1,2,3,4) or INLST -exp2(5,6,7,8) as localization part of a REPLACE command results in identical modifications of the output, currently being analysed.
Error messages are implemented to inform the user that specimen of <expl> or <exp2> are missing in the command or that the former is not found in the user specified part of the output. When the leading operator differs from PLUS or TIMES a replacement fails if the number of arguments of such an alternative operator ought to be changed by this command and an error message reports the failure. Again it ought to be mentioned that we tried to create fast facilities. The just described limitations can hardly be qualified as unbearable. A two stage replacement easily allows to accomplish the user desired form.

2.8 Error messages and warnings

When ???? appears in the description of an error message some output dependent information is meant, and also typed, when occurring in practice.

"***** NO OUTPUT GIVEN FOR SELECTION"
No output available for selection activities (use SETSEL or GETSEL).

"*****???? IS NOT AN OPERATOR"
The ???? - part of the expression is not an operator with arguments.

"***** ???? IS NOT ALLOWED"
Only operators T,F,A, are allowed in the slst.

"***** CANNOT SELECT ON ARGUMENTS OF ????"
Only PLUS and TIMES are allowed as leading operators for this command.

"***** NAME MISSING"
This command needs a NAME-specification.

"***** EMPTY SEARCH-LIST"
No ilst given for SELTRMS or SELFACS.

"***** MISSSNG EXPRESSION"
expl and/or exp2 missing in the command REPLACE.

"***** ???? NOT FOUND"
expl of the command REPLACE is not found in the specified part.

"***** REPLACE FAILED"
Illegal attempt to replace an argument of an operator, which differs from PLUS or TIMES.

"***** CANNOT SELCT ON ????"

The output-expression is not an operator with arguments.

"***** NO SAVED OUTPUT ON ????"

There was not any output saved on ???? when calling GETSEL.

"***** ILLEGAL CONFIGURATION (PROBABLY ON DIV)"

Ambiguity, probably due to ON DIV and/or ON RAT, when executing for instance SELEX, SELCEX, LOCSUB or TABLO.

"*** NO INTEGER POWERS FOUND"

Warning indicating that no integer powers were found with HIHGPOW or LOWPOW (The program remains running).

"*** SUBEXPRESSION NOT FOUND"

Obvious warning (The program remains running).

```
%
% 3. A sample session.
%

*FLOAD SEL;

*IN TEST.EXP$

%
% The expression EX is used to illustrate the
% implemented analysis commands.
%

                2          2
              (X  + X*Y + Y )
EX :=  - E                  *SIN(X*Y) +

                    2
         (X*Y + 2*Y )              2
        E             *SIN(X*Y)*X  -

                    2
         (X*Y + 2*Y )              2
        E             *SIN(X*Y)*Y  +

                          2
         (SIN(X*Y)*X*Y + X )
        E                    *SIN(X)*COS(X) +

                 2
         (X*Y + Y )
        E           *SIN(X)

%-----------------------------------------
%          Control commands
%-----------------------------------------

*SAVESEL EXS;

%
% The prefixform of EX is stored on the property-
% list of EXS as property of the indicator
% SELOUT. This is easily verified with the
% instruction LISP CDR 'EXS;
%

*GETSEL EXS;

%
% The property of SELOUT, the prefix form of EX,
% corresponding with its presented form is
% assigned to !*OUTP!*. This is easily verified
% with the instruction LISP !*OUTP!*;
%
%-----------------------------------------
%          Analysis commands
%-----------------------------------------

*NARGS TOP;

5

*LOP TOP;

PLUS

*NARGS T(1);

1

%
% NARGS T(1) returns 1 since MINUS is considered
% as unary operator. Thus access to portions of
% abs(T(1)) is achieved via extensions of the
% slst T(1),A(1).
%

*LOP T(1);

MINUS

*NARGS T(1),A(1);

2

*LOP T(1),A(1);

TIMES

*HIGHPOW E;

*** NO INTEGER-POWERS FOUND

*HIGHPOW E INTO T(4);

*** NO INTEGER-POWERS FOUND

*LOWPOW Y;

2

*HIGHPOW Y INTO T(1),A(1),F(1),A(2);

2

%-----------------------------------------
%          Selection commands
%-----------------------------------------
```

```
*SELTRMS 4,5;

                          2
        (SIN(X*Y)*X*Y + X )
SIN(X)*(E                   *COS(X) +

                 2
        (X*Y + Y )
       E          )

%
% NARGS returns the total number of arguments.
% The SELTRMS ilst or SELFACS ilst however
% allows extraction of any user selected
% top level portion of an expression.
%

*OPERATOR T;

*SELTRMS 4,5 NAME T;

%
% Extending the command with a name part allows
% assignment to opid operators.The frompart allows
% lower level selection. An additional user action
% is needed to vizualize the effect, i.e. T(4);
% T(5); or A:=A1*F(1)+A2*F(2)+A3*F(3).
%

*T(4);

                   2
 (SIN(X*Y)*X*Y + X )
E                    *SIN(X)*COS(X)

*T(5);

         2
 (X*Y + Y )
E          *SIN(X)

*OPERATOR F;

*SELFACS 1,2,3 FROM T(4) NAME F;

*A:=A1*F(1)+A2*F(2)+A3*F(3);

                       2
      (SIN(X*Y)*X*Y + X )
A := E                    *A1 + SIN(X)*A2 + COS(X)*

     A3

%
% Selection of SIN(X)* its coefficient with SELEX
% and of the coefficient of SIN(X*Y), occurring
% in the third term of EX with SELCEX.
%

*SELEX SIN(X);

                          2
        (SIN(X*Y)*X*Y + X )
SIN(X)*(E                   *COS(X) +

                 2
        (X*Y + Y )
       E          )

*SELCEX SIN(X*Y) FROM T(3);
***** CANNOT SELECT ON ARGUMENTS OF MINUS

*SELCEX SIN(X*Y) FROM T(3),A(1);

           2
 (X*Y + 2*Y )  2
E             *Y

%
% Extraction of all different arguments of the
% exponential function E with SELPOWS. Notice
% that Y*(X+2*Y) occurs twice in EX.
%

*OPERATOR PE;

*SELPOWS E NAME PE;

4

*FOR I:=1:4 DO WRITE PE(I):=PE(I);

           2          2
PE(1) := X  + X*Y + Y

PE(2) := Y*(X + 2*Y)

PE(3) := X*(SIN(X*Y)*Y + X)

PE(4) := Y*(X + Y)

%----------------------------------------
%          Localization commands
%----------------------------------------

*LOCSUB X*Y NAME LXY;

1: T(5),F(1),A(2)
2: T(4),F(1),A(2),T(1),F(1)
3: T(4),F(1),A(2),T(1)
4: T(3),A(1),F(2)
5: T(3),A(1),F(1),A(2)
6: T(2),F(2)
7: T(2),F(1),A(2)
8: T(1),A(1),F(2)
9: T(1),A(1),F(1),A(2)
%
% The 9 occurrences of X*Y in EX are listed.
% 1: T(5),F(1),A(2) expresses that X*Y (i.e.
% (TIMES X Y)) occurs in the second argument
% of the first factor of the fifth term. This
% factor is (EXPT E (PLUS (TIMES X Y) (EXPT Y Z)))
% When using a $-symbol as terminator only
% seemingly the number of occurrences is given.
% However the INLST indicator SINXY is also
% available for replacement commands.
%

*LOCSUB SIN(X*Y) NAME SINXY$

4

%
% The TABLO command produces the number of
% occurrences of arbitrary subexpressions and
% delivers thus indications for possible opti-
% mizations of numerical code to be constructed.
%

*TABLO X*Y,SIN(X*Y);

1: 9
2: 4
```

```
%-----------------------------------------
%          Replacement commands
%-----------------------------------------

%
% The occurrences of X*Y in EX are known via LXY.
% The next two commands accomplish the replacement
% of X*Y by K in the last three terms of EX.
%

*REPLACE X*Y BY K INLST LXY(1,2,3,4,5);

   2                    2
 (Y  + K)            (2*Y  + K)  2
E        *SIN(X) - E           *Y *SIN(K) +

   2
 (X  + K*SIN(K))
E               *SIN(X)*COS(X) -

   2          2
 (X  + X*Y + Y )
E               *SIN(X*Y) +

           2
 (X*Y + 2*Y )           2
E            *SIN(X*Y)*X


*EX2:=REPLACE X*Y BY K INLST -LXY(6,7,8,9);

         2                    2
       (Y  + K)            (2*Y  + K)  2
EX2 := E        *SIN(X) - E           *Y *SIN(K) +

         2
       (X  + K*SIN(K))
      E               *SIN(X)*COS(X) -

         2          2
       (X  + X*Y + Y )
      E               *SIN(X*Y) +

                 2
       (X*Y + 2*Y )           2
      E            *SIN(X*Y)*X

%
% How to make errors ?
% EX3 ought to represent EX with all occur-
% rences of SIN(X*Y) replaced by L. But a
% $-symbol as terminator suppresses output, i.e.
% SAVESEL EXS2, results in storage of the prefix-
% from of EX2 on the property list of EXS2.
% The command TABLO L; illustrates this.
% Hence the selected expression EXS2 is removed
% with the REMSEL command an the previous
% situation restored with GETSEL EXS;
%

*EX3:=REPLACE SIN(X*Y) BY L INLST SINXY$

*SAVESEL EXS2;

*TABLO L;

1: 0

*REMSEL EXS2;

*GETSEL EXS;
```

```
%-----------------------------------------
% A potpourri to illustrate the possibilities
% for extraction of common subexpressions, as
% to increase understandibility, or in combination
% with code optimization facilities to construct
% portions of code for numurical evaluation.
%-----------------------------------------

*F1:=2*X*DF((SELFACS 1 FROM T(1),A(1)),X)+
*    DF((SELFACS 1,3 FROM T(2))/X,Y);

                2          2          2
       (X*Y + Y )        (X )       (X )
F1 := E          *X*(4*E    *X + 2*E    *Y +

            2          2
          (Y )       (Y )
         E    *X + 4*E    *Y)

*SETSEL;

*F2:=REPLACE E**(X*Y+Y**2) BY Z;

               2          2          2
             (X )       (X )       (Y )
F2 := X*Z*(4*E    *X + 2*E    *Y + E    *X + 4*

             2
           (Y )
          E    *Y)

*OFF ALLFAC;

*F2;

     2             2             2
   (X )  2       (X )          (Y )  2
4*E    *X *Z + 2*E    *X*Y*Z + E    *X *Z + 4*

   2
 (Y )
E    *X*Y*Z

*SETSEL;

*F3:=(SELCEX Z*E**(X**2))*UX+
*    (SELCEX Z*E**(Y**2))*UY;

           2                    2
F3 := 4*UX*X  + 2*UX*X*Y + UY*X  + 4*UY*X*Y

%
% Let us now try to to replace X**2+2*X*Y by S
%

*SETSEL;

*SUM:=X**2+2*X*Y;

        2
SUM := X  + 2*X*Y

*LOCSUB SUM NAME SM;
***** NAME MISSING
%
% This message is generated by the program,
% because REDUCE uses SUM in a special way.
%

*SXY:=X**2+2*X*Y;

        2
SXY := X  + 2*X*Y
```

```
*LOCSUB SXY NAME SM;

*** SUBEXPRESSION NOT FOUND

%
% This message indicates that SXY can only be
% found if UX and UY are factored out.
%

*FACTOR UX,UY;

*F3;

       2                  2
UX*(4*X  + 2*X*Y) + UY*(X  + 4*X*Y)


*SETSEL;

*LOCSUB SXY NAME SM;

1: T(2),F(2)
2: T(1),F(2)

*F4:=REPLACE SXY BY S INLST SM;

                   2
F4 := UX*(S + 3*X ) + UY*(S + 2*X*Y)

% When a numerical evaluation of F1 is
% desired we can write out :
% ------
% Z=EXP(Y*(X+Y))
% UX=Z*EXP(X*X)
% UY=Z*EXP(Y*Y)
% S=X*(X+2*Y)
% F1="F4"
% ------

*GETSEL EXS;

%
% To illustrate the code optimization facilities
% we single out the product of the first two fac-
% tors of the second term of EX, compute its first
% and second derivative with respect to Y and use,
% after some replacements, CODOPTSEL to create
% an optimized version of these three expressions
% assuming they all have to be evaluated for
% identical input.
%

*FY:=SELFACS 1,2 FROM T(2)$

*FY1:=DF(FY,Y)$

*FY2:=DF(FY1,Y)$

*CLEAR A,B,C;

*LET E**(X*Y+2*Y**2)=A,
*     SIN(X*Y)=B,COS(X*Y)=C;

*FY;

B*A

*SAVESEL CFY;

*FY1;

X*B*A + X*C*A + 4*B*A*Y

*SAVESEL CFY1;

*FY2;

   2                                              2
2*X *C*A + 8*X*B*A*Y + 8*X*C*A*Y + 16*B*A*Y  + 4*B

*A

*SAVESEL CFY2;

*FLOAD NSYS;

*CODOPT;

*ON FASTREAD;

*INIT(50)$

*CODOPTSEL CFY,CFY1,CFY2;

*CALC;

%
% The FLOAD NSYS, CODOPT, ON FASTREAD,
% INIT(50), and CALC commands are required
% for a proper use of the code opti-
% mization facilities, in accordance
% with [15], section 4.4.
%

NUMBER OF (+,-)-OPERATIONS : 6
NUMBER OF (*)-OPERATIONS : 24
10487 MS

BREUER-SEARCH :
64 MS

G0006:=B*A$
CFY:=G0006$
G0004:=4*G0006$
G0005:=A*X$
CFY1:=Y*G0004+C*G0005+B*G0005$
G0003:=Y*8*G0005$
CFY2:=G0004+16*Y**2*G0006+C*G0003+B*G0003+A*C*2*
X**2$

NUMBER OF OPERATIONS AFTER THE BREUER-ALGORITHM :
NUMBER OF (+,-)-OPERATIONS : 6
NUMBER OF (*)-OPERATIONS : 15
49 MS

Remark:
A new version of the code-optimizer, not yet
completely documented, allows FORTRAN output and
a more compact represention for common sub-
expression, i.e. G1 instead of G0001 etc. This
requires some extra commands :

*SCALAR GCOUNT;

*SYMBOLIC PROCEDURE NEWSYM;
*<<GCOUNT:=GCOUNT+1;
*  LIST('G,GCOUNT)
*>>;

*GCOUNT:=0;

0

*DOCONCAT G;
```

```
*INIT(50)$

*ON FORT;

*OFF PERIOD;

*CODOPTSEL CFY,CFY1,CFY2;

*CALC;

NUMBER OF (+,-)-OPERATIONS : 6
NUMBER OF (*)-OPERATIONS : 24
2002 MS

BREUER-SEARCH :
49 MS

      G4=B*A
      CFY=G4
      G2=4*G4
      G3=A*X
      CFY1=Y*G2+C*G3+B*G3
      G1=Y*8*G3
      CFY2=G2+16*Y**2*G4+C*G1+B*G1+A*C*2*X**2

NUMBER OF OPERATIONS AFTER THE BREUER-ALGORITHM :
NUMBER OF (+,-)-OPERATIONS : 6
NUMBER OF (*)-OPERATIONS : 15
120 MS
```

4. Hearns' problem of substitution reconsidered

The output, shown in figure 1a, was given by Hearn [5] as "a non-artificial example of the use of substitutions" (citing Moses [10], who also used it to illustrate his discussion about substitution as an aid to comprehension). The identities given in figure 1b were used to produce an equivalent form, shown in figure 1c. But is this example, stemming from "the dark Middle Ages of computer algebra", really illustrative? And can an expression analysis package really be instrumental for improving its comprehensibility? Although this example can indeed be qualified as non-artificial, reconsidering it certainly is artificial. Comprehensibility of output is strongly related to the origin of the problem and consequently to attempts to solve this problem in a structure preserving way. A quick inspection of figure 1a shows that all terms consist of 5 factors, when counting m^2 for one factor. Let us assume that this output is the result of some computation in high energy physics. Then it is obvious that by elementary operations on expanded polynomials, involving in fact certain matrices, cancelation, addition and substraction, due to simplification activities, are disturbing the problem structure. Instead of 5!=120 terms only 86 are left. Applying the given identities, in the given order (eventually with deletion of the first, since PQ does not occur in the output as a TABLO-command will show) and via straightforward LET-statements, shows that the coefficient of m^6 for instance is given by 8 * (RS-RT)(RS-QR). This is, of course, in accordance with figure 1c since 4 (-2 QR + 2 PR.RS/RT)= 4 (-2 QR.RT + 2 RS.(QR + RT -RS))/RT. Etc., etc. Such considerations are indeed artificial, only being attempts to retrieve the original structure of the problem. Our experience with NETFORM stipulates that this kind of problems requires structure preserving techniques [13,14]. To illustrate this, let us look again at the last example of the previous section. We had:

DF = exp(poly1). sin(x.y)

DF1 = poly2. exp(poly1). sin(x.y) + x. exp(poly1). cos(x.y)

DF2 = poly3. exp(poly1), sin(x.y) + $(poly2)^2$.exp(poly1).sin(x.y) + x. poly2. exp(poly1). cos(x.y) + ...

Here

poly1 = y(x + 2y)

poly2 = d(poly1)/dy

poly3 = d(poly2)/dy

Neglecting this simple polynomial-structure, as we did, implies that it is almost immediately lost, and consequently that code optimization (or understandibility) can be violated unnecessarily. In spite of these conclusions it might be illustrative to show what, in principle, can be achieved with code optimization features when applied on the output as given in figure 1a. The result is shown in figure 2. If we had not used the CODOPTSEL command, working on the presented output, influenced by a FACTOR M instruction, the result might have been different, since the internally expanded form embodies a high m^2 prevalency. Figure 2 also illustrates that (forthcoming) improvements of our code optimization facilities. (for sets of multivariate polynomials) are certainly worthwhile. We mainly reconsidered Hearn's "problem" to emphasize, that in our opinion, both expression analysis and code optimization are in principle on line instruments, which are mainly profitable when working in a problem structure preserving way.

```
    4                                  2          2             2
(2*M *( - PROP1*PR*RS + PROP1*PR*RT + 2*PR *RS + 2*PR *RT + 2*PR*RS  - 7*PR*RS*RT - PR*RS*PROP2 + 2*

                                                                                  2
       PR*RS*PS + 2*PR*RS*PT + 2*PR*RS*QT + 2*PR*RS*QS + 2*PR*RS*QR + 5*PR*RT  + PR*RT*PROP2 - 2*PR*

                                                                            2           2
       RT*PS - 2*PR*RT*PT - 2*PR*RT*QT - 2*PR*RT*QS + 2*PR*RT*QR + 3*RS *RT + 2*RS*RT  + 3*RS*RT*QR

            3       2        2                              2                             2
        + 3*RT  - 3*RT *QR) + M *(PROP1*PR*RS*RT - PROP1*PR*RT  - PROP1*PR*RT*PROP2 - PROP1*RS *RT

                    2                         3            2           2             2
     - 2*PROP1*RS*RT  + 2*PROP1*RS*RT*PT - PROP1*RT  - 2*PROP1*RT *PS - 6*PR *RT*QT + 2*PR *RT*QS -

         2
     4*PR *RT*QR + 4*PR*RS*RT*PROP2 - 4*PR*RS*RT*PS - 8*PR*RS*RT*PT - 4*PR*RS*RT*QT - 2*PR*RS*RT*QS

                                                                 3         2                2
     + 4*PR*RS*RT*QR - 8*PR*RS*PS*QT - 8*PR*RS*PS*QR + 4*PR*RT  - 2*PR*RT *PROP2 - 4*PR*RT *PT - 6*

           2           2               2
     PR*RT *QT + 4*PR*RT *QS - PR*RT*PROP2  + 2*PR*RT*PROP2*PS + 2*PR*RT*PROP2*QS + 8*PR*RT*PS*QT + 2

                                            2            2              3          2              2
     *PR*RT*PS*QR + 2*PR*RT*PT*QR - 2*RS *RT*QT - 4*RS *RT*QR + 4*RS*RT  + 2*RS*RT *PROP2 - 4*RS*RT *

                 2           2           2
     PS + 4*RS*RT *PT - 6*RS*RT *QT + 2*RS*RT *QS + 2*RS*RT*PROP2*PT - RS*RT*PROP2*QR - 4*RS*RT*PS*PT

                                        2                                   3          3          3
     - 2*RS*RT*PS*QR - 4*RS*RT*PT  - 4*RS*RT*PT*QT - 4*RS*RT*PT*QS - 4*RT *PS + 2*RT *QS - 4*RT *QR

           2               2               2  2       2            2            2            2
     - 2*RT *PROP2*PS + RT *PROP2*QR + 4*RT *PS  + 4*RT *PS*PT + 4*RT *PS*QT + 4*RT *PS*QS - 4*RT *

              2                               2
     PS*QR - 2*RT *PT*QR) + 2*RT*(PROP1*RS*RT*PS + PR *PROP2*QT - 4*PR*RS*RT*QT - PR*RS*PROP2*QT + 4*

     PR*RS*PS*QT + 4*PR*RS*PS*QR + 2*PR*RS*PT*QT + 2*PR*RT*PS*QT - 2*PR*RT*PS*QS - 2*PR*PROP2*PS*QT

                                 2                                                                2
     - PR*PROP2*PS*QR - 2*RS *PT*QT + 2*RS*RT*PS*QT + 4*RS*PS*PT*QT + 2*RS*PS*PT*QR - 4*RT*PS *QT))/

                    2
(4*PROP1*PR*RS*RT *PROP3)
```

figure 1a.

5. A short description of the code

The expression analysis package consists of approximately 1000 lines of code, including extensive comment. The code can be subdivided into three main sections.

1- A set of general service procedures, operating on prefix expressions, and designed to gain information about the structure of an expression, occurrence(s) of a subexpression and the like.
2- A set of special STAT-procedures for a syntactic check of the commands and a translation of these commands into proper procedure calls. This part of the code heavily relies on the design philosophy of REDUCE 2 and its sourcecode.
3- A set of procedures corresponding with the different commands. These are the procedures for which calls are prepared by the special STAT-procedures. This set, in turn, can be thought of as being subdivided into three subsets corresponding, respectively, with control commands, selection commands and analysis-, localization- and replacement commands.

Since the package is completely written in Standard LISP [9] it is fully portable. Once the package is installed and compiled the instruction FLOAD SEL; suffices for a user to make it operational. However, the restriction has of course to be made that the command CODOPTSEL is useless without having installed and compiled the Code Optimizer [15]. A user can obtain extra information about the use of the package by typing in SELCT.HLP;. The complete code and the text provided via the SELCT.HLP is given in [18].

```
NUMBER OF (+,-)-OPERATIONS : 95
NUMBER OF (*)-OPERATIONS : 296
4072 MS

BREUER-SEARCH :
4609 MS

G0039:=PS*4$
G0012:=RS*G0039$
G0001:=PR*G0012$
G0040:=2*QS$
G0018:=RT*G0040$
G0002:=PR*G0018$
G0031:=RS*RT$
G0004:=4*G0031$
G0003:=QT*G0004$
G0029:=2*QT$
G0005:=PS*G0029$
G0036:=2*QR$
G0006:=PS*G0036$
G0032:=PR*PS$
G0007:=RT*G0032$
G0033:=PR*RS$
G0008:=QT*G0033$
G0009:=PR*G0036$
G0010:=PT*G0031$
G0011:=QR*G0032$
G0013:=RT*G0029$
G0035:=2*PT$
G0014:=PR*G0035$
G0015:=2*G0033$
G0016:=PROP1*G0031$
G0024:=2*RT**2$
G0017:=PROP2*G0024$
G0034:=PR*PROP2$
G0019:=RT*G0034$
G0049:=RT**2*4$
G0020:=PT*G0049$
G0021:=QR*G0031$
G0022:=RS*G0024$
G0023:=QS*G0049$
G0030:=QT*6$
G0025:=RT**2*G0030$
G0026:=RT*QR*4$
G0027:=RT**2*G0039$
G0028:=4*PS**2$
G0037:=PR*RT$
G0038:=PR*RT**2$
G0041:=RT*PR**2$
G0042:=RT**3*4$
G0043:=QR*RT**2$
G0044:=RT*G0009$
G0045:=PT*G0004$
G0046:=RT*RS**2$
G0047:=2*G0010$
G0048:=2*G0007$
G0050:=PR*G0003$
EXS:=(2*RT*(-G0050+RS*PT*G0006+PT*QT*G0012+G0005*G0031+G0005*G0037+G0008*G0035+QR*G0001+QT*G0001+PR**
2*PROP2*QT+PS*G0016-PROP2*G0008-PS*G0002-G0005*G0034-PROP2*G0011-RS**2*PT*G0029-RT*QT*G0028)+M**2*
(-G0050+PS*G0023+QT*G0027+PS*G0020+RT**2*G0028+PROP2*G0043+RT**3*G0040+PROP2*G0047+QS*G0022+RS*G0020+
RS*G0017+RS*G0042+PT*G0044+G0006*G0037+QT*8*G0007+PROP2*G0002+PROP2*G0048+PR*G0023+PR*G0042+PR*QR*
G0004+G0004*G0034+PR**2*G0018+PROP1*G0047+PR*G0016-PROP1*G0038-PROP1*G0019-PROP1*G0046-PROP1*G0022-
PROP1*RT**3-PROP1*PS*G0024-G0030*G0041-PR**2*G0026-RT*G0001-PR*8*G0010-RS*G0002-PS*8*G0008-RS*8*G0011
-PR*G0017-PR*G0020-PR*G0025-PROP2**2*G0037-RS**2*G0013-RS**2*G0026-RT**2*G0012-RS*G0025-PROP2*G0021-
PS*G0045-G0006*G0031-PT**2*G0004-PT*G0003-QS*G0045-RT**3*G0039-QR*G0042-PS*G0017-QR*G0027-PT*QR*G0024
)+2*M**4*(3*RT**3+3*G0021+G0022+3*G0046+G0044+G0019+5*G0038+RS*G0009+QS*G0015+2*G0008+RS*G0014+PS*
G0015+2*PR*RS**2+2*G0041+2*RS*PR**2+PROP1*G0037-PROP1*G0033-PR*7*G0031-PROP2*G0033-G0048-RT*G0014-PR*
G0013-G0002-3*G0043))/
     (4*PROP1*PR*RS*RT**2*PROP3)$

NUMBER OF OPERATIONS AFTER THE BREUER-ALGORITHM :
NUMBER OF (+,-)-OPERATIONS : 95
NUMBER OF (*)-OPERATIONS : 160
592 MS
```

figure 2

```
PQ = M**2 - PROP1/2,

PR = QR + RT - RS,

PS = QS + RT - PROP1/2,

PT = QS - PR + RT,

QS = M**2 - PROP3/2,

QT = PS - QR - RT,

PROP2 = PROP1 - 2*RT + 2*RS
```

figure 1b.

```
((4*M**4 - (PROP1+PROP3)**2)*(- 2*M**2*QR - 4*QR*RT
        + 2*RT**2 - RT*(PROP1+PROP3)+(PR*PROP1+RS*PROP3)
        + 2*M**2*PR*RS/RT)
+ 4*M**2*QR*(PR + RS)*(2*M**2 + RT + (PROP1+PROP3))
+ 2*M**2*PR*RS*(2*QR - 6*RT - 3*(PROP1+PROP3))
+ 2*(QR - RT)*((PR*PROP1+RS*PROP3)*(M**2 - (PROP1+PROP3))
        + 2*QR*RT*(PROP1+PROP3))
+ 2*(QR**2 + RT**2)*(2*QR*RT - (PR*PROP1+RS*PROP3)
        + RT*(PROP1+PROP3)) + 6*M**2*RT**2*(PROP1+PROP3))
/ ( 4*PROP1*PROP3*RT*PR*RS)
```

figure 1c.

References.

[1] Cowan, R., Griss, M.: Hashing, the Key Rapid Pattern Matching. Symbolic and Algebraic Computation (E.W. Ng, ed.), Springer LNCS series nr.72, 266-278. Berlin-Heidelberg - New York: Springer-Verlag (1979).

[2] Golden, J.P.: The Evaluation of Atomic Variables in MACSYMA. Proceedings of the 1977 MACSYMA User's Conference, 109-122. Washington D.C.: NASA Report CP-2012. (1977).

[3] Fateman, R.J.: The User-Level Semantic Matching Capability in MACSYMA. Proceedings SYMSAM 2 (S.R. Petrick, ed.), 311-323. New York: ACM (1971).

[4] Hearn, A.C.: Computation of Algebraic Properties of Elementary Particle Reactions using a Digital Computer. Comm. ACM 9, 573-577 (1966).

[5] Hearn, A.C.: The Problem of Substitution Proceedings of the 1968 Summer Institute on Symbolic Mathematical Computation (R.G. Tobey, ed.), 3-20. Cambridge, Mass.: IBM (1969).

[6] Hearn, A.C.: REDUCE User's Manual. Univ. of Utah: Report UCP-19 (1973).

[7] Hulshof, B.J.A., van Hulzen, J.A., Smit, J.: Code Optimization Facilities applied in the NETFORM Context. Twente University of Technology; TW memorandum nr. 368 (December 1981).

[8] Kanoui, H., Bergman, M.: Generalized Substitutions. Proceedings Conference on Symbolic Comp. Methods and Applications, Saint Maximin, France (A. Visconti, ed.), 44-55. Marseille(1977).

[9] Marti, J., e.a.: Standard LISP Report, SIGSAM Bulletin 14,1, 23-43. New York: ACM (1980).

[10] Moses, J.: Algebraic Simplification: A Guide for the Perplexed. Proceedings SYMSAM 2 (S.R. Petrick, ed.), 282-304. New York: ACM (1971).

[11] Moses, J.: The Variety of Variables in Mathematical Expressions. Proceedings of the 1977 MACSYMA User's Conference, 123-130. Washington D.C.: NASA Report CP-2012 (1977).

[12] Norman, A.C.: Symbolic and Algebraic Modes in REDUCE. REDUCE-Newsletter 3, 5-9. Univ. of Utah: Symb. Comp. Group (1978).

[13] Smit, J. : A Cancelation free Algorithm with Factoring Capabilities, for the Efficient Solution of Large, Sparse Sets of Equations. Proceedings SYMSAC '81 (P.S. Wang, ed.), 146-154. New York: ACM (1981).

[14] Smit, J., van Hulzen, J.A.: Symbolic-Numeric Methods in Microwave Technology. Proceedings EUROCAM '82 (J. Calmet, ed.). Berlin-Heidelberg-New York: Springer Verlag LNCS Series (to appear).

[15] Smit, J., van Hulzen, J.A., Hulshof B.J.A.: NETFORM and Code Optimizer Manual. SIGSAM Bulletin 15,4, 23-32. New York: ACM (1981).

[16] Stoutemyer, D.R.: sin(x)**2 + cos(x)**2=1. Proceedings of the 1977 MACSYMA User's Conference, 425-234. Washington D.C.: NASA Report CP 2012. (1977).

[17] van Hulzen, J.A.: Breuer's Grow Factor Algorithm in Computer Algebra. Proceedings SYMSAC '81 (P.S. Wang, ed.), 100-104. New York: ACM (1981).

[18] van Hulzen, J.A., Hulshof, B.J.A. : An Expression Analysis Package for REDUCE. Twente University of Technology, Memorandum nr: INF-82-4 (July 1982).