Artificial Intelligence Project---RLE and MIT Computation Center

Memo 27 -- SIMPLIFY

by Tim Hart

1. General Description

simplify is a compilable set of 45 S-expression-defined functions which simplify algebraic expressions. The expressions which are appropriate for simplify are defined recursively as follows:

P = all atoms, fixed and floating point numbers

Q = all expressions of the form:

(PLUS, s_1, s_2, ..., s_n),
(PRDCT, s_1, s_2, ..., s_n),
(MINUS . s),
(RECIP . s),
(DIVIDE, s_1, s_2),
(POWER, s_1, s_2),
(SUBT, s_1, s_2) ,

where $s, s_1, s_2, \ldots, s_n \in P \cup Q$.

simplify is a function, not a pseudo-function, that is, the list structure of an expression is not modified by simplify. simplify takes about 6000 words of free storage when stored as an S-expression and about 9000 words when compiled. It takes about 5 minutes to read all the functions into the 709 using the online card reader, and about 4 minutes from tape.

2. Ordering

It is possible for two expressions having the same algebraic structure to be represented by different list structures, e. g. (PLUS 2 X) and (PLUS X 2). This creates a serious problem, for a LISP function to recognize equality of algebraic form would be quite complex and time consuming. Therefore the simplification routines use the device of canonically ordering all the algebraic expressions which they create. Thus it is possible to recognize equality of form with equal.

The ordering scheme used provides a hierarchy of S-expressions

This page is missing from the original document.

7. perform all the above simplifications on terms occurring in the denominator if the argument contains DIVIDE's and RECIP's.

simmin[x] will:
1. change the sign of x if it is a number
2. throw away the MINUS if x is of the form (MINUS . E)
3. reverse the order of SUBT expressions.

simrcp[x] will:
1. divide 1.0 by x if x is a number
2. yield cons[MINUS;simrcp[E]] if x is (MINUS . E)
3. throw away initial RECIP.
4. reverse the order of DIVIDE expressions, keeping numerical terms (if present) in the numerator
5. simplify each term in PRDCT expressions separately.

simdiv[x;y] = simprd[list[x;simrcp[y]]]

simpwr[b;e] will:
1. yield 1 if e is 0.
2. yield 0 if b is 0.
3. yield b if e is 1.
4. perform the proper numerical operation if imaginary numbers are not involved, e. g. $(-1)^3 = -1$, $(1)^{2.4} = 1$, $(-1)^{2.4}$ = (POWER -1 2.4).
5. yield cons[RECIP;simpwr[E;e]] if b = (RECIP E).
6. simplify each term individually in PRDCT or DIVIDE expressions.
7. treat a symbolic minus sign if possible (if e is an integer).
8. multiply exponents if b is itself a POWER expression.

simsub[x;y] = simpls[list[x;simmin[y]]]

4. Using Simplify

It is quite acceptable to use the subfunctions simpls[ℓ], simprd[ℓ], simmin[x], etc. separately, provided that the arguments of these functions have been simplified. In fact this is advisable, for if one lets these functions put the connectives

This page is missing from the original document.

abs[x] gets the absolute value of fixed or floating point numbers. If x is not a number an error will result.

divp[x;y] is primarily for fixed point numbers. If x is divisable by y or x is a floating point number it is T; F otherwise. If either x or y is not a number an error will occur.

gcd[x;y] is for fixed point numbers. It finds the greatest common divisor if $x \neq 0$ and $y \neq 0$, and x and y are fixed point numbers. If $x = 0$ gcd[x;y] = y, if $y = 0$ gcd[x;y] = x.

mapend[ℓ;fn] is the same as mapcon, except it appends resultant lists instead of nconc-ing them.

mapend[ℓ;p] p is a predicate in one variable. mapend is T if $p[x_1]$ is T for every $x_1 \in \ell$ ($\ell = (x_1,x_2,x_3,\ldots,x_n)$).

maketlg[e] is an output function - it turns the expression e into a list which will print out like Fortran. This is for convenience in reading the output.

greater[x;y] gives a unique ordering of atomic symbols. If x and y are atoms, greater[x;y] = T if y occupies a higher storage location that x.

larger[x;y] orders S-expressions uniquely. The relation between the different possible elements x and y is:

NIL < number < atom < list.

The order among numbers is by increasing magnitude, the atoms are ordered by greater (see above), and lists are ordered according to the first element, or by succeeding elements in the case when car[x] = car[y], and by length if corresponding elements are equal.

order[ℓ] = orders the top level of the list ℓ by the scheme given above.

Example: order[(X (A,B,C) (A) (A,B) 3 NIL 4)]
= (NIL 3 4 X (A) (A,B) (A,B,C))

insord[x;e] recopies the top level of ℓ, inserting x into the proper position to yield an ordered list.

All subfunctions have been tested on a limited number of trial cases. The connective functions are known to work on a large class of expressions, but some bugs may remain. Please bring any information about bugs to 26-265.