The Expression Evaluator of the Optimization Program INVERSE

(FOR VERSION 3.11)

Igor Grešovnik

Ljubljana, 27 September, 2005

Contents:

4. Expression Evaluator ***3***

4.1 Basic Properties of the Expression Evaluator **3**

4.2 Expression Evaluator Operators **4**

4.2.1 Expression Evaluator Binary Operators 4

4.2.2 Expression Evaluator Unary Operators (functions) 4

4.3 Pre-defined Functions of the Expression Evaluator **6**

4.3.1 random *[ ]* 6

4.3.2 If *[cond, exp]* 6

4.3.3 Else *[cond, exp1, exp2]* 6

4.3.4 Case *[ ]* 6

4.3.5 Min *[arg1, arg2, arg3, ...]* 6

4.3.6 Max *[arg1, arg2, arg3, ...]* 6

4.3.7 Sum *[arg1, arg2, arg3, ...]* 7

4.3.8 Prod *[arg1, arg2, arg3, ...]* 7

4.3.9 Trapint *[func[0.0], left, right, numsteps]* 7

4.4 Pre-defined Variables of the Expression Evaluator **7**

4.4.1 c_pi 7

4.4.2 c_e 7

4.5 File Interpreter Functions which Affect the Expression Evaluator **8**

4.5.1 = *{ var : expr }* 8

4.5.2 $ *{ var : expr },* $ *{ func[arg1, arg2, ...] : expr }* 8

4.5.3 definefunction *{ funcname [defblock] }* 8

4.5.4 return *{ expr }* 10

4.5.5 argument *[num]* 10

4.5.6 numargs *[ ]* 10

4. EXPRESSION EVALUATOR

4.1 Basic Properties of the Expression Evaluator

The expression evaluator, also referred to as calculator, is a system which evaluates mathematical expressions. These expressions consist of *numbers*, *operators* (unary and binary), *parentheses*, and *symbols*. Spaces are allowed between these expression parts, but they have no meaning in expressions. Symbols can be *variable or function names*. They are case sensitive, can consist of letters, digits and underscore characters, and must begin by a letter. There are some pre-defined functions of the expression evaluator, but functions can also be defined anew, either by the file interpreter or the expression evaluator itself. There are also some pre-defined variables, which represent mathematical constants. The expression evaluator variables differ from the user-defined variables of the shell.

Expressions can be formed in conventional way. When there are several binary operators in the expression, operators with lower piority number take effect first. When there are several binary operators with the same priority, the operators which appear before take effect first. The order of operations can be changed by round parentheses which group sub-expressions.

Variable and function names consist of letters and numbers. The first character of the name must be a letter. Difference is made between capital and small letters.

The expression evaluator functions can take one or more arguments. They must be listed in square brackets which follow the function name, and must be separated by commas. Arguments can be expressions which can be evaluated in the expression evaluator. Of course, these expressions can consist of a single variable or number.

Some expression evaluator functions can also take string arguments. This feature was actually added to support functions which access the shell's user defined variables. The use of strings as arguments is limited due to the fact that any mathematical expression evaluated in the expression evaluator can evaluate only to a real number, not to a string. Therefore, the expression evaluator functions defined by the **definefunction** command can not take string arguments.

4.2 Expression Evaluator Operators

4.2.1 Expression Evaluator Binary Operators

Operator	Meaning	Priority number
+	addition	4
-	substraction	4
*	multiplication	3
/	division	3
%	int. modulus	3
^	Power	2
P	power	2
CP	power with integer exponent	2
LOG	logarithm with arbitrary basis	2
MIN	lesser of both arguments	2
MAX	greater of both arguments	2
<	is lesser than	5
>	is greater than	5
!=	is not equal	5
<=	is lesser or equal	5
>=	is greater or equal	5
==	is equal	5
&&	and	6
||	or	6
:	definition	10
,	enumeration	9
=	assignment	10

4.2.2 Expression Evaluator Unary Operators (functions)

All unary operators of the expression evaluator have the priority 1.

Operator name	Meaning
EQ	equality
-	negative value

NEGV	negative value
exp	exponential function
ln	natural logarithm
sqr	square
sqrt	square root
abs	absolute value
sin	sine
cos	cosine
tg	tangent
ctg	cotangent
arcsin	inverse sine
arccos	inverse cosine
arctg	inverse tangent
arcctg	inverse cotangent
sh	hyperbolic sine
ch	hyperbolic cosine
th	hyperbolic tangent
cth	hyperbolic cotangent
arsh	inverse hyperbolic sine
arch	inverse hyperbolic cosine
arth	inverse hyperbolic tangent
arcth	inverse hyperbolic cotangent
st	converts radians to degrees
deg	converts radians to degrees
rad	converts degrees to radians
round	rounds its argument to the nearest integer
trunc	truncates its argument to the nearest integer below
floor	truncates its argument to the nearest integer below
int	truncates its argument to the nearest integer below
frac	returns the fractional part of the argument
sign	returns 1 if the argument is positive, -1 if it is negative, and 0 if it is 0
positive	returns 1 if the argument is positive and 0 otherwise
negative	returns 1 if the argument is negative and 0 otherwise
pospart	returns the argument if it is positive, otherwise it returns 0
negpart	returns the argument if it is negative, otherwise it returns 0.

4.3 Pre-defined Functions of the Expression Evaluator

Expression evaluator functions can have more than one argument. Their arguments must be in square brackets and separated by commas. The expression evaluator has some basic pre/defined functions:

4.3.1 random *[ ]*

Returns a random number between 0 and 1.

4.3.2 If *[cond, exp]*

Returns the value of *exp* if the value of *cond* is not 0, else it returns 0.

4.3.3 Else *[cond, exp1, exp2]*

If the value of *cond* is not 0, it returns the value of *exp1*, else it returns the value of *exp2*.

4.3.4 Case *[ ]*

4.3.5 Min *[arg1, arg2, arg3, ...]*

Returns the value of the least of its arguments *arg1*, *arg2*, etc. It requires at least one argument and can take unlimited number of arguments.

4.3.6 Max *[arg1, arg2, arg3, ...]*

Returns the value of the least of its arguments *arg1*, *arg2*, etc. It requires at least one argument and can take unlimited number of arguments.

4.3.7 Sum *[arg1, arg2, arg3, ...]*

Returns the sum of its arguments *arg1*, *arg2*, etc. It requires at least one argument and can take unlimited number of arguments.

4.3.8 Prod *[arg1, arg2, arg3, ...]*

Returns the product of its arguments *arg1*, *arg2*, etc. It requires at least one argument and can take unlimited number of arguments.

4.3.9 Trapint *[func[0.0], left, right, numsteps]*

Returns the integral of function *func* between *left* and *right* calculated by the trapezoidal rule with *numsteps* steps. The first argument must be a function call with an argument which is a real number. Such arrangement speeds up the calculation. *func* can be a pre-defined function of one argument, a function defined with the expression evaluator, or a function defined by the **definefunction** command.

Example of function use:

```
= { b : Trapint[ sin[0.0 ], 0, 1, 100 }
write { "Integral of the sine function between 0 and 1 is " $b ".\n\n" }
```

4.4 Pre-defined Variables of the Expression Evaluator

Expression evaluator pre-defined variables are used for keeping mathematical constants. They can be re-defined, but this should be avoided.

4.4.1 c_pi

Holds the value of $\pi = 2 \cdot \arcsin(1) \approx 3.141592654$ (ratio between circle circumference and diameter).

4.4.2 c_e

Holds the value of $e \approx 2.718281828$ (basis of natural logarithm).

4.5 File Interpreter Functions which Affect the Expression Evaluator

4.5.1 *= { var : expr }*

The = function assigns the value of the expression *expr* to the expression evaluator variable *var*. If a variable named *var* does not yet exist, it is created.

4.5.2 **$** *{ var : expr },* **$** *{ func[arg1, arg2, ...] : expr }*

The **$** function assigns the expression *expr* to the expression evaluator variable *var*. After the execution of this function, the value of variable named *var* changes if the values of variables or definitions of functions which are included in the expression *expr* change. If a variable named *var* does not yet exist, it is created.

The **$** function can also be used for the definition of new expression evaluator functions. In this case, *func* is the name of the newly defined function, *arg1*, *arg2*, etc. are the names of function arguments which must be listed in square brackets and separtated by commas, and *expr* is the expression which defines how the newly defined function willl be evaluated. The expression *exp* usually contains objects named as formal arguments listed in the sqare brackets. At the function evaluation, these objects are replaced by the actual arguments with wich function is called.

The expression *expr* can also contain other variables and functions of the expression evaluator. If the definitions of these variables or functions are changed later, the definition of the variable or function defined by the **$** function changes accordingly.

Example:

$ *{ powsum[x,y,z]:x^[y*z] }*

defines a new function of the expressionn evaluator named *powsum*, which takes three arguments and returns the first argument raised to the power of the sum of the second and the third argument.

4.5.3 **definefunction** *{ funcname [defblock] }*

The file interpreter's function **definefunction** defines a new expression evaluator function. *funcname* is the name of the function and the *defblock* is the definition block of the function. At every evaluation of the function after this definition, this block is interpreted.

Calculator functions defined by the **definefunction** command can be called with arbitrary number of arguments. These can be accessed by the expression evaluator function **argument**. This function returns the value of a specific argument which was passed to the expression evaluator function defined by the **definefunction** command. Therefore, the **argument** function can be evaluated only within the definition block of an expression evaluator function defined by the file interpreter's command **definefunction**. The only argument of the **argument** function is the sequential number of argument the value of which should be returned. Since expression evaluator functions can only return (i.e. evaluate to) real numbers, the expression evaluator functions defined by the **definefunction** command can not take string arguments. Such arguments could not be accessed in the *defblock* block because the only way of accessing arguments in this block is through the calculator function **argument**. Arguments must therefore be expressions which evaluate to real numbers.

The number of arguments which have been passed to the function defined by the **definefunction** command can be accessed by the expression evaluator function **numargs**. This function takes no arguments and can also be evaluated only within the definition block of the **definefunction** command.

The value which is returned by a function defined by the **definefunction** command must be specified by the file interpreter's function **return**. The only argument of this function must be a mathematical expression which can be evaluated in the expression evaluator. The value of this expression is what the function defined by the **definefunction** command returns.

Example:

The following portion of code defines an expression evaluator function *Sumation* which takes an arbitrary number of arguments and returns their sum:

```
definefunction {  Sumation
[
        ={retsum:0}
        ={indsum:0}
        while { (indsum<=numargs[ ])
        [
                ={retsum:retsum+argument[indsum]}
                ={indsum:indsum+1}
        ] }
        return{retsum}
] }
```

After the function is defined, it can be used in mathematical expressions. For example, the expression "Sumation[3,2*4,5]" will evaluate to 16 (=3+2*4+5).

Warning:

At the definition of new expression evaluator functions we must be careful at choosing names for auxiliary variables used as counters or for carrying intermediate results. A concept of local variables is not implemented in the file interpreter, therefore all variables are global. The variables used locally in the definition block of the **definefunction** command can therefore interfer with global variables if we accidentally chose the same name for them.

4.5.4 **return** *{ expr }*

The file interpreter's function **return** is used for setting the value which is returned by user defined functions of the expression evaluator which are defined by the **definefunction** command. *expr* is a mathematical expression which defines the value which will be returned by such function. The **return** function can be used only in the definition block of the **definefunction** command.

4.5.5 **argument** *[num]*

The expression evaluator-s function **argument** returns the value of *num*-th argument passed to the expression evaluator function which is currently being evaluatd. Therefore, this function can be evaluated only within the definition block of the **definefunction** command.

4.5.6 **numargs** *[ ]*

The expression evaluator-s function **numargs** returns the number of arguments that were passed to the expression evaluatro-s function which is currently being evaluated. This function can therefore be evaluated only within the definition block of the **definefunction** function.