`matchlib` — library for pattern matching

Table of contents

`matchlib::analyze` – structure of an expression

`matchlib::analyze` analyses the structure of any expression.

Call(s):

- `matchlib::analyze(ex <option, ...>)`

Parameters:

`ex` — any MuPAD expression

Options:

`Const` `= set`	— a set or list of expressions that will be taken as constants
`Cond` `= list`	— a list of functions that will be taken to analyze the given expression
`Ident` `= ident`	— an ident or the option `Any`

Return Value: an object of the type `adt::Tree`

Related Functions: `match`, `adt::Tree`, `prog::exprtree`

Details:

- With `matchlib::analyze` expressions will be patitioned and returned in tree structure as `adt::Tree`.
- The returned tree has the following structure. The root is an expression of the same type as the given expression, but the arguments are new generated pattern variables that stands for the arguments. The nodes of this tree are the operands. If one operand should be also partitioned the operand is a subtree of the same structure.
- Every operand of this tree is either a single expression the will be taken as constant, or (generally) a sequence of tree operands `X, S, L`. The first operand `X` is the above describes expression with the pattern variables as operands, the second operand `S` is a set of equations of the form `ident = expression`, whereby `ident` is a generated pattern variable and `expression` the corresponding subexpression of the original expression. The third operand `L` is a list of all operands of the partitioned expression, that was replaced by a pattern variable.
- The given tree can be shown with the function `expose`. The operands can be taken with the methods of the tree, which is an object of the type `adt::Tree`.

Option <`Const = C`>:

- `C` is a set or list of expressions that will be taken as constants and not further examined.

Option <`Cond = C`>:

- `C` is a list of functions that will be taken to analyze the given expression. Every function will be called with the given expression (or a subexpression) and must be return `TRUE` or `FALSE` (or an expression that will be evaluated to this with `bool`). The order of the functions in the list `C` determines the precedence to analyze the expression.
- By default the given expression will be partitioned, if the operation is an elementary function.

Option <`Ident = x`>:

- `x` is an identifier or the option `Any` (default). If an identifier will be given only expressions with this identifier will be analyzed, all others will be taken as constants.

Example 1.

```
>> X:= a^2 + x^2:
   expose(matchlib::analyze(sin(X)/cos(X)))

 1/X1*X2, {X1 = cos(a^2 + x^2), X2 = sin(a^2 + x^2)}, [cos(a^2 \
 + x^2), sin(a^2 + x^2)]
 |
 +-- cos(X3), {X3 = a^2 + x^2}, [a^2 + x^2]
 |   |
 |   '-- X4^2 + X5^2, {X4 = a, X5 = x}, [a, x]
 |       |
 |       +-- a
 |       |
 |       '-- x
 |
 '-- sin(X6), {X6 = a^2 + x^2}, [a^2 + x^2]
     |
     '-- X7^2 + X8^2, {X7 = a, X8 = x}, [a, x]
         |
```

```
        +-- a
        |
        '-- x
```

Subexpression of the form `a^2 + x^2` are constant.

```
>> expose(matchlib::analyze(sin(X)/cos(X), Const = [X]))

 1/X9*X10, {X9 = cos(a^2 + x^2), X10 = sin(a^2 + x^2)}, [cos(a^\
 2 + x^2), sin(a^2 + x^2)]
 |
 +-- cos(X12), {X12 = a^2 + x^2}, [a^2 + x^2]
 |
 '-- sin(X13), {X13 = a^2 + x^2}, [a^2 + x^2]
```

Only expressions of the type `"_mult"` and `"_plus"` will be examined.

```
>> F:= X -> type(X) = "_mult" or type(X) = "_plus":
   T:= matchlib::analyze(sin(X)/cos(X), Cond = [F])

                              Tree3

>> expose(%)

 X14*X15, {X15 = sin(a^2 + x^2), X14 = 1/cos(a^2 + x^2)}, [1/co\
 s(a^2 + x^2), sin(a^2 + x^2)]
 |
 +-- 1/cos(a^2 + x^2)
 |
 '-- sin(a^2 + x^2)
```

Changes:

- `matchlib::analyze` is a new function.