Third-party maxima software

John Lapeyre

September 5, 2013

Contents

1 Array Representation For Expressions **1**

2 Attributes **1**
2.1 Function: attributes . . . 1
2.2 Function: attributes_find . . . 1
2.3 Function: set_match_form . . . 2
2.4 Function: set_nowarn . . . 2
2.5 Function: unset_match_form . . . 2
2.6 Function: unset_nowarn . . . 2

3 Functions and Variables for Array Represention for Expressions **3**
3.1 Function: aeop . . . 3
3.2 Function: aex . . . 4
3.3 Function: aex_cp . . . 4
3.4 Function: aex_get . . . 4
3.5 Function: aex_new . . . 4
3.6 Function: aex_set . . . 4
3.7 Function: aex_shift . . . 5
3.8 Function: aex_unshift . . . 5
3.9 Function: aexg . . . 5
3.10 Function: aexs . . . 6
3.11 Function: copy_aex_type . . . 6
3.12 Function: deep_copy . . . 6
3.13 Function: faex . . . 6
3.14 Function: flex . . . 6
3.15 Function: iapply . . . 6
3.16 Function: iargs . . . 7
3.17 Function: ilength . . . 7
3.18 Function: iop . . . 7
3.19 Function: ipart . . . 8
3.20 Function: ipart_set . . . 8
3.21 Function: ireverse . . . 8
3.22 Function: lex . . . 9
3.23 Function: raex . . . 9
3.24 Function: rlex . . . 9

4 Functions and Variables for Combinatorics **9**
4.1 Function: ae_random_permutation . . . 10
4.2 Function: cycles_to_perm . . . 10
4.3 Function: inverse_permutation . . . 10
4.4 Function: perm_to_cycles . . . 10
4.5 Function: perm_to_transpositions . . . 11
4.6 Function: permutation_p . . . 11
4.7 Function: permutation_p1 . . . 11
4.8 Function: random_cycle . . . 11
4.9 Function: random_permutation_sym . . . 12
4.10 Function: signature_permutation . . . 12
4.11 Function: transpositions_to_perm . . . 12

5 Functions and Variables for Documentation **12**
5.1 Variable: doc_system_list . . . 13
5.2 Variable: error_code . . . 13
5.3 Function: maxdoc . . . 14
5.4 Function: maxdoc_author . . . 14
5.5 Function: maxdoc_copyright . . . 14
5.6 Function: maxdoc_examples . . . 14
5.7 Function: maxdoc_set_cur_sec . . . 14
5.8 Function: maxdoc_set_mext_package . . . 15
5.9 Function: maxdoc_split_text . . . 15
5.10 Function: mext_package_record . . . 15
5.11 Variable: pager_command . . . 15
5.12 Variable: print_authors . . . 15
5.13 Variable: print_copyrights . . . 15
5.14 Function: print_entry_latex . . . 16
5.15 Variable: print_implementation . . . 16
5.16 Function: print_maxdoc_entry . . . 16
5.17 Function: print_maxdoc_sections . . . 16
5.18 Function: print_sections_latex . . . 16
5.19 Variable: read_docs_with_pager . . . 16
5.20 Function: set_all_doc_systems . . . 17
5.21 Function: simple_doc_add . . . 17
5.22 Function: simple_doc_delete . . . 17
5.23 Function: simple_doc_get . . . 17
5.24 Function: simple_doc_init . . . 17
5.25 Function: simple_doc_print . . . 18

6 Functions and Variables for Equations **18**
6.1 Function: alt_eigen . . . 18
6.2 Function: nelder_mead . . . 19

7 Functions and Variables for Function Definition **19**
7.1 Function: comp_load . . . 19
7.2 Function: compile_file1 . . . 20

8 Functions and Variables for Input and Output **20**
8.1 Function: dtest12 . . . 20
8.2 Function: dtest9 . . . 20
8.3 Function: pager_string . . . 20
8.4 Function: restore . . . 21
8.5 Function: restore_fast . . . 21
8.6 Function: store . . . 21
8.7 Function: store_fast . . . 22

9 Functions and Variables for Lists **22**
9.1 Function: aelistp . . . 23
9.2 Function: constant_list . . . 23
9.3 Function: count . . . 23
9.4 Function: drop_while . . . 24
9.5 Function: every1 . . . 24
9.6 Function: fold . . . 24
9.7 Function: fold_list . . . 25
9.8 Function: icons . . . 25
9.9 Function: imap . . . 25
9.10 Function: length_while . . . 26
9.11 Function: lrange . . . 26
9.12 Function: nest . . . 27
9.13 Function: nest_list . . . 27
9.14 Function: nest_while . . . 27
9.15 Function: nreverse . . . 28
9.16 Function: partition_list . . . 28
9.17 Function: select . . . 29
9.18 Argument type: sequence specifier . . . 29
9.19 Function: table . . . 29
9.20 Function: take . . . 30
9.21 Function: take_while . . . 31
9.22 Function: tuples . . . 31

10 Functions and Variables for Number Theory **32**
10.1 Function: abundant_p . . . 33
10.2 Function: aliquot_sequence . . . 33
10.3 Function: aliquot_sum . . . 33
10.4 Function: amicable_p . . . 34
10.5 Function: catalan_number . . . 34
10.6 Function: divisor_function . . . 34
10.7 Function: divisor_summatory . . . 35
10.8 Function: from_digits . . . 35
10.9 Function: integer_digits . . . 35
10.10Function: integer_string . . . 36
10.11Function: oeis_A092143 . . . 36
10.12Function: perfect_p . . . 36
10.13Function: prime_pi . . . 37
10.14Function: prime_pi_soe . . . 37
10.15Function: prime_twins . . . 37
10.16Function: primes1 . . . 38
10.17Function: to_poly_clean . . . 38

11 Functions and Variables for Numerical Computation **38**
11.1 Function: mquad_qag . . . 39
11.2 Function: mquad_qagi . . . 40
11.3 Function: mquad_qagp . . . 40
11.4 Function: mquad_qags . . . 40
11.5 Function: mquad_qawc . . . 40
11.6 Function: mquad_qawf . . . 41
11.7 Function: mquad_qawo . . . 41
11.8 Function: mquad_qaws . . . 41
11.9 Function: n_abs . . . 41
11.10Function: n_acos . . . 42
11.11Function: n_acosh . . . 42
11.12Function: n_asin . . . 42
11.13Function: n_asinh . . . 42
11.14Function: n_atan . . . 42
11.15Function: n_atanh . . . 42
11.16Function: n_cos . . . 42
11.17Function: n_cosh . . . 43
11.18Function: n_exp . . . 43
11.19Function: n_expt . . . 43
11.20Function: n_log . . . 43
11.21Function: n_sin . . . 43
11.22Function: n_sinh . . . 43
11.23Function: n_sqrt . . . 43
11.24Function: n_tan . . . 44
11.25Function: n_tanh . . . 44
11.26Function: nintegrate . . . 44

12 Functions and Variables for Numerics **46**

13 Functions and Variables for Predicates **46**
13.1 Function: aex_p . . . 46
13.2 Function: cmplength . . . 46
13.3 Function: length0p . . . 46
13.4 Function: length1p . . . 47
13.5 Function: length_eq . . . 47
13.6 Function: type_of . . . 47

14 Functions and Variables for Program Flow **48**
14.1 Function: error_str . . . 48

15 Functions and Variables for Quicklisp **48**
15.1 Function: quicklisp_apropos . . . 48
15.2 Function: quicklisp_install . . . 49
15.3 Function: quicklisp_load . . . 49
15.4 Function: quicklisp_start . . . 49

16 Functions and Variables for Runtime Environment **49**
16.1 Function: allow_kill . . . 50
16.2 Function: allow_kill_share . . . 50
16.3 Function: chdir . . . 51
16.4 Function: dir_exists . . . 51

16.5 Function: dirstack . . . 51
16.6 Function: dont_kill . . . 51
16.7 Function: dont_kill_share . . . 52
16.8 Function: get_dont_kill . . . 52
16.9 Variable: homedir . . . 52
16.10Variable: lisp_bin_ext . . . 52
16.11Variable: lisp_type . . . 52
16.12Variable: lisp_version . . . 52
16.13Function: list_directory . . . 53
16.14Variable: maxima_version . . . 53
16.15Function: mcompile_file . . . 53
16.16Function: mext_clear . . . 53
16.17Function: mext_find_package . . . 53
16.18Function: mext_info . . . 54
16.19Function: mext_list . . . 54
16.20Function: mext_list_loaded . . . 54
16.21Function: mext_list_package . . . 54
16.22Function: mext_test . . . 54
16.23Function: mtranslate_file . . . 55
16.24Function: popdir . . . 55
16.25Function: probe_file . . . 56
16.26Function: pwd . . . 56
16.27Function: require . . . 56
16.28Function: timing . . . 56
16.29Function: truename . . . 57
16.30Function: updir . . . 57

17 Functions and Variables for Strings **57**
17.1 Function: string_drop . . . 57
17.2 Function: string_reverse . . . 58
17.3 Function: string_take . . . 58
17.4 Function: with_output_to_string . . . 58

18 Miscellaneous Functions **58**
18.1 Function: examples . . . 59
18.2 Function: examples_add . . . 59

19 Miscellaneous utilities **59**
19.1 Variable: compile_lambda_verbose . . . 59

20 Options **59**
20.1 Option: adj . . . 60
20.2 Option: compile . . . 60
20.3 Function: foptions . . . 60
20.4 Option: ot . . . 60

1 Array Representation For Expressions

Maxima expressions are normally implemented internally as lisp lists, but they may also be represented by lisp arrays. Each representation has advantages.

2 Attributes

A function may possess a list of attributes. The attributes control how the arguments to the function are evaluated and how errors are handled.

- `attributes`
- `attributes_find`
- `set_match_form`
- `set_nowarn`
- `unset_match_form`
- `unset_nowarn`

2.1 Function: attributes

attributes(*name*)
mext package: defmfun1

Description Returns a list of the 'attributes' of function *name*.

Arguments `attributes` requires one argument *name*, which must be a string or a symbol.

See also `unset_match_form`, `set_match_form`, `set_nowarn`, and `unset_nowarn`.

2.2 Function: attributes_find

attributes_find(:optional *attribute*)
mext package: defmfun1

Description Return a list of all functions for which the attribute *attribute* is set. Some attributes are *match_form*,*hold_all*, and *nowarn*.

Arguments `attributes_find` requires either zero or one arguments. If present, the argument *attribute* must be a a string or a symbol.

2.3 Function: set_match_form

set_match_form(*names*)
mext package: defmfun1

Description Set the 'match_form' attribute for function(s) *names*. If the argument checks for a function call fail, and the attribute 'match_form' is set, then rather than signaling an error, the unevaluated form is returned. Furthemore, if the attribute 'nowarn' is not set, then a warning message is printed. Currently, only automatic argument checks generated from the defmfun1 protocol are controlled. Argument checks and errors written within the body of the functions occur regardless of function attributes.

Arguments `set_match_form` requires one argument *names*, which must be a string, a symbol, or a list of strings or symbols.

See also `unset_match_form`, `set_nowarn`, `unset_nowarn`, and `attributes`.

2.4 Function: set_nowarn

set_nowarn(*names*)
mext package: defmfun1

Description Set the 'nowarn' attribute for function(s) *names*. If the argument checks for a function call fail, and the attribute 'match_form' is set, and the attribute 'nowarn' is set, then rather than signaling an error, the unevaluated form is returned and no warning message is printed.

Arguments `set_nowarn` requires one argument *names*, which must be a string, a symbol, or a list of strings or symbols.

See also `unset_match_form`, `set_match_form`, `unset_nowarn`, and `attributes`.

2.5 Function: unset_match_form

unset_match_form(*names*)
mext package: defmfun1

Description Unset the 'match_form' attribute for function(s) *names*. If the argument checks for a function call fail, and the attribute 'match_form' is set, then rather than signaling an error, the unevaluated form is returned. Furthemore, if the attribute 'nowarn' is not set, then a warning message is printed. Currently, only automatic argument checks generated from the defmfun1 protocol are controlled. Argument checks and errors written within the body of the functions occur regardless of function attributes.

Arguments `unset_match_form` requires one argument *names*, which must be a string, a symbol, or a list of strings or symbols.

See also `set_match_form`, `set_nowarn`, `unset_nowarn`, and `attributes`.

2.6 Function: unset_nowarn

unset_nowarn(*names*)
mext package: defmfun1

Description Unset the 'nowarn' attribute for function(s) *names*. If the argument checks for a function call fail, and the attribute 'match_form' is set, and the attribute 'nowarn' is set, then rather than signaling an error, the unevaluated form is returned and no warning message is printed.

Arguments `unset_nowarn` requires one argument *names*, which must be a string, a symbol, or a list of strings or symbols.

See also `unset_match_form`, `set_match_form`, `set_nowarn`, and `attributes`.

3 Functions and Variables for Array Represention for Expressions

These functions operate on the the array expression data structure.

- `aeop`
- `aex`
- `aex_cp`
- `aex_get`
- `aex_new`

- aex_set
- aex_shift
- aex_unshift
- aexg
- aexs
- copy_aex_type
- deep_copy
- faex
- flex
- iapply
- iargs
- ilength
- iop
- ipart
- ipart_set
- ireverse
- lex
- raex
- rlex

3.1 Function: aeop

aeop(*expr*)
mext package: aex

Description op function for aex. returns op if *e* is not an aex.

Arguments `aeop` requires one argument *expr*, which must be non-atomic.

3.2 Function: aex

aex(:optional *x*)
mext package: aex

Calling

> **aex**(*e*) Converts expression *e* to an array representation. The input expression *e* is returned unchanged if it is already an array expression or is a symbol or number or specially represented maxima expression. This function converts only at the first level.

Arguments `aex` requires either zero or one arguments.

Options `aex` takes options with default values: `adj->true`.

3.3 Function: aex_cp

aex_cp(*e* :optional *head*)
mext package: aex

Calling

> **aex_cp**(*e*) Returns an aex form copy of *e*. *e* may be in either lex or aex form. Conversion to aex representation occurs only on the first level.

Arguments `aex_cp` requires either one or two arguments. The first argument *e* must be non-atomic.

Options `aex_cp` takes options with default values: `adj->true`.

3.4 Function: aex_get

mext package: aex

Description Returns the *n-1*th part of aexpr *e*. A value of *n* less than 0 is not allowed. This is more efficient than `aexg`, which returns the head of the expression when *n* is equal to zero.

Examples

```
(%i1) a : aex([5,6,7]), aex_get(a,2);
(%o1) 7
```

3.5 Function: aex_new

aex_new(*n* :optional *head*)
mext package: aex

Arguments `aex_new` requires either one or two arguments. The first argument *n* must be a non-negative integer.

3.6 Function: aex_set

mext package: aex

Description Destructively sets the *n*th part of aexpr *e* to value *v*. A value of 0 for *n* is not allowed. This is more efficient than `aexs`. No argument checking is done.

Examples
Destructively assign to a part of an expression.

```
(%i1) a : aex([1,2,3]), aex_set(a,1,x), a;
(%o1) <[1,x,3]>
```

See also `aexs` and `ipart`.

3.7 Function: aex_shift

aex_shift(*e*)
mext package: aex

Description destructively removes an element from the end of *e*. For array representation of expressions we use the words 'push' and 'pop' for the beginning of and expression, and 'shift' and 'unshift' for the end

of an expression, whether the representation is an array or a list. This is consistent with maxima, but the reverse of the meaning of the terms in perl.

Arguments `aex_shift` requires one argument *e*, which must be an adjustable array expression.

Examples

```
(%i1) a : lrange(10,ot->ar);
(%o1) <[1,2,3,4,5,6,7,8,9,10]>
(%i1) b : aex_shift(a);
(%o1) 10
(%i2) a;
(%o2) <[1,2,3,4,5,6,7,8,9]>
```

3.8 Function: aex_unshift

aex_unshift(*v*, *e*)
mext package: aex

Description Destructively pushes an element *v* onto the end of *e*. The return value is *v*. For array representation of expressions we use the words 'push' and 'pop' for the beginning of and expression, and 'shift' and 'unshift' for the end of an expression, whether the representation is an array or a list. This is consistent with maxima, but the reverse of the meaning of the terms in perl.

Arguments `aex_unshift` requires two arguments. The second argument *e* must be an adjustable array expression.

Examples

```
(%i1) a : lrange(10,ot->ar), aex_unshift("dog",a), a;
(%o1) <[1,2,3,4,5,6,7,8,9,10,"dog"]>
```

3.9 Function: aexg

mext package: aex

Description aexg(e,n) returns the nth part of aexpr e. If n is 0, the head of e is returned. No argument checking is performed.

See also `aex_get`, `ipart`, `inpart`, and `part`.

3.10 Function: aexs

mext package: aex

Description destructively sets the *n*th part of aexpr *e* to value *v*. A value of 0 for *n* returns the head (or op) of *e*.

3.11 Function: copy_aex_type

copy_aex_type(*ein*)
mext package: aex

Description Create a new aex with same head,length,adjustability,etc. but contents of expression are not copied.

Arguments `copy_aex_type` requires one argument *ein*, which must be an array-representation expression.

3.12 Function: deep_copy

deep_copy(*expr*)
mext package: aex

Description Note: it appears that the core maxima function `copy` acheives the same result as `deep_copy`, so that the latter is redundant. It will probably be removed. `deep_copy` returns a copy of expression *expr* which may be of mixed lex/aex representation. An exact copy is made; that is, the representation is preserved at all levels. `deep_copy` is similar to `copylist`, except that it can copy some expressions that `copylist` cannot. For instance, if *expr* is of aex representation at the top level.

Arguments `deep_copy` requires one argument.

3.13 Function: faex

faex(*e*)
mext package: aex

Description deep copy *e* converting to aex representation at all levels.

Arguments `faex` requires one argument.

3.14 Function: flex

flex(*e*)
mext package: aex

Description deep copy *e* converting to lex representation at all levels.

Arguments `flex` requires one argument.

3.15 Function: iapply

iapply(*fun*, *arg*)
mext package: aex

Description `iapply` is like maxima `apply`, but it supports aex lists. *arg* is converted to an ml if it is an aex expression. By default, output is ml regardless of the input representation.

Arguments `iapply` requires two arguments. The first argument *fun* must be a function. The second argument *arg* must be non-atomic.

Options `iapply` takes options with default values: `adj->true`, `ot->ml`.

Examples

```
(%i1) iapply(%%ff,lrange(4));
(%o1) %%ff(1,2,3,4)
```

```
(%i1) iapply(%%ff,lrange(4,[ot->ar]));
(%o1) %%ff(1,2,3,4)
```

```
(%i1) iapply(%%ff,lrange(4,[ot->ar]), [ot->ar] );
(%o1) %%ff<1,2,3,4>
```

```
(%i1) iapply(%%ff,lrange(4), [ot->ar] );
(%o1) %%ff<1,2,3,4>
```

3.16 Function: iargs

iargs(*e*)
mext package: aex

Description returns a list of the arguments of *e*, which may be either a lex or aex expression. This works like `args`, but is more general.

Arguments `iargs` requires one argument *e*, which must be a subscripted variable or non-atomic.

Options `iargs` takes options with default values: `adj->true`, `ot->ml`.

3.17 Function: ilength

ilength(*e*)
mext package: aex

Description Returns the length of the expression *e*. This is like maxima `length`, but here, *e* can be either an aex or a lex.

Arguments `ilength` requires one argument *e*, which must be a subscripted variable or non-atomic.

3.18 Function: iop

iop(*expr*)
mext package: aex

Description returns the main operator of the expression *expr*, which may be either a lex or aex expression. This works like 'op', but is more general.

Arguments `iop` requires one argument *expr*, which must be a subscripted variable or non-atomic.

3.19 Function: ipart

mext package: aex

Calling

> **ipart**(*e*, *ind1*, *ind2*, ...) Returns the part of expression *e* specified by indices. *e* may be a mixed (lex and aex) representation expression. When used as an lvalue, ipart can be used to assign to a part of an expression. If an index is negative, then it counts from the end of the list. If *e* is an ordinary maxima list (lex), then using a negative index is potentially slower than using a positive index because the entire list must first be traversed in order to determine it's length. If *e* is in aex representation, then this inefficiency is not present.

Examples
Destructively assign to a part of an exression.

```
(%i1) (a : [1,2,3], ipart(a,1) : 7, a);
(%o1) [7,2,3]
```

Implementation Some tests were performed with large lists of numbers. If we set `a:lrange(10^7)`, then the times required for `ipart(a,10^7)`, `ipart(a,-1)`, `inpart(a,10^7)`, and `part(a,10^7)` were 30, 60, 90, and 90 ms.

3.20 Function: ipart_set

mext package: aex

Calling

ipart_set(*e*, *val*, *ind1*, *ind2*, ...) Set part of *e* specified by the final arguments to *val*. *e* is a mixed representation expression.

3.21 Function: ireverse

ireverse(*e*)
mext package: aex

Description ireverse is like maxima reverse, but is works on both aex and list objects. ireverse is tries to be identical to maxima reverse for a non-aex argument.

Arguments `ireverse` requires one argument *e*, which must be non-atomic.

Options `ireverse` takes options with default values: `adj->true`, `ot->ml`.

Examples

```
(%i1) ireverse(lrange(4));
(%o1) [4,3,2,1]
```

```
(%i1) ireverse(lrange(4), [ot->ar] );
(%o1) <[4,3,2,1]>
```

```
(%i1) ireverse(lrange(4, [ot->ar]) );
(%o1) <[4,3,2,1]>
```

```
(%i1) ireverse(lrange(4, [ot->ar]), [ot->ml] );
(%o1) [4,3,2,1]
```

3.22 Function: lex

mext package: aex

Calling

lex(*e*) converts the aex expression *e* to lex. If *e* is not an aex expression, *e* is returned. Conversion is only done on the first level.

3.23 Function: raex

raex(*e* :optional *spec*)
mext package: aex

Description Convert *e* to aex representation at the specified levels.

Arguments `raex` requires either one or two arguments.

3.24 Function: rlex

rlex(*e* :optional *spec*)
mext package: aex

Description Deep copy, converting *e* to lex representation at the specified levels.

Arguments `rlex` requires either one or two arguments.

4 Functions and Variables for Combinatorics

- `ae_random_permutation`
- `cycles_to_perm`
- `inverse_permutation`
- `perm_to_cycles`
- `perm_to_transpositions`
- `permutation_p`
- `permutation_p1`
- `random_cycle`
- `random_permutation_sym`
- `signature_permutation`
- `transpositions_to_perm`

4.1 Function: ae_random_permutation

ae_random_permutation(*a*)
mext package: discrete_aex

Description returns *a* with subexpressions permuted randomly.

Arguments `ae_random_permutation` requires one argument *a*, which must be non-atomic.

Options `ae_random_permutation` takes options with default values: `adj->true`, `ot->ml`.

See also `random_cycle`, `random_permutation_sym`, `signature_permutation`, `perm_to_cycles`, and `cycles_to_perm`.

4.2 Function: cycles_to_perm

cycles_to_perm(*cycles*)
mext package: discrete_aex

Description Returns a permutation from its cycle decomposition *cycles*, which is a list of lists. Here 'permutation' means a permutation of a list of the integers from 1 to some number *n*. The default output representation is aex.

Arguments `cycles_to_perm` requires one argument *cycles*, which must be a list (lex or aex).

Options `cycles_to_perm` takes options with default values: `adj->true`, `ot->ml`.

See also `random_cycle`, `random_permutation_sym`, `ae_random_permutation`, `signature_permutation`, and `perm_to_cycles`.

4.3 Function: inverse_permutation

inverse_permutation(*perm*)
mext package: discrete_aex

Description Returns the inverse permutation of *perm*.

Arguments `inverse_permutation` requires one argument *perm*, which must be a list (lex or aex).

Options `inverse_permutation` takes options with default values: `adj->true`, `ot->ml`.

Examples

```
(%i1) inverse_permutation([5,1,4,2,6,8,7,3,10,9]);
(%o1) <[2,4,8,3,1,5,7,6,10,9]>
(%i1) inverse_permutation(inverse_permutation([5,1,4,2,6,8,7,3,10,9]),ot->ml);
(%o1) [5,1,4,2,6,8,7,3,10,9]
```

4.4 Function: perm_to_cycles

perm_to_cycles(*ain*)
mext package: discrete_aex

Description Returns a cycle decomposition of the input permutation *ain*. The input must be a permutation of n integers from 1 through n.

Arguments `perm_to_cycles` requires one argument *ain*, which must be a list (lex or aex).

Options `perm_to_cycles` takes options with default values: `adj->true`, `ot->ml`.

Examples

```
(%i1) perm_to_cycles([5,4,3,2,1,10,6,7,8,9]);
(%o1) [[7,8,9,10,6],[3],[4,2],[5,1]]
```

See also `random_cycle`, `random_permutation_sym`, `ae_random_permutation`, `signature_permutation`, and `cycles_to_perm`.

4.5 Function: perm_to_transpositions

perm_to_transpositions(*ain*)
mext package: discrete_aex

Description Returns a list representing the permutation *ain* as a product of transpositions. The output representation type is applied at both levels.

Arguments `perm_to_transpositions` requires one argument *ain*, which must be a list (lex or aex).

Options `perm_to_transpositions` takes options with default values: `adj->true`, `ot->ml`.

4.6 Function: permutation_p

permutation_p(*ain*)
mext package: discrete_aex

Calling

permutation_p(*list*) Returns true if the list *list* of length n is a permutation of the integers from 1 through n. Otherwise returns false.

Arguments `permutation_p` requires one argument.

Implementation Separate routines for aex and lex input are used.

4.7 Function: permutation_p1

permutation_p1(*ain*)
mext package: discrete_aex

Description This is the same as `permutation_p`, but, if the input is a list, it assumes all elements in the input list are fixnum integers, while `permutation_p` does not.

Arguments `permutation_p1` requires one argument.

Implementation Some variables are declared fixnum, but this does not seem to improve performance with respect to `permutationp`.

4.8 Function: random_cycle

random_cycle(*n*)
mext package: discrete_aex

Calling

> **random_cycle**(*n*) Returns a random cycle of length *n*. The return value is a list of the integers from 1 through *n*, representing an element of the symmetric group S_n that is a cycle.

Arguments `random_cycle` requires one argument *n*, which must be a positive integer.

Options `random_cycle` takes options with default values: `adj->true`, `ot->ml`.

See also `random_permutation_sym`, `ae_random_permutation`, `signature_permutation`, `perm_to_cycles`, and `cycles_to_perm`.

Implementation This function uses Sattolo's algorithm.

4.9 Function: random_permutation_sym

random_permutation_sym(*n*)
mext package: discrete_aex

Calling

> **random_permutation_sym**(*n*) Returns a random permutation of the integers from 1 through *n*. This represents a random element of the symmetric group S_n.

Arguments `random_permutation_sym` requires one argument *n*, which must be a positive integer.

Options `random_permutation_sym` takes options with default values: `adj->true`, `ot->ml`.

See also `random_cycle`, `ae_random_permutation`, `signature_permutation`, `perm_to_cycles`, and `cycles_to_perm`.

4.10 Function: signature_permutation

signature_permutation(*ain*)
mext package: discrete_aex

Calling

signature_permutation(*list*) returns the sign, or signature, of the symmetric permutation *list*, which must be represented by a permuation the integers from 1 through n, where n is the length of the list.

Arguments `signature_permutation` requires one argument *ain*, which must be a list (lex or aex).

See also `random_cycle`, `random_permutation_sym`, `ae_random_permutation`, `perm_to_cycles`, and `cycles_to_perm`.

4.11 Function: transpositions_to_perm

transpositions_to_perm(*ain*)
mext package: discrete_aex

Description Returns the permutation specified by the list of transpositions *ain*.

Arguments `transpositions_to_perm` requires one argument *ain*, which must be a list (lex or aex).

Options `transpositions_to_perm` takes options with default values: `adj->true`, `ot->ml`.

Implementation Input is converted to lex on both levels. Default output is aex.

5 Functions and Variables for Documentation

- `doc_system_list`
- `error_code`
- `maxdoc`
- `maxdoc_author`
- `maxdoc_copyright`
- `maxdoc_examples`
- `maxdoc_set_cur_sec`
- `maxdoc_set_mext_package`
- `maxdoc_split_text`
- `mext_package_record`
- `pager_command`
- `print_authors`
- `print_copyrights`
- `print_entry_latex`
- `print_implementation`
- `print_maxdoc_entry`
- `print_maxdoc_sections`
- `print_sections_latex`
- `read_docs_with_pager`

- `set_all_doc_systems`
- `simple_doc_add`
- `simple_doc_delete`
- `simple_doc_get`
- `simple_doc_init`
- `simple_doc_print`

5.1 Variable: doc_system_list

mext package: maxdoc

default value `false`.

Description A list of the documenatation systems that will be searched by ? and ??. This can be set to all avaliable systems with the function `set_all_doc_systems`. If this variable is false, then all documentation is enabled.

5.2 Variable: error_code

mext package: defmfun1

default value `false`.

Description This is an error code set by `merror1`.

5.3 Function: maxdoc

maxdoc(*name*, *docs*)
mext package: maxdoc

Description Add maxdoc documentation entry for item *name* specified by *docs*.

Arguments `maxdoc` requires two arguments. The first argument *name* must be a string.

Options `maxdoc` takes options with default values: `package->false`, `source_filename->false`.

Attributes maxdoc has attributes: [hold_all]

5.4 Function: maxdoc_author

maxdoc_author(*name*, *author*)
mext package: maxdoc

Description Set the author(s) for the documentation item *name*.

Arguments `maxdoc_author` requires two arguments. The first argument *name* must be a string. The second argument *author* must be a string or a list of strings.

5.5 Function: maxdoc_copyright

maxdoc_copyright(*name*, *copyright*)
mext package: maxdoc

Description Set the copyright information for the documentation item *name*. *copyright* should typically be a list whose first element is an integer (the year), with the remaining strings naming the copyright holder. This copyright information will not be printed with documentation, unless *print_copyrights* is true.

Arguments `maxdoc_copyright` requires two arguments. The first argument *name* must be a string.

5.6 Function: maxdoc_examples

maxdoc_examples(*name* :rest *examples*)
mext package: maxdoc

Description Add maxdoc examples entry for item *name* specified by *examples*.

Arguments `maxdoc_examples` requires one or more arguments. The first argument *name* must be a string.

Attributes maxdoc_examples has attributes: [hold_all]

5.7 Function: maxdoc_set_cur_sec

maxdoc_set_cur_sec(*shortname*)
mext package: maxdoc

Description Set the current section for maxdoc to *shortname*. This section will be used by functions such as `maxdoc`, and `maxdoc_author`.

Arguments `maxdoc_set_cur_sec` requires one argument *shortname*, which must be a string.

5.8 Function: maxdoc_set_mext_package

maxdoc_set_mext_package(*packagename*)
mext package: maxdoc

Description Set the current mext package name for maxdoc to *packagename*. This name will be used by functions specifying documentation for functions until the name is set to another value. When documenting functions written in maxima code, calling `mext_record_package` is probably more useful.

Arguments `maxdoc_set_mext_package` requires one argument *packagename*, which must be a string.

5.9 Function: maxdoc_split_text

maxdoc_split_text(*text*)
mext package: maxdoc

Description Split the string *text* into a list of strings, using a sequence of one or more spaces as the delimeter. Single newlines are removed.

Arguments `maxdoc_split_text` requires one argument *text*, which must be a string.

5.10 Function: mext_package_record

mext_package_record(*docitems*, *packagename* :optional *source-filename*)
mext package: maxdoc

Description Set the mext packagename for the function or variable (or list of them) *docitems* to *packagename*.This name will be used when displaying documentation.The function `maxdoc_set_mext_package` is useful for setting the package name of a group of functions, but there is currently no maxima hook for doing this.

Arguments `mext_package_record` requires either two or three arguments. The first argument *docitems* must be a string, a symbol, or a list of strings or symbols. The second argument *packagename* must be a string. The third argument *source-filename* must be a string.

5.11 Variable: pager_command

mext package: defmfun1

default value `/usr/bin/less`.
Description The pathname to the system command used for paged output, for instance, for reading documentation.

5.12 Variable: print_authors

mext package: defmfun1

default value `true`.
Description If true, then print the names of the authors with maxdoc documentation.

5.13 Variable: print_copyrights

mext package: defmfun1

default value `false`.
Description If true, then print copyright information with maxdoc documentation.

5.14 Function: print_entry_latex

print_entry_latex(*item*)
mext package: maxdoc

Arguments `print_entry_latex` requires one argument *item*, which must be a string.

5.15 Variable: print_implementation

mext package: defmfun1

default value `true`.
Description If true, then print implementation information with maxdoc documentation.

5.16 Function: print_maxdoc_entry

print_maxdoc_entry(*item*)
mext package: maxdoc

Arguments `print_maxdoc_entry` requires one argument *item*, which must be a string.

5.17 Function: print_maxdoc_sections

print_maxdoc_sections()
mext package: maxdoc

Description Print all sections of maxdoc documentation. This does not include other documentation databases, such as the main maxima documentation.

Arguments `print_maxdoc_sections` requires zero arguments.

5.18 Function: print_sections_latex

print_sections_latex(:optional *filename*)
mext package: maxdoc

Description Print all sections of maxdoc documentation currently loaded in latex format to the file *filename*. This does not include other documentation databases, such as the main maxima documentation.

Arguments `print_sections_latex` requires either zero or one arguments. If present, the argument *filename* must be a a string.

5.19 Variable: read_docs_with_pager

mext package: maxdoc

default value `false`.
Description If true, then documenation printed by `describe` or ? or ?? is sent through the pager specified by `pager_command`. This will most likely only work with a command line interface under linux/unix with certain lisp implementations.

5.20 Function: set_all_doc_systems

set_all_doc_systems()
mext package: maxdoc

Description Enable all documentation databases for describe, ? and ??. This sets doc_system_list to a list of all doc systems.

Arguments `set_all_doc_systems` requires zero arguments.

5.21 Function: simple_doc_add

simple_doc_add(*name*, *content*)
mext package: maxdoc

Description Adds documentation string *content* for item *name*. These documentation strings are accessible via '?' and '??'.

Arguments `simple_doc_add` requires two arguments. The first argument *name* must be a string. The second argument *content* must be a string.

See also `simple_doc_init`, `simple_doc_delete`, `simple_doc_get`, and `simple_doc_print`.

5.22 Function: simple_doc_delete

simple_doc_delete(*name*)
mext package: maxdoc

Description Deletes the simple_doc documentation string for item *name*.

Arguments `simple_doc_delete` requires one argument *name*, which must be a string.

See also `simple_doc_init`, `simple_doc_add`, `simple_doc_get`, and `simple_doc_print`.

5.23 Function: simple_doc_get

simple_doc_get(*name*)
mext package: maxdoc

Description Returns the simple_doc documentation string for item *name*.

Arguments `simple_doc_get` requires one argument *name*, which must be a string.

See also `simple_doc_init`, `simple_doc_add`, `simple_doc_delete`, and `simple_doc_print`.

5.24 Function: simple_doc_init

simple_doc_init()
mext package: maxdoc

Description Initialize the simple_doc documentation database.

Arguments `simple_doc_init` requires zero arguments.

See also `simple_doc_add`, `simple_doc_delete`, `simple_doc_get`, and `simple_doc_print`.

5.25 Function: simple_doc_print

simple_doc_print(*name*)
mext package: maxdoc

Description Prints the simple_doc documentation string for item *name*.

Arguments `simple_doc_print` requires one argument *name*, which must be a string.

See also `simple_doc_init`, `simple_doc_add`, `simple_doc_delete`, and `simple_doc_get`.

6 Functions and Variables for Equations

- `alt_eigen`
- `nelder_mead`

6.1 Function: alt_eigen

mext package: alt_eigen

Description `alt_eigen(mat, ['var=x, 'maxdegree=n, 'orthogonal=boolean])`

Express the eigenvectors of the matrix *mat* as a polynomial in the eigenvalue *var*. When the degree of a factor of the characteristic polynomial has degree *maxdegree* or less, the code attempts to find all the roots of the factor. The optional variables *var*, *maxdegree* (default 1), and *orthogonal* (default false) can be in any order. When the dimension of an eigenspace is greater than one and *orthogonal* is true, the list of eigenvectors is orthogonal.

Examples The eigenvectors of `matrix([1,2],[4,5])`, are

```
(%i1) alt_eigen(matrix([1,2],[4,5]),'var=z);
(%o1) [z^2=6*z+3,[matrix([2],[z-1])]]
```

Substituting the two roots of `z^2=6*z+3` in to the column vector `matrix([2],[z-1])` gives the two eigenvectors. To find explicit expressions for these eigenvectors, set *maxdegree* to 2; thus

```
(%i2) alt_eigen(matrix([1,2],[4,5]),'var=z, 'maxdegree=2);
(%o2) [z=2*sqrt(3)+3,[matrix([2],[2*sqrt(3)+2])],z=3-2*sqrt(3),
    [matrix([2],[2-2*sqrt(3)])]]
```

Here is a matrix with a degenerate eigenvalue:

```
(%i3) m : matrix([5,6,5,6,5,6],[6,5,6,5,6,5],[5,6,5,6,5,6],[6,5,6,5,6,5],
   [5,6,5,6,5,6],[6,5,6,5,6,5])$
(%i4) alt_eigen(m,'var=z);
(%o4) [z=-3,[matrix([-1],[1],[-1],[1],[-1],[1])],z=0,[matrix([0],[-1],[0],[1],[0],[0]),
     matrix([0],[0],[0],[-1],[0],[1]), matrix([0],[0],[1],[0],[-1],[0]),
     matrix([1],[0],[-1],[0],[0],[0])],z=33,[matrix([-1],[-1],[-1],[-1],[-1],[-1])]]
```

There are four eigenvectors with eigenvalue 0. To find an orthogonal basis for this eigenspace, set the optional variable *orthogonal* to true; thus

```
(%i5) alt_eigen(m,'var=z, 'orthogonal=true);
(%o5) [z=-3,[matrix([-1],[1],[-1],[1],[-1],[1])],z=0,[matrix([1],[0],[-1/2],[0],
    [-1/2],[0]), matrix([0],[0],[1],[0],[-1],[0]),matrix([0],[-1/2],[0],[-1/2],[0],[1]),
    matrix([0],[-1],[0],[1],[0],[0])],z=33,[matrix([-1],[-1],[-1],[-1],[-1],[-1])]]
```

Author Barton Willis.

6.2 Function: nelder_mead

nelder_mead(*expr*, *vars*, *init*)
mext package: nelder_mead

Description The Nelder-Mead optimization algorithm.

Arguments `nelder_mead` requires three arguments. The second argument *vars* must be a list of symbols. The third argument *init* must be a list of numbers.

Examples

Find the minimum of a function at a non-analytic point.

```
(%i1) nelder_mead(if x<0 then -x else x^2, [x], [4]);
(%o1) [x = 9.5364e-11]
```

```
(%i1) f(x) := if x<0 then -x else x^2$
(%i2) nelder_mead(f, [x], [4]);
(%o2) [x = 9.536387892694628e-11]
(%i3) nelder_mead(f(x), [x], [4]);
(%o3) [x = 9.536387892694628e-11]
```

```
(%i1) nelder_mead(x^4+y^4-2*x*y-4*x-3*y, [x,y], [2,2]);
(%o1) [x = 1.1572,y = 1.0993]
```

Author Mario S. Mommer.

7 Functions and Variables for Function Definition

- `comp_load`
- `compile_file1`

7.1 Function: comp_load

comp_load(*fname* :optional *pathlist*)
mext package: defmfun1

Description Compile and load a lisp file. Maxima does not load it by default with `compile_file`. If the input filename does not end with ".lisp", it will be appended. If *pathlist* is specified, then *fname* is only searched for in directories in *pathlist*.

Arguments `comp_load` requires either one or two arguments. The first argument *fname* must be a string. The second argument *pathlist* must be a string or a list of strings.

7.2 Function: compile_file1

compile_file1(*input-file* :optional *bin-file*, *translation-output-file*)
mext package: defmfun1

Description This is copied from maxima `compile_file`, with changes. Sometimes a loadable binary file is apparently compiled, but an error flag is set and `compile_file` returns false for the output binary filename. Here we return the binary filename in any case.

Arguments `compile_file1` requires between one and three arguments. The first argument *input-file* must be a string.

8 Functions and Variables for Input and Output

- `dtest12`
- `dtest9`
- `pager_string`

- `restore`
- `restore_fast`
- `store`
- `store_fast`

8.1 Function: dtest12

dtest12(:rest *args*)
mext package: test_defmfun1

Description There is no doc.

Arguments `dtest12` requires zero or more arguments. Each of the remaining arguments must be a string.

8.2 Function: dtest9

dtest9(*x*)
mext package: test_defmfun1

Description This is the description of dtest9

Arguments `dtest9` requires one argument *x*, which must be a number.

Options `dtest9` takes options with default values: `match->false`.

8.3 Function: pager_string

pager_string(*s*)
mext package: defmfun1

Description Read the string *s* in the pager given by the maxima variable `pager_command`.This works at least with gcl under linux.

Arguments `pager_string` requires one argument *s*, which must be a string.

8.4 Function: restore

restore(*file*)
mext package: store

Calling

restore(*file*) Reads and returns expressions from the file *file*.

Description Reads maxima expressions from file *file* created by the function `store`.

Arguments `restore` requires one argument *file*, which must be a string.

See also `store`, `store_fast`, and `restore_fast`.

8.5 Function: restore_fast

restore_fast(*file*)
mext package: store

Calling

> **restore_fast**(*file*) Reads and returns expressios from the file *file*. No checking for circular references is done.

Description Reads maxima expressions from file *file* created by the function `store`, or `store_fast`. No checks for circular references are done.

Arguments `restore_fast` requires one argument *file*, which must be a string.

See also `store`, `restore`, and `store_fast`.

8.6 Function: store

store(*file* :rest *exprs*)
mext package: store

Calling

> **store**(*file*, *expr1*, *expr2*, ...) stores the expressions to the file *file*.

Description Stores maxima expressions *exprs* in *file* in binary format. Many types of lisp expressions and subexpressions are supported: numbers,strings,list,arrays,hashtables,structures,....

Arguments `store` requires one or more arguments. The first argument *file* must be a string.

Examples
Save a graph to a file. This cannot be done with the command ¡save¿.

```
(%i1) load(graphs)$
(%i2) c : petersen_graph();
(%o2) GRAPH(10 vertices, 15 edges)
(%i3) factor(graph_charpoly(c,x));
(%o3) (x-3)*(x-1)^5*(x+2)^4
(%i4) store("graph.cls",c)$
(%i5) factor(graph_charpoly( restore("graph.cls"), x));
(%o5) (x-3)*(x-1)^5*(x+2)^4
```

See also `restore`, `store_fast`, and `restore_fast`.

Implementation store uses the cl-store library. See the cl-store documentation for more information.

8.7 Function: store_fast

store_fast(*file* :rest *exprs*)
mext package: store

Calling

> **store_fast**(*file*, *expr1*, *expr2*, ...) stores the expressions to the file *file*. No checking for circular references is done.

Description Stores maxima expressions *exprs* in *file* in binary format. This is like `store`, except that no checks for circular references are done.

Arguments `store_fast` requires one or more arguments. The first argument *file* must be a string.

See also `store`, `restore`, and `restore_fast`.

9 Functions and Variables for Lists

These functions manipulate lists. They build lists, take them apart, select elements, etc.

- `aelistp`
- `constant_list`
- `count`
- `drop_while`
- `every1`
- `fold`
- `fold_list`
- `icons`
- `imap`
- `length_while`
- `lrange`
- `nest`
- `nest_list`
- `nest_while`
- `nreverse`
- `partition_list`
- `select`
- `sequence specifier`
- `table`
- `take`
- `take_while`
- `tuples`

9.1 Function: aelistp

mext package: lists_aex

Description Returns true if e is a list, either ml or ar representation.

Examples

```
(%i1) aelistp([1,2,3]);
(%o1) true
(%i1) aelistp( aex([1,2,3]));
(%o1) true
(%i2) aelistp(3);
(%o2) false
(%i3) aelistp(x);
(%o3) false
(%i4) x:lrange(10),aelistp(x);
(%o4) true
(%i5) aelistp(%%f(y));
(%o5) false
(%i6) aelistp( aex( %%f(y) ));
(%o6) false
```

9.2 Function: constant_list

constant_list(*expr*, *list*)
mext package: lists_aex

Description Returns a list of n elements, each of which is an independent copy of expr. `constant_list(expr,[n,m,..])` returns a nested list of dimensions n,m,... where each leaf is an independent copy of expr and the copies of each list at each level are independent. If a third argument is given, then it is used as the op, rather than 'list', at every level.

Arguments `constant_list` requires either two or three arguments. The second argument *spec* must be a positive integer or a list of positive integers.

Options `constant_list` takes options with default values: `adj->true`, `ot->ml`.

See also `makelist`, `lrange`, and `table`.

9.3 Function: count

count(*expr*, *item*)
mext package: lists_aex

Description Counts the number of items in *expr* matching *item*. If *item* is a lambda function then *compile* must be true.

Arguments `count` requires two arguments. The first argument *expr* must be non-atomic and either aex or represented by a lisp list.

Options `count` takes options with default values: `compile->true`.

Examples

```
(%i1) count([1,2,"dog"], 'numberp);
(%o1) 2
```

```
(%i1) count([1,2,"dog"], "dog");
(%o1) 1
(%i2) count(lrange(10^4), lambda([x], is(mod(x,3) = 0)));
(%o2) 3333
(%i3) count( %%ff(1,2,"dog"), "dog");
(%o3) 1
(%i4) count(lrange(100,ot->ar), 'evenp);
(%o4) 50
(%i5) count(lrange(10^5), 'primep);
(%o5) 9592
```

9.4 Function: drop_while

drop_while(*expr*, *test*)
mext package: lists_aex

Calling

> **drop_while**(*expr*, *test*) Tests the elements of *expr* in order, dropping them until *test* fails. The remaining elements are returned in an expression with the same op as that *expr*.

Arguments `drop_while` requires two arguments. The first argument *expr* must be non-atomic and represented by a lisp list.

Options `drop_while` takes options with default values: `adj->true`, `ot->ml`, `compile->true`.

Examples
Drop elements as long as they are negative.

```
(%i1) drop_while([-3,-10,-1,3,6,7,-4], lambda([x], is(x<0)));
(%o1) [3,6,7,-4]
```

9.5 Function: every1

every1(*expr*, *test*)
mext package: lists_aex

Calling

> **every1**(*expr*, *test*) Returns true if *test* is true for each element in *expr*. Otherwise, false is returned. This is like `every` but allow a test that takes only one argument. For some inputs, every1 is much faster than every.

Arguments `every1` requires two arguments. The first argument *expr* must be non-atomic and represented by a lisp list.

Options `every1` takes options with default values: `compile->true`.

9.6 Function: fold

mext package: lists_aex

Description `fold(f,x,[a,b,c])` returns `f(f(f(x,a),b),c).`

Arguments `fold` requires three arguments. The third argument *v* must be non-atomic.

Options `fold` takes options with default values: `adj->true`, `ot->ml`, `compile->true`.

See also `fold_list` and `nest`.

9.7 Function: fold_list

mext package: lists_aex

Description fold_list(f,x,[a,b,c]) returns [f(x,a),f(f(x,a),b),f(f(f(x,a),b),c)].

Arguments `fold_list` requires three arguments. The third argument v must be non-atomic.

Options `fold_list` takes options with default values: `adj->true`, `ot->ml`, `compile->true`.

See also `fold` and `nest`.

9.8 Function: icons

icons(x, e)
mext package: aex

Description `icons` is like maxima `cons`, but less general, and much, much faster. x is a maxima object. e is a maxima list or list-like object, such as `[a]`, or `f(a)`. It is suitable at a minimum, for pushing a number or list or string onto a list of numbers, or strings or lists. If you find `icons` gives buggy behavior that you are not interested in investigating, use `cons` instead.

Implementation In a function that mostly only does icons in a loop, icons defined with defmfun rather than defmfun1 runs almost twice as fast. So icons is defined with defmfun rather than defmfun1. icons does no argument checking.

9.9 Function: imap

imap(f, *expr*)
mext package: lists_aex

Description Maps functions of a single argument. I guess that `map` handles more types of input without error. But `imap` can be much faster for some inputs. This is especially true if a lambda function is passed to imap, as it can be compiled.

Arguments `imap` requires two arguments. The second argument *expr* must be non-atomic.

Options `imap` takes options with default values: `compile->true`.

Examples

Map sqrt efficiently over a list of floats

```
(%i1) (a : lrange(1.0,4),
       imap(lambda([x],modedeclare(x,float),sqrt(x)),a));
(%o1) [1.0,1.4142,1.7321,2.0]
```

With aex expression, no conversions to lex are done.

```
(%i1) (a : lrange(1.0,4,ot->ar),
         imap(lambda([x],modedeclare(x,float),sqrt(x)),a));
(%o1) <[1.0,1.4142,1.7321,2.0]>
```

9.10 Function: length_while

length_while(*expr*, *test*)
mext package: lists_aex

Description Computes the length of *expr* while *test* is true.

Arguments `length_while` requires two arguments. The first argument *expr* must be non-atomic and represented by a lisp list.

Options `length_while` takes options with default values: `compile->true`.

Examples

```
(%i1) length_while([-3,-10,-1,3,6,7,-4], lambda([x], is(x<0)));
(%o1) 3
```

9.11 Function: lrange

mext package: lists_aex

Calling

lrange(*stop*) returns a list of numbers from 1 through *stop*.

lrange(*start*, *stop*) returns a list of expressions from *start* through *stop*.

lrange(*start*, *stop*, *incr*) returns a list of expressions from *start* through *stop* in steps of *incr*.

Description lrange is much more efficient than makelist for creating ranges, particularly for large lists (e.g. 10^5 or more items.) Functions for creating a list of numbers, in order of decreasing speed, are: `lrange`, `table`, `create_list`,`makelist`.

Arguments `lrange` requires between one and three arguments. The third argument *incr* must be an expression that is not zero.

Options `lrange` takes options with default values: `adj->true`, `ot->ml`.

Examples

```
(%i1) lrange(6);
(%o1) [1,2,3,4,5,6]
(%i1) lrange(2,6);
(%o1) [2,3,4,5,6]
(%i2) lrange(2,6,2);
(%o2) [2,4,6]
(%i3) lrange(6,1,-1);
(%o3) [6,5,4,3,2,1]
(%i4) lrange(6,1,-2);
(%o4) [6,4,2]
(%i5) lrange(6,ot->ar);
(%o5) <[1,2,3,4,5,6]>
```

The type of the first element and increment determine the type of the elements.

```
(%i1) lrange(1.0,6);
(%o1) [1.0,2.0,3.0,4.0,5.0,6.0]
```

```
(%i1) lrange(1.0b0,6);
(%o1) [1.0b0,2.0b0,3.0b0,4.0b0,5.0b0,6.0b0]
(%i2) lrange(1/2,6);
(%o2) [1/2,3/2,5/2,7/2,9/2,11/2]
(%i3) lrange(6.0,1,-1);
(%o3) [6.0,5.0,4.0,3.0,2.0,1.0]
```

Symbols can be used for limits or increments.

```
(%i1) lrange(x,x+4);
(%o1) [x,x+1,x+2,x+3,x+4]
(%i1) lrange(x,x+4*a,a);
(%o1) [x,x+a,x+2*a,x+3*a,x+4*a]
```

See also `makelist`, `table`, and `constant_list`.

9.12 Function: nest

mext package: lists_aex

Description nest(f,x,n) returns f(...f(f(f(x)))...) where there are n nested calls of f.

Arguments `nest` requires three arguments. The first argument *f* must be a function. The third argument *n* must be a non-negative integer.

Options `nest` takes options with default values: `adj->true`, `ot->ml`, `compile->true`.

9.13 Function: nest_list

nest_list(*f*, *x*, *n*)
mext package: lists_aex

Arguments `nest_list` requires three arguments. The third argument *n* must be a non-negative integer.

Options `nest_list` takes options with default values: `adj->true`, `ot->ml`, `compile->true`.

Examples

Find the first 10 primes after 100.

```
(%i1) nest_list(next_prime,100,10);
(%o1) [101,103,107,109,113,127,131,137,139,149]
```

See also `nest`, `fold`, and `fold_list`.

9.14 Function: nest_while

nest_while(*f*, *x*, *test* :optional *min*, *max*)
mext package: lists_aex

Calling

nest_while(*f*, *x*, *test*) applies *f* to *x* until *test* fails to return true when called on the nested result.

nest_while(*f*, *x*, *test*, *min*) applies *f* at least *min* times.

nest_while(*f*, *x*, *test*, *min*, *max*) applies *f* not more than *max* times.

Arguments `nest_while` requires between three and five arguments. The fourth argument *min* must be a non-negative integer. The fifth argument *max* must be a non-negative integer.

Options `nest_while` takes options with default values: `adj->true`, `ot->ml`, `compile->true`.

Implementation This should be modified to allow applying test to more than just the most recent result.

9.15 Function: nreverse

nreverse(*e*)
mext package: lists_aex

Description Destructively reverse the arguments of expression *e*. This is more efficient than using reverse.

Arguments `nreverse` requires one argument *e*, which must be non-atomic.

Examples
Be careful not to use *a* after applying nreverse. Do assign the result to another variable.

```
(%i1) a : lrange(10), b : nreverse(a);
(%o1) [10,9,8,7,6,5,4,3,2,1]
(%i1) a : lrange(10,ot->ar), b : nreverse(a);
(%o1) <[10,9,8,7,6,5,4,3,2,1]>
```

See also `reverse`.

9.16 Function: partition_list

partition_list(*e*, *nlist* :optional *dlist*)
mext package: lists_aex

Calling

partition_list(*e*, *n*) partitions *e* into sublists of length *n*

partition_list(*e*, *n*, *d*) partitions *e* into sublists of length *n* with offsets *d*.

Description Omitting *d* is equivalent to giving *d* equal to *n*. *e* can be any expression, not only a list. If *n* is a list, then `partition_list` partitions at sucessively deeper levels with elements of *n*. If *n* and *d* are lists, the first elementsof *n* and *d* apply at the highest level and so on. If *n* is a list and *d* is a number, then the offset *d* is used with each of the *n*.

Arguments `partition_list` requires either two or three arguments. The first argument *e* must be non-atomic. The second argument *nlist* must be an integer or a list of integers. The third argument *dlist* must be an integer or a list of integers.

Examples
Partition the numbers from 1 through 10 into pairs.

```
(%i1) partition_list([1,2,3,4,5,6,7,8,9,10],2);
(%o1) [[1,2],[3,4],[5,6],[7,8],[9,10]]
```

9.17 Function: select

select(*expr*, *test* :optional *n*)
mext package: lists_aex

Description Returns a list of all elements of *expr* for which *test* is true. *expr* may have any op. If *n* is supplied, then at most *n* elements are examined. `select` is much faster than `sublist`, but may be less generally applicable.

Arguments `select` requires either two or three arguments. The first argument *expr* must be non-atomic and represented by a lisp list. The third argument *n* must be a positive integer.

Options `select` takes options with default values: `adj->true`, `ot->ml`, `compile->true`.

Examples
Select elements less than 3

```
(%i1) select([1,2,3,4,5,6,7], lambda([x], is(x<3)));
(%o1) [1,2]
```

9.18 Argument type: sequence specifier

mext package: lists_aex

Description A sequence specification specifies a subsequence of the elements in an expression. A single positive number *n* means the first *n* elements. $-n$ means the last *n* elements. A list of three numbers `[i1,i2,i3]` means the `i1`th through the `i2`th stepping by `i3`. If `i1` or `i2` are negative, they count from the end. If `i3` is negative, stepping is down and `i1` must be greater than or equal to `i2`. If `i3` is omitted, it is taken to be 1. A sequence specifiier can also be one of 'all 'none or 'reverse, which mean all elements, no elements or all elements in reverse order respectively.

See also `take` and `string_take`.

9.19 Function: table

mext package: lists_aex

Calling

table(*expr*, [*n*]) Evaluates expression *number* times. If *number* is not an integer or a floating point number, then `float` is called. If we have a floating point number, it is truncated into an integer. This type of iterator is the fastest, since no variable is bound.

table(*expr*, [*variable*, *initial*, *end*, *step*]) Returns a list of evaluated expressions where *variable* (a symbol) is set to a value. The first element of the returned list is *expression* evaluated with *variable* set to *initial*. The i-th element of the returned list is *expression* evaluated with *variable* set to *initial*$+(i-1)$*step*. The iteration stops once the value is greater (if *step* is positive) or smaller (if *step* is negative) than *end*. Requirement: The difference between *end* and *intial* must return a `numberp` number. *step* must be a nonzero `numberp` number. This allows for iterators of rather general forms like `[i, %i - 2, %i, 0.1b0] ...`

table(*expr*, [*variable*, *initial*, *end*]) This iterator uses a step of 1 and is equal to [*variable*,*initial*,*end*, 1].

Arguments `table` requires two or more arguments. The second argument *iterator1* must be a list. Each of the remaining arguments must be a list.

Options `table` takes options with default values: `adj->true`, `ot->ml`.

Attributes table has attributes: [hold_all]

Examples

Make a list of function values

```
(%i1) table(sin(x),[x,0,2*%pi,%pi/4]);
(%o1) [0,1/sqrt(2),1,1/sqrt(2),0,-1/sqrt(2),-1,-1/sqrt(2),0]
```

Make a nested list.

```
(%i1) table( x^y, [x,1,2], [y,1,2]);
(%o1) [[1,1],[2,4]]
```

See also `makelist`, `lrange`, and `constant_list`.

Author Ziga Lenarcic.

9.20 Function: take

take(*e* :rest *v*)
mext package: lists_aex

Calling

take(*e*, *n*) returns a list of the first *n* elements of list or expression *e*.

take(*e*, [*n1*, *n2*]) returns a list of the *n1*th through *n2*th elements of list or expression *e*.

take(*e*, [*n1*, *n2*, *step*]) returns a list of the *n1*th through *n2*th elements stepping by *step* of list or expression *e*.

take(*e*, -n) returns the last *n* elements.

take(*e*, *spec1*, *spec2*, ...) applies the sequence specifications at sucessively deeper levels in *e*.

Description *e* can have mixed lex and aex expressions on different levels. If more sequence specifications are given, they apply to sucessively deeper levels in *e*.

Arguments `take` requires one or more arguments. The first argument *e* must be non-atomic. Each of the remaining arguments must be a sequence specification.

Examples

Take the first 3 elements of a list.

```
(%i1) take([a,b,c,d,e],3);
(%o1) [a,b,c]
```

Take the last 3 elements of a list.

```
(%i1) take([a,b,c,d,e],-3);
(%o1) [c,d,e]
```

Take the second through third elements of a list.

```
(%i1) take([a,b,c,d,e],[2,3]);
(%o1) [b,c]
```

Take the second through tenth elements of a list counting by two.

```
(%i1) take([1,2,3,4,5,6,7,8,9,10],[2,10,2]);
(%o1) [2,4,6,8,10]
```

Take the last through first elements of a list counting backwards by one.

```
(%i1) take([a,b,c,d],[-1,1,-1]);
(%o1) [d,c,b,a]
```

Shorthand for the previous example is 'reverse.

```
(%i1) take([a,b,c,d],'reverse);
(%o1) [d,c,b,a]
```

Take the second through third elements at the first level and the last 2 elements at the second level.

```
(%i1) take([[a,b,c], [d,e,f], [g,h,i]], [2,3],-2);
(%o1) [[e,f],[h,i]]
```

9.21 Function: take_while

take_while(*expr*, *test*)
mext package: lists_aex

Calling

take_while(*expr*, *test*) collects the elements in *expr* until *test* fails on one of them. The op of the returned expression is the same as the op of *expr*.

Arguments `take_while` requires two arguments. The first argument *expr* must be non-atomic and represented by a lisp list.

Options `take_while` takes options with default values: `adj->true`, `ot->ml`, `compile->true`.

Examples
Take elements as long as they are negative.

```
(%i1) take_while([-3,-10,-1,3,6,7,-4], lambda([x], is(x<0)));
(%o1) [-3,-10,-1]
```

9.22 Function: tuples

tuples(*list-or-lists* :optional *n*)
mext package: lists_aex

Calling

tuples(*list*, *n*) Return a list of all lists of length *n* whose elements are chosen from *list*.

tuples([*list1*, *list2*, ...]) Return a list of all lists whose *i*_th element is chosen from *listi*.

Arguments `tuples` requires either one or two arguments. The first argument *list-or-lists* must be non-atomic and represented by a lisp list. The second argument n must be a non-negative integer.

Options `tuples` takes options with default values: `adj->true`, `ot->ml`.

Examples

Make all three letter words in the alphabet 'a,b'.

```
(%i1) tuples([a,b],3);
(%o1) [[a,a,a],[a,a,b],[a,b,a],[a,b,b],[b,a,a],[b,a,b],[b,b,a],[b,b,b]]
```

Take all pairs chosen from two lists.

```
(%i1) tuples([ [0,1] , [x,y,z] ]);
(%o1) [[0,x],[0,y],[0,z],[1,x],[1,y],[1,z]]
```

tuples works for expressions other than lists.

```
(%i1) tuples(f(0,1),3);
(%o1) [f(0,0,0),f(0,0,1),f(0,1,0),f(0,1,1),f(1,0,0),f(1,0,1),f(1,1,0),f(1,1,1)]
```

10 Functions and Variables for Number Theory

- `abundant_p`
- `aliquot_sequence`
- `aliquot_sum`
- `amicable_p`
- `catalan_number`
- `divisor_function`
- `divisor_summatory`
- `from_digits`
- `integer_digits`
- `integer_string`
- `oeis_A092143`
- `perfect_p`
- `prime_pi`
- `prime_pi_soe`
- `prime_twins`
- `primes1`
- `to_poly_clean`

10.1 Function: abundant_p

abundant_p(n)
mext package: discrete_aex

Description Returns true if n is an abundant number. Otherwise, returns false.

Arguments `abundant_p` requires one argument n, which must be a positive integer.

Examples
The abundant numbers between 1 and 100

```
(%i1) select(lrange(100),abundant_p);
(%o1) [12,18,20,24,30,36,40,42,48,54,56,60,66,70,72,78,80,84,88,90,96,100]
```

See also `divisor_function`, `aliquot_sum`, `aliquot_sequence`, `divisor_summatory`, and `perfect_p`.

10.2 Function: aliquot_sequence

aliquot_sequence(k, n)
mext package: discrete_aex

Description The aliquot sequence is a recursive sequence in which each term is the sum of the proper divisors of the previous term. This function returns the first n elements (counting from zero) in the aliquot sequence whose first term is k. The sequence is truncated at an element if it is zero or repeats the previous element.

Arguments `aliquot_sequence` requires two arguments. The first argument k must be a positive integer. The second argument n must be a non-negative integer.

Examples
Perfect numbers give a repeating sequence of period 1.

```
(%i1) imap(lambda([x],aliquot_sequence(x,100)),[6,28,496,8128]);
(%o1) [[6],[28],[496],[8128]]
```

Aspiring numbers are those which are not perfect, but terminate with a repeating perfect number.

```
(%i1) imap(lambda([x],aliquot_sequence(x,100)),[25, 95, 119, 143, 417, 445, 565, 608, 650, 652, 675, 68
(%o1) [[25,6],[95,25,6],[119,25,6],[143,25,6],[417,143,25,6],[445,95,25,6],[565,119,25,6],[608,652,496]
```

See also `divisor_function`, `aliquot_sum`, `divisor_summatory`, `perfect_p`, and `abundant_p`.

10.3 Function: aliquot_sum

aliquot_sum(n)
mext package: discrete_aex

Description Returns the aliquot sum of n. The aliquot sum of n is the sum of the proper divisors of n.

Arguments `aliquot_sum` requires one argument n, which must be a positive integer.

Attributes aliquot_sum has attributes: [match_form]

See also `divisor_function`, `aliquot_sequence`, `divisor_summatory`, `perfect_p`, and `abundant_p`.

10.4 Function: amicable_p

amicable_p(n, m)
mext package: discrete_aex

Description Returns true if n and m are amicable, and false otherwise.

Arguments `amicable_p` requires two arguments. The first argument n must be a positive integer. The second argument m must be a positive integer.

Examples
The first few amicable pairs.

```
(%i1) map(lambda([x],amicable_p(first(x),second(x))), [[220, 284],
          [1184, 1210], [2620, 2924], [5020, 5564], [6232, 6368]]);
(%o1) [true,true,true,true,true]
```

10.5 Function: catalan_number

catalan_number(n)
mext package: discrete_aex

Description Returns the nth catalan number.

Arguments `catalan_number` requires one argument.

Examples
The catalan number for n from 1 through 12.

```
(%i1) map(catalan_number,lrange(12));
(%o1) [1,2,5,14,42,132,429,1430,4862,16796,58786,208012]
```

The n'th catalan number.

```
(%i1) catalan_number(n);
(%o1) binomial(2*n,n)/(n+1)
```

OEIS number: A000108.

10.6 Function: divisor_function

divisor_function(n :optional x)
mext package: discrete_aex

Description Returns the divisor function, or sum of positive divisors function

$$\sigma_x(n) = \sum_{d|n} d^x,$$

where $d|x$ means d divides n. If x is omitted it takes the default value 0. Currently, complex values for x are not supported. After writing this, I noticed that the function is implemented in the maxima core and is callled `divsum`.

Arguments `divisor_function` requires either one or two arguments. The first argument n must be a non-negative integer. The second argument x must be a number.

Attributes divisor_function has attributes: [match_form]

OEIS number: A000005 for x=0 and A000203 for x=1.
See also `aliquot_sum`, `aliquot_sequence`, `divisor_summatory`, `perfect_p`, and `abundant_p`.

10.7 Function: divisor_summatory

divisor_summatory(x)
mext package: discrete_aex

Description Returns the divisor summatory function $D(x)$ for x. The `divisor_function` $\sigma_0(n)$ counts the number of unique divisors of the natural number n. $D(x)$ is the sum of $\sigma_0(n)$ over $n \leq x$.

Arguments `divisor_summatory` requires one argument x, which must be a non-negative number.

Attributes divisor_summatory has attributes: [match_form]

Examples
D(n) for n from 1 through 12

```
(%i1) map(divisor_summatory,lrange(12));
(%o1) [1,3,5,8,10,14,16,20,23,27,29,35]
```

OEIS number: A006218.
See also `divisor_function`, `aliquot_sum`, `aliquot_sequence`, `perfect_p`, and `abundant_p`.

10.8 Function: from_digits

from_digits(*digits* :optional *base*)
mext package: discrete_aex

Calling

from_digits(*digits*) returns the integer represented by the decimal digits in the list or string *digits*.

from_digits(*digits*, *base*) returns the integer represented by the base *base* digits in the list or string *digits*.

Description *base* need not be number, but may be, for instance, a symbol. If *base* is a number it must be an integer between 2 and 36. *digits* may be a string rather than a list.

Arguments `from_digits` requires either one or two arguments. The first argument *digits* must be a list (lex or aex) or a string.

See also `integer_digits` and `integer_string`.

10.9 Function: integer_digits

integer_digits(n :optional *base*, *len*)
mext package: discrete_aex

Calling

integer_digits(n) returns a list of the base 10 digits of n.

integer_digits(n, *base*) returns a list of the base *base* digits of n.

integer_digits(n, *base*, *len*) returns a list of the base *base* digits of n padded with 0's so that the total length of the list is *len*.

Arguments `integer_digits` requires between one and three arguments. The first argument n must be an integer. The second argument *base* must be a valid radix (an integer between 2 and 36). The third argument *len* must be a non-negative integer.

Options `integer_digits` takes options with default values: `adj->true`, `ot->ml`.

See also `from_digits` and `integer_string`.

Implementation gcl is much faster than the others. `integer_digits(2^(10^6))`: typical times for lisps: ecl-12.12.1 0.09s, sbcl-1.1.11 0.5s, clisp-2.49 9s, ccl-1.9 62s, cmucl-20d error, gcl-2.6.(7,8,9) 0.09s, allegro-8.2 = 23s, Mma-3.0 = 5s, Mma-8 = 0.04s.

10.10 Function: integer_string

integer_string(n :optional *base*, *pad*)
mext package: discrete_aex

Calling

integer_string(n) returns a string containing the decimal digits of the integer n.

integer_string(n, *base*) returns a string containing the base *base* digits of the integer n.

integer_string(n, *base*, *pad*) pads the string on the left with 0's so that the length of the string is *pad*.

integer_string(n, "roman") returns a string containing the roman-numeral form of the integer n.

integer_string(n, "cardinal") returns a string containing the english word form of the integer (cardinal number) n.

integer_string(n, "ordinal") returns a string containing the english word form of the ordinal (counting) number n.

Arguments `integer_string` requires between one and three arguments. The first argument n must be an integer. The second argument *base* must be a valid radix (an integer between 2 and 36) or a string. The third argument *pad* must be a positive integer.

See also `integer_digits` and `from_digits`.

10.11 Function: oeis_A092143

oeis_A092143(n)
mext package: discrete_aex

Description Returns the cumulative product of all divisors of integers from 1 to n.

Arguments `oeis_A092143` requires one argument n, which must be a positive integer.

10.12 Function: perfect_p

perfect_p(n)
mext package: discrete_aex

Description Returns true if n is a perfect number. Otherwise, returns false.

Arguments `perfect_p` requires one argument n, which must be a positive integer.

See also `divisor_function`, `aliquot_sum`, `aliquot_sequence`, `divisor_summatory`, and `abundant_p`.

Implementation This function computes divisors. It would be far more efficient to use a table of known perfect numbers, as very few of them are accessible by current computer hardware.

10.13 Function: prime_pi

prime_pi(n)
mext package: prime_pi

Calling

prime_pi(n) returns the number of primes less than or equal to n.

Description Computes the prime counting function. The option *threads* specifies the maximum number of cpu threads to use. The routine may use fewer threads, depending on the value of n. The percent of the calculation that is finished is printed during the calculation if the option *status* is true. The status will only work under some terminals.

Arguments `prime_pi` requires one argument n, which must be equivalent to an unsigned 64 bit integer (that is, an integer between 0 and 2 to the power 64) (We need to modify the doc system so we can use notation for powers in arg check strings. .

Options `prime_pi` takes options with default values: `status->false`, `threads->1`.

See also `prime_pi_soe`, `next_prime`, and `prev_prime`.

Implementation This algorithm is fast, for a general purpose mathematics program. It combines a segmented sieve implemented as a C library with tables.

Authors Kim Walisch (C library), Tomas Oliveira e Silva (tables), and John Lapeyre (lisp).

10.14 Function: prime_pi_soe

prime_pi_soe(n)
mext package: discrete_aex

Description The prime counting function. The algorithm is the sieve of Eratosthenes. Internally an array of n bits is used.

Arguments `prime_pi_soe` requires one argument n, which must be a non-negative integer.

See also `prime_pi`, `next_prime`, and `prev_prime`.

Implementation This is not the most efficient way to compute primes.

10.15 Function: prime_twins

prime_twins(*min* :optional *max*)
mext package: prime_pi

Calling

prime_twins(n) returns the number of prime twins less than or equal to n.

prime_twins(*nmin*, *nmax*) returns the number of prime twins between *nmin* and *max*.

Description The option *ktuplet* counts the *ktuplet*-constellation rather than the twins. *ktuplet* must be an integer between 1 and 7.

Arguments `prime_twins` requires either one or two arguments. The first argument *min* must be equivalent to an unsigned 64 bit integer (that is, an integer between 0 and 2 to the power 64) (We need to modify the

doc system so we can use notation for powers in arg check strings. . The second argument *max* must be equivalent to an unsigned 64 bit integer (that is, an integer between 0 and 2 to the power 64) (We need to modify the doc system so we can use notation for powers in arg check strings. .

Options `prime_twins` takes options with default values: `ktuplet->2`, `status->false`, `threads->1`.

See also `prime_pi`, `next_prime`, `prev_prime`, and `primep`.

Implementation No tables are used in this algorithm.

10.16 Function: primes1

primes1(*n1* :optional *n2*)
mext package: discrete_aex

Calling

primes1(*max*) returns a list of the primes less than or equal to *max*.

primes1(*min*, *max*) returns a list of the primes between *min* and *max*.

Description The algorithm is the sieve of Eratosthenes. This is not an efficient algorithm.

Arguments `primes1` requires either one or two arguments. The first argument *n1* must be a non-negative integer. The second argument *n2* must be a non-negative integer.

Options `primes1` takes options with default values: `adj->true`, `ot->ml`.

10.17 Function: to_poly_clean

to_poly_clean()
mext package: tpsolve

Description Removes the temporary variables created by `to_poly`. These are also created by `to_poly_solve`. They can be created from maxima with `new_variable`. These variables can be found in the infolist `props`. `to_poly_clean` returns the number of variables removed.

Arguments `to_poly_clean` requires zero arguments.

11 Functions and Variables for Numerical Computation

Functions for numerical computations: Numeric integration; Mathematical functions— cos,sin,etc. —that accept only numerical arguments. Tests of loops in untranslated code show that these are much more efficient than using the standard maxima versions. But, for most applications, the standard Maxima versions are probably ok.

- `mquad_qag`
- `mquad_qagi`
- `mquad_qagp`
- `mquad_qags`
- `mquad_qawc`
- `mquad_qawf`

- mquad_qawo
- mquad_qaws
- n_abs
- n_acos
- n_acosh
- n_asin
- n_asinh
- n_atan
- n_atanh
- n_cos
- n_cosh
- n_exp
- n_expt
- n_log
- n_sin
- n_sinh
- n_sqrt
- n_tan
- n_tanh
- nintegrate

11.1 Function: mquad_qag

mquad_qag(*fun*, *var*, *a*, *b*, *key*)
mext package: numerical

Description This is an interface to qag that is modified from quad_qag.

Arguments mquad_qag requires five arguments. The second argument *var* must be a symbol or a subscripted variable. The third argument *a* must be an expression that can be converted to a float. The fourth argument *b* must be an expression that can be converted to a float. The fifth argument *key* must be an integer between 1 and 6.

Options mquad_qag takes options with default values: match->false, epsrel->1.e-8, epsabs->0.0, limit->200.

11.2 Function: mquad_qagi

mquad_qagi(*fun*, *var*, *a*, *b*)
mext package: numerical

Description This is an interface to qagi that is modified from `quad_qagi`.

Arguments `mquad_qagi` requires four arguments. The second argument *var* must be a symbol or a subscripted variable. The third argument *a* must be inf, minf, or an expression that can be converted to a float. The fourth argument *b* must be inf, minf, or an expression that can be converted to a float.

Options `mquad_qagi` takes options with default values: `match->false`, `epsrel->1.e-8`, `epsabs->0.0`, `limit->200`.

11.3 Function: mquad_qagp

mquad_qagp(*fun*, *var*, *a*, *b*, *points*)
mext package: numerical

Description This is an interface to qagp that is modified from `quad_qagp`.

Arguments `mquad_qagp` requires five arguments. The second argument *var* must be a symbol or a subscripted variable. The third argument *a* must be an expression that can be converted to a float. The fourth argument *b* must be an expression that can be converted to a float. The fifth argument *points* must be a list.

Options `mquad_qagp` takes options with default values: `match->false`, `epsrel->1.e-8`, `epsabs->0.0`, `limit->200`.

11.4 Function: mquad_qags

mquad_qags(*fun*, *var*, *a*, *b*)
mext package: numerical

Description This is an interface to qags that is modified from `quad_qags`.

Arguments `mquad_qags` requires four arguments. The second argument *var* must be a symbol or a subscripted variable. The third argument *a* must be an expression that can be converted to a float. The fourth argument *b* must be an expression that can be converted to a float.

Options `mquad_qags` takes options with default values: `match->false`, `epsrel->1.e-8`, `epsabs->0.0`, `limit->200`.

11.5 Function: mquad_qawc

mquad_qawc(*fun*, *var*, *c*, *a*, *b*)
mext package: numerical

Description This is an interface to qawc that is modified from `quad_qawc`.

Arguments `mquad_qawc` requires five arguments. The second argument *var* must be a symbol or a subscripted variable. The third argument *c* must be an expression that can be converted to a float. The fourth argument *a* must be an expression that can be converted to a float. The fifth argument *b* must be an expression that can be converted to a float.

Options `mquad_qawc` takes options with default values: `match->false`, `epsrel->1.e-8`, `epsabs->0.0`, `limit->200`.

11.6 Function: mquad_qawf

mquad_qawf(*fun*, *var*, *a*, *omega*, *trig*)
mext package: numerical

Description This is an interface to qawf that is modified from `quad_qawf`.

Arguments `mquad_qawf` requires five arguments. The second argument *var* must be a symbol or a subscripted variable. The third argument *a* must be an expression that can be converted to a float. The fourth argument *omega* must be an expression that can be converted to a float. The fifth argument *trig* must be one of '($COS 1

Options `mquad_qawf` takes options with default values: `match->false`, `limlst->10`, `maxp1->100`, `epsabs->1.e-10`, `limit->200`.

11.7 Function: mquad_qawo

mquad_qawo(*fun*, *var*, *a*, *b*, *omega*, *trig*)
mext package: numerical

Description This is an interface to qawo that is modified from `quad_qawo`.

Arguments `mquad_qawo` requires six arguments. The second argument *var* must be a symbol or a subscripted variable. The third argument *a* must be an expression that can be converted to a float. The fourth argument *b* must be an expression that can be converted to a float. The fifth argument *omega* must be an expression that can be converted to a float. The sixth argument *trig* must be one of '($COS 1

Options `mquad_qawo` takes options with default values: `match->false`, `epsrel->1.e-8`, `epsabs->0.0`, `limit->200`, `maxp1->100`.

11.8 Function: mquad_qaws

mquad_qaws(*fun*, *var*, *a*, *b*, *alfa*, *beta*, *wfun*)
mext package: numerical

Description This is an interface to qaws that is modified from `quad_qaws`.

Arguments `mquad_qaws` requires seven arguments. The second argument *var* must be a symbol or a subscripted variable. The third argument *a* must be an expression that can be converted to a float. The fourth argument *b* must be an expression that can be converted to a float. The fifth argument *alfa* must be an expression that can be converted to a float. The sixth argument *beta* must be an expression that can be converted to a float. The seventh argument *wfun* must be an integer between 1 and 4.

Options `mquad_qaws` takes options with default values: `match->false`, `epsrel->1.e-8`, `epsabs->0.0`, `limit->200`.

11.9 Function: n_abs

mext package: aex

Description n_abs calls the lisp numeric function ?abs. This function accepts only float or integer arguments from maxima (lisp complex and rationals, as well.). n_abs may be considerably faster in some code, particularly untranslated code.

11.10 Function: n_acos

mext package: aex

Description n_acos calls the lisp numeric function ?acos. This function accepts only float or integer arguments from maxima (lisp complex and rationals, as well.). n_acos may be considerably faster in some code, particularly untranslated code.

11.11 Function: n_acosh

mext package: aex

Description n_acosh calls the lisp numeric function ?acosh. This function accepts only float or integer arguments from maxima (lisp complex and rationals, as well.). n_acosh may be considerably faster in some code, particularly untranslated code.

11.12 Function: n_asin

mext package: aex

Description n_asin calls the lisp numeric function ?asin. This function accepts only float or integer arguments from maxima (lisp complex and rationals, as well.). n_asin may be considerably faster in some code, particularly untranslated code.

11.13 Function: n_asinh

mext package: aex

Description n_asinh calls the lisp numeric function ?asinh. This function accepts only float or integer arguments from maxima (lisp complex and rationals, as well.). n_asinh may be considerably faster in some code, particularly untranslated code.

11.14 Function: n_atan

mext package: aex

Description n_atan calls the lisp numeric function ?atan. This function accepts only float or integer arguments from maxima (lisp complex and rationals, as well.). n_atan may be considerably faster in some code, particularly untranslated code.

11.15 Function: n_atanh

mext package: aex

Description n_atanh calls the lisp numeric function ?atanh. This function accepts only float or integer arguments from maxima (lisp complex and rationals, as well.). n_atanh may be considerably faster in some code, particularly untranslated code.

11.16 Function: n_cos

mext package: aex

Description n_cos calls the lisp numeric function ?cos. This function accepts only float or integer arguments from maxima (lisp complex and rationals, as well.). n_cos may be considerably faster in some code, particularly untranslated code.

11.17 Function: n_cosh

mext package: aex

Description n_cosh calls the lisp numeric function ?cosh. This function accepts only float or integer arguments from maxima (lisp complex and rationals, as well.). n_cosh may be considerably faster in some code, particularly untranslated code.

11.18 Function: n_exp

mext package: aex

Description n_exp calls the lisp numeric function ?exp. This function accepts only float or integer arguments from maxima (lisp complex and rationals, as well.). n_exp may be considerably faster in some code, particularly untranslated code.

11.19 Function: n_expt

mext package: aex

Description n_expt calls the lisp numeric function ?expt. This function accepts only float or integer arguments from maxima (lisp complex and rationals, as well.). n_expt may be considerably faster in some code, particularly untranslated code.

11.20 Function: n_log

mext package: aex

Description n_log calls the lisp numeric function ?log. This function accepts only float or integer arguments from maxima (lisp complex and rationals, as well.). n_log may be considerably faster in some code, particularly untranslated code.

11.21 Function: n_sin

mext package: aex

Description n_sin calls the lisp numeric function ?sin. This function accepts only float or integer arguments from maxima (lisp complex and rationals, as well.). n_sin may be considerably faster in some code, particularly untranslated code.

11.22 Function: n_sinh

mext package: aex

Description n_sinh calls the lisp numeric function ?sinh. This function accepts only float or integer arguments from maxima (lisp complex and rationals, as well.). n_sinh may be considerably faster in some code, particularly untranslated code.

11.23 Function: n_sqrt

mext package: aex

Description n_sqrt calls the lisp numeric function ?sqrt. This function accepts only float or integer arguments from maxima (lisp complex and rationals, as well.). n_sqrt may be considerably faster in some code, particularly untranslated code.

11.24 Function: n_tan

mext package: aex

Description n_tan calls the lisp numeric function ?tan. This function accepts only float or integer arguments from maxima (lisp complex and rationals, as well.). n_tan may be considerably faster in some code, particularly untranslated code.

11.25 Function: n_tanh

mext package: aex

Description n_tanh calls the lisp numeric function ?tanh. This function accepts only float or integer arguments from maxima (lisp complex and rationals, as well.). n_tanh may be considerably faster in some code, particularly untranslated code.

11.26 Function: nintegrate

nintegrate(*expr*, *varspec* :optional *singlist*)
mext package: numerical

Description Numerically integrate *expr*, with the variable and limits supplied in the list *varspec* as [*var*,*lo*,*hi*]. Only one-dimensional integrals are implemented. `nintegrate` automatically chooses and combines `qags`,`qagp`, and `qagi`. Some support for complex numbers is implemented. Some integrable singularities are found automatically.

If the option *call* is true, then calls made to quadpack are also returned in a list. If *call* is *short*, then only the name of the quadpack routine is included.

By default, information on the integration is returned with the results. If the option *info* is false, then only the result of the integration is returned.

If the option *sing* is false, then `nintegrate` will not search for internal singularities, but user supplied singularities will still be used.

This function is not well tested and may give incorrect results.

See the Maxima documentation for quadpack.

Arguments `nintegrate` requires either two or three arguments. The second argument *varspec* must be a list. The third argument *singlist* must be a list.

Options `nintegrate` takes options with default values: `epsrel->1.e-8`, `epsabs->0`, `subint->200`, `info->true`, `words->true`, `calls->false`, `sing->true`.

Examples

```
(%i1) nintegrate(sin(sin(x)), [x,0,2]);
(%o1) [1.24706, 1.38451e-14, 21, no problems]
```

Integrate over a semi-infinite interval with an internal singularity. The location of the singularity is supplied. This cannot be done with a single call to quadpack routines.

```
(%i2) nintegrate(1/sqrt(abs(1-x))*exp(-x),[x,0,inf], [1] );
(%o2) [1.72821, 1.87197e-10, 660, no problems]
```

If a list of possible singular points is not supplied, then they will be searched for using `solve`.

```
(%i3) nintegrate(1/sqrt(abs(1-x))*exp(-x),[x,0,inf]);
(%o3) [1.72821, 1.87197e-10, 660, no problems]
```

In some cases, complex numbers are treated correctly. (The simplifier replaces `cos(%i*x)` with `cosh(x)` before the routine is called. So this works with `quad_qags` as well.)

```
(%i4) nintegrate( cos(%i*x), [x,0,1]);
(%o4) [1.1752, 1.30474e-14, 21, no problems]
```

```
(%i5) sinh(1.0);
(%o5) 1.1752
```

But, the quadpack routines cannot handle the complex numbers in this example.

```
(%i6) nintegrate(exp(%i*x) * exp(-x*x), [x,0,inf]);
(%o6) [.424436 %i + .690194, 4.988325e-9, 300, no problems]
```

Return quadpack error code rather than error text.

```
(%i7) nintegrate(sin(sin(x)), [x,0,2], words->false);
(%o7) [1.24706, 1.38451e-14, 21, 0]
```

Request a relative error.

```
(%i8) nintegrate(exp(%i*x) * exp(-x*x), [x,0,inf], epsrel -> 1e-12);
(%o8) [.424436 %i + .690194, 1.06796e-13, 480, no problems]
```

Trying to do the integral with too few sub-intervals fails.

```
(%i9) nintegrate(1/(1+x^2), [x, 0, inf], subint -> 2, epsrel -> 1e-10);
(%o9) [1.5708, 2.57779e-10, 45, too many sub-intervals]
```

This integral is not handled well. Giving limits of `minf` and `inf` fails.

```
(%i10) nintegrate(exp(%i*x*x),[x,-200,200],subint->10000);
(%o10) [1.25170114 %i + 1.25804682, 2.507635982e-8, 760578, no problems]
```

```
(%i11) integrate(exp(%i*x*x),x,minf,inf);
(%o11) sqrt(%pi)*(%i/sqrt(2)+1/sqrt(2))
```

```
(%i12) rectform(float(%));
(%o12) 1.25331414*%i+1.25331414
```

Return a list of calls made to quadpack.

```
(%i13) nintegrate(1/sqrt(abs(1-x)) * exp(-x),[x,0,inf], calls->true);
(%o13) [1.72821,1.87197e-10,660, no problems,
 [quad_qagi(%e^-x/sqrt(abs(x-1)),x,1.0,inf,epsrel = 1.e-8,epsabs = 0,limit = 200),
  quad_qagp(%e^-x/sqrt(abs(x-1)),x,0,1.0,[],epsrel = 1.e-8,epsabs = 0,limit = 200)]]
```

Here we must supply the roots of `sin(x)` within the range of integration.

```
(%i14) nintegrate(1/(sqrt(sin(x))),[x,0,10], [%pi,2*%pi,3*%pi]);
(%o14) [10.48823021716687 - 6.769465521725385 %i, 9.597496930524585e-8, 1596, no problems]
```

See also `quad_qags`, `quad_qagi`, and `quad_qagp`.

12 Functions and Variables for Numerics

These are mathematical functions— cos,sin,etc. —that accept only numerical arguments. Tests of loops in untranslated code show that these are much more efficient than using the standard maxima versions. But, for most applications, the standard maxima versions are probably ok.

13 Functions and Variables for Predicates

- `aex_p`
- `cmplength`
- `length0p`
- `length1p`
- `length_eq`
- `type_of`

13.1 Function: aex_p

aex_p(e)
mext package: aex

Description Returns true if e is an aex expression, otherwise false.

Arguments `aex_p` requires one argument.

13.2 Function: cmplength

cmplength(e, n)
mext package: aex

Description return the smaller of n and `length(e)`. This is useful if e is very large and n is small, so that computing the entire length of e is inefficient. Expression e can be either a lex of aex expression.

Arguments `cmplength` requires two arguments. The second argument n must be a non-negative integer.

See also `length0p`, `length_eq`, and `length1p`.

Implementation cmplength is implemented with defmfun1, which slows things down a bit. So be cautious using it in a tight loop.

13.3 Function: length0p

length0p(e)
mext package: aex

Description Returns true if ¡e¿ is of length 0, false otherwise. This implementation traverse no more elements of ¡e¿ than necessary to return the result.

Arguments `length0p` requires one argument e, which must be a string or non-atomic.

See also `cmplength`, `length_eq`, and `length1p`.

Implementation length0p is implemented with defmfun1, which slows things down a bit. So be cautious using it in a tight loop.

13.4 Function: length1p

length1p(*e*)
mext package: aex

Description Returns true if *e* is of length 1, false otherwise. This implementation traverse no more elements of *e* than necessary to return the result.

Arguments `length1p` requires one argument *e*, which must be a string or non-atomic.

See also `length0p`, `cmplength`, and `length_eq`.

Implementation length1p is implemented with defmfun1, which slows things down a bit. So be cautious using it in a tight loop.

13.5 Function: length_eq

length_eq(*e*, *n*)
mext package: aex

Description Returns true if *e* is of length *n*, false otherwise. This implementation traverses no more elements of *e* than necessary to return the result.

Arguments `length_eq` requires two arguments. The first argument *e* must be a string or non-atomic. The second argument *n* must be a non-negative integer.

See also `length0p`, `cmplength`, and `length1p`.

Implementation length_eq is implemented with defmfun1, which slows things down a bit. So be cautious using it in a tight loop.

13.6 Function: type_of

type_of(*e*)
mext package: aex

Description Return something like the 'type' of a maxima expression. This is a bit ill defined currently. `type_of` uses the lisp function `type-of`.

If the option *verbose* is true, then more information is returned.

Arguments `type_of` requires one argument.

Options `type_of` takes options with default values: `verbose->false`.

Examples

```
(%i1) type_of(1);
(%o1) integer
(%i1) type_of(1.0);
(%o1) lisp_double_float
(%i2) type_of(1.0b0);
(%o2) bfloat
(%i3) type_of(1/3);
(%o3) /
(%i4) type_of("dog");
(%o4) string
(%i5) type_of([1,2,3]);
(%o5) [
```

```
(%i6) type_of(aex([1,2,3]));
(%o6) [
(%i7) type_of(%e);
(%o7) symbol
(%i8) type_of(%i);
(%o8) symbol
(%i9) type_of(%i+1);
(%o9) +
```

type_of returns the type of the lisp struct corresponding to a maxima object.

```
(%i1) load(graphs)$
(%i2) type_of(new_graph());
(%o2)   graph
```

14 Functions and Variables for Program Flow

- `error_str`

14.1 Function: error_str

error_str()
mext package: defmfun1

Description Returns the last error message as a string. This differs from `errormsg`, which prints the error message.

Arguments `error_str` requires zero arguments.

See also `error` and `errormsg`.

15 Functions and Variables for Quicklisp

- `quicklisp_apropos`
- `quicklisp_install`
- `quicklisp_load`
- `quicklisp_start`

15.1 Function: quicklisp_apropos

quicklisp_apropos(*term*)
mext package: quicklisp

Description Search quicklisp for lisp 'systems' (packages) matching *term*.

Arguments `quicklisp_apropos` requires one argument *term*, which must be a string.

15.2 Function: quicklisp_install

quicklisp_install()
mext package: quicklisp

Description Download and install quicklisp from the internet. This is usually done automatically as the final step of building and installing the maxima interface to quicklisp.

Arguments `quicklisp_install` requires zero arguments.

15.3 Function: quicklisp_load

quicklisp_load(*package_name*)
mext package: quicklisp

Description Load the asdf lisp package *package_name*, or, if not installed, install from the internet and then load.

Arguments `quicklisp_load` requires one argument *package_name*, which must be a string.

15.4 Function: quicklisp_start

quicklisp_start()
mext package: quicklisp

Description Load (setup) quicklisp. It must already be installed.

Arguments `quicklisp_start` requires zero arguments.

16 Functions and Variables for Runtime Environment

- `allow_kill`
- `allow_kill_share`
- `chdir`
- `dir_exists`
- `dirstack`
- `dont_kill`
- `dont_kill_share`
- `get_dont_kill`
- `homedir`
- `lisp_bin_ext`
- `lisp_type`
- `lisp_version`
- `list_directory`
- `maxima_version`

- mcompile_file
- mext_clear
- mext_find_package
- mext_info
- mext_list
- mext_list_loaded
- mext_list_package
- mext_test
- mtranslate_file
- popdir
- probe_file
- pwd
- require
- timing
- truename
- updir

16.1 Function: allow_kill

allow_kill(:rest *items*)
mext package: mext_defmfun1

Description Remove *item*s from the list of symbols that are not killed by `kill(all)`. This facility is part of the maxima core, but is apparantly unused. Maybe putting a property in the symbol's property list would be better.

Arguments `allow_kill` requires zero or more arguments.

Attributes allow_kill has attributes: [hold_all]

See also `dont_kill`, `dont_kill_share`, and `get_dont_kill`.

16.2 Function: allow_kill_share

allow_kill_share(*package*)
mext package: mext_defmfun1

Description Allow symbols in maxima share package *package* from being killed by `kill`. This undoes the effect of `dont_kill_share`. Currently (if this document is up-to-date) only 'basic' and 'lrats' are in the database.

Arguments `allow_kill_share` requires one argument *package*, which must be a string or a symbol.

16.3 Function: chdir

chdir(:optional *dir*)
mext package: mext_defmfun1

Calling

chdir() Set the working directory to the value it had when mext was loaded.

chdir(*dir*) Set the working directory to *dir*.

Description Set the working directory for maxima/lisp. With some lisps, such as cmu lisp the system directory is changed as well. This should be made uniform across lisp implementations.

Arguments `chdir` requires either zero or one arguments. If present, the argument *dir* must be a a string.

See also `pwd`, `popdir`, `updir`, `dirstack`, and `list_directory`.

16.4 Function: dir_exists

dir_exists(*dir*)
mext package: mext_defmfun1

Description Returns the pathname as a string if *dir* exists, and `false` otherwise.

Arguments `dir_exists` requires one argument *dir*, which must be a string.

16.5 Function: dirstack

dirstack()
mext package: mext_defmfun1

Description Return a list of the directories on the directory stack. This list is manipulated with `chdir`, `updir`, and `popdir`.

Arguments `dirstack` requires zero arguments.

See also `chdir`, `pwd`, `popdir`, `updir`, and `list_directory`.

16.6 Function: dont_kill

dont_kill(:rest *item*)
mext package: mext_defmfun1

Description Add the *item*s to the list of symbols that are not killed by `kill(all)`. This facility is part of the maxima core, but is apparantly unused. Maybe putting a property in the symbol's property list would be better.

Arguments `dont_kill` requires zero or more arguments.

Attributes dont_kill has attributes: [hold_all]

See also `dont_kill_share`, `get_dont_kill`, and `allow_kill`.

16.7 Function: dont_kill_share

dont_kill_share(*package*)
mext package: mext_defmfun1

Description Prevent symbols in maxima share package *package* from being killed by `kill`. Currently (if this document is up-to-date) only 'basic' and 'lrats' are in the database.

Arguments `dont_kill_share` requires one argument *package*, which must be a string or a symbol.

See also `dont_kill`, `get_dont_kill`, and `allow_kill`.

16.8 Function: get_dont_kill

get_dont_kill()
mext package: mext_defmfun1

Description Returns the list of symbols that are not killed by `kill(all)`. Items are added to this list with `dont_kill`.

Arguments `get_dont_kill` requires zero arguments.

See also `dont_kill`, `dont_kill_share`, and `allow_kill`.

16.9 Variable: homedir

mext package: mext_defmfun1

default value `/home/jlapeyre/`.
Description The user's home directory.

16.10 Variable: lisp_bin_ext

mext package: mext_defmfun1

default value `fasl`.
Description The extension of compiled lisp binaries for the lisp implementation used by Maxima. This should be read-only. Setting it has no effect.

16.11 Variable: lisp_type

mext package: mext_defmfun1

default value `SBCL`.
Description The name of the lisp implementation on which Maxima is running.

16.12 Variable: lisp_version

mext package: mext_defmfun1

default value `1.1.11`.
Description The lisp version number of the lisp implementation on which Maxima is running.

16.13 Function: list_directory

list_directory(:optional *dir*)
mext package: mext_defmfun1

Description Returns a directory listing for *dir* or the current directory if no argument is given.

Arguments `list_directory` requires either zero or one arguments. If present, the argument *dir* must be a a string.

See also `chdir`, `pwd`, `popdir`, `updir`, and `dirstack`.

16.14 Variable: maxima_version

mext package: mext_defmfun1

default value `5.31.0`.
Description The Maxima version number.

16.15 Function: mcompile_file

mcompile_file(*input-file* :optional *bin_file*)
mext package: mext_defmfun1

Description Like `compile_file`, except that the intermediate, translated filename may be specified as an option. If the intermediate filename, *is not given, then* ,it will be written in the same directory as the ouput (binary) file.

Arguments `mcompile_file` requires either one or two arguments. The first argument *input-file* must be a string. The second argument *bin_file* must be a string.

Options `mcompile_file` takes options with default values: `tr_file->false`.

16.16 Function: mext_clear

mext_clear()
mext package: mext_defmfun1

Description Clears the list of mext packages that have been loaded with require. Subsequent calls to require will reload the packages.

Arguments `mext_clear` requires zero arguments.

See also `mext_list_loaded`, `mext_list`, `mext_info`, `mext_list_package`, and `mext_find_package`.

16.17 Function: mext_find_package

mext_find_package(:rest *items*)
mext package: mext_defmfun1

Description Find mext packages in which the function or variable *items* are defined. This only works if the package has been loaded, and its symbols registered. If more than one package is found, then all are listed. If the option *file* is true, then the filename in which the item is defined is also returned.

Arguments `mext_find_package` requires zero or more arguments. Each of the remaining arguments must be a string or a symbol.

Options `mext_find_package` takes options with default values: `file->false`.

See also `mext_list_loaded`, `mext_list`, `mext_info`, `mext_clear`, and `mext_list_package`.

16.18 Function: mext_info

mext_info(*distname*)
mext package: mext_defmfun1

Description Print information about installed mext distribution *distname*. The list of installed distributions is built by calling `mext_list`.

Arguments `mext_info` requires one argument *distname*, which must be a string or a symbol.

See also `mext_list_loaded`, `mext_list`, `mext_clear`, `mext_list_package`, and `mext_find_package`.

16.19 Function: mext_list

mext_list()
mext package: mext_defmfun1

Description Returns a list of all installed mext distributions. These are installed, but not neccessarily loaded.

Arguments `mext_list` requires zero arguments.

See also `mext_list_loaded`, `mext_info`, `mext_clear`, `mext_list_package`, and `mext_find_package`.

16.20 Function: mext_list_loaded

mext_list_loaded()
mext package: mext_defmfun1

Description Returns a list of mext packages currently loaded.

Arguments `mext_list_loaded` requires zero arguments.

See also `mext_list`, `mext_info`, `mext_clear`, `mext_list_package`, and `mext_find_package`.

16.21 Function: mext_list_package

mext_list_package(*package*)
mext package: mext_defmfun1

Description List functions and variables defined in the mext pacakge *package*. A mis-feature is that an empty list is returned if the package is not loaded. This function incorrectly returns an empty list for some packages, and may miss some functions.

Arguments `mext_list_package` requires one argument *package*, which must be a string or a symbol.

See also `mext_list_loaded`, `mext_list`, `mext_info`, `mext_clear`, and `mext_find_package`.

16.22 Function: mext_test

mext_test(:rest *dists*)
mext package: mext_defmfun1

Description Run the test suites for a mext distribution or list of distributions. If the argument `all` is given, then all tests are run for all installed mext distributions. If the argument `loaded` is given, then all tests are run for all loaded mext distributions. If no argument is given, a subfolder named `rtests` is searched

for in the current directory. An item may be a list, in which case, the first element is the package name and the remaining elements are strings specifying the name of the rtests to run. The strings must not include directory or file extension parts. If the option *list* is `true`, then the tests are not performed, but a list of the rtest files is returned. If *list* is *long*, then the full pathnames of the rtest files are listed. Note: if the package `mext_defmfun1` is not loaded, then only a rudimentary version of `mext_test`, which does not accept options, is available.

Arguments `mext_test` requires zero or more arguments. Each of the remaining arguments must be a string, a symbol, or a list of strings or symbols.

Options `mext_test` takes options with default values: `list->false`.

Examples Run regression tests for the packages `aex`, and `lists_aex`.

```
(%i1) mext_test(aex,lists_aex);
(%o1) done
```

Run only some of the regression tests.

```
(%i2) mext_test([aex, "rtest_aex"],[lists_aex, "rtest_table"]);
(%o2) done
```

Only list the test files; do not run them.

```
(%i3) mext_test(tpsolve, list->true);
(%o3) [rtest_to_poly_solve, rtest_to_poly]
```

16.23 Function: mtranslate_file

mtranslate_file(*input-file* :optional *ttymsgsp*)
mext package: mext_defmfun1

Description Like `translate_file`, except that the output filename may be specified as an option.

Arguments `mtranslate_file` requires either one or two arguments. The first argument *input-file* must be a string.

Options `mtranslate_file` takes options with default values: `output_file->false`.

16.24 Function: popdir

popdir(:optional *n*)
mext package: mext_defmfun1

Description Pop a value from the current directory stack and chdir to this value. If *n* is given, pop *n* values and chdir to the last value popped.

Arguments `popdir` requires either zero or one arguments. If present, the argument *n* must be a a non-negative integer.

See also `chdir`, `pwd`, `updir`, `dirstack`, and `list_directory`.

16.25 Function: probe_file

mext package: maxdoc

Calling

> **probe_file**(*filespec*) returns a string representing a canonical pathname to the file specified by *filespec*. False is returned if the file can't be found.

Description Probe_File tries to find a canonical pathname for a file specified by the string *filespec*.

Examples

```
(%i1) probe_file("a/b.txt");
(%o1) "/home/username/c/a/b.txt"
```

16.26 Function: pwd

pwd()
mext package: mext_defmfun1

Description Return the current working directory.

Arguments `pwd` requires zero arguments.

See also `chdir`, `popdir`, `updir`, `dirstack`, and `list_directory`.

16.27 Function: require

require(*distname* :optional *force*)
mext package: mext_defmfun1

Description Load the mext pacakge *distname* and register that it has been loaded. `require('all)` will load all installed mext packages. If *force* is true, then *distname* is loaded even if it has been loaded previously. *distname* may also be a list of package names to be loaded.

Arguments `require` requires either one or two arguments. The first argument *distname* must be a string, a symbol, or a list of strings or symbols.

16.28 Function: timing

timing(:rest *exprs*)
mext package: runtime

Calling

> **timing**(*expr*) evaluates each of the expressions *expr* and returns a list of the time in seconds used, together with the result of evaluating the final expression.
>
> **timing**(*expr*, *print-¿true*) returns the result of evaluating the final expression and prints the time in seconds used.
>
> **timing**(*expr*, *result-¿false*) returns the time in seconds used, and discards all results.

Description `timing` evaluates each of the *exprs*, and returns a list of the total time in seconds used, together the result of the last expression. See also `showtime`.

Arguments `timing` requires zero or more arguments.

Options `timing` takes options with default values: `result->true`, `print->false`.

Attributes timing has attributes: [hold_all]

16.29 Function: truename

mext package: maxdoc

Calling

truename(*filespec*) returns a string representing a canonical pathname to the file specified by *filespec*

Description Truename tries to find a canonical pathanme for a file specified by the string *filespec*.

16.30 Function: updir

updir(:optional *n*)
mext package: mext_defmfun1

Description Change the working directory to be n (or 1 if n is not given) subdirectories higher than the current working directory.

Arguments `updir` requires either zero or one arguments. If present, the argument n must be a a non-negative integer.

See also `chdir`, `pwd`, `popdir`, `dirstack`, and `list_directory`.

17 Functions and Variables for Strings

- `string_drop`
- `string_reverse`
- `string_take`
- `with_output_to_string`

17.1 Function: string_drop

string_drop(*s*, *spec*)
mext package: lists_aex

Arguments `string_drop` requires two arguments. The first argument s must be a string. The second argument *spec* must be a sequence specification.

Examples

```
(%i1)  string_drop("abracadabra",1);
(%o1) bracadabra
```

```
(%i1)  string_drop("abracadabra",-1);
(%o1) abracadabr
```

```
(%i1)  string_drop("abracadabra",[2,10]);
(%o1) aa
```

17.2 Function: string_reverse

string_reverse(*s*)
mext package: lists_aex

Calling

string_reverse(*s*) returns a copy of string *s* with the characters in reverse order.

Arguments `string_reverse` requires one argument *s*, which must be a string.

17.3 Function: string_take

string_take(*s*, *spec*)
mext package: lists_aex

Calling

string_take(*s*, *n*) returns a string of the first *n* characters of the string *s*.

string_take(*s*, -n) returns a string of the last *n* characters of *s*.

Arguments `string_take` requires two arguments. The first argument *s* must be a string. The second argument *spec* must be a sequence specification.

Examples

```
(%i1) string_take("dog-goat-pig-zebra",[5,12]);
(%o1) goat-pig
```

17.4 Function: with_output_to_string

mext package: lists_aex

Description Evaluates *expr_1*,*expr_2*,*expr_3*,... and writes any output generated to a string, which is returned.

Examples

```
(%i1) sreverse(with_output_to_string(for i:5 thru 10 do print("i! for i=",i,i!)));
(%o1)
 0088263 01 =i rof !i
 088263 9 =i rof !i
 02304 8 =i rof !i
 0405 7 =i rof !i
 027 6 =i rof !i
 021 5 =i rof !i
```

See also `with_stdout`.

18 Miscellaneous Functions

- `examples`
- `examples_add`

18.1 Function: examples

examples(*item*)
mext package: defmfun1

Calling

> **examples**(*item*) Print examples for the topic *item*. Note these examples are different from those extracted from the maxima manual with the command `example`.

Arguments `examples` requires one argument *item*, which must be a string or a symbol.

18.2 Function: examples_add

examples_add(*item*, *text*, *protected-var-list*, *code*)
mext package: defmfun1

Calling

> **examples_add**(*item*, *text*, *protected-var-list*, *code*) Add an example for item *item*. *text* will be printed before the example is displayed. *protected-var-list* is string giving a list of variables such as "[x,y]" that appear in the example code. The example code will be wrapped in a block that makes *protected-var-list* local. *code* may be a string or list of strings that is/are the example code.

Arguments `examples_add` requires four arguments. The first argument *item* must be a string or a symbol. The second argument *text* must be a string. The third argument *protected-var-list* must be a string. The fourth argument *code* must be a string or a list of strings.

Examples
Add an example for the function 'last'.

```
(%i1) examples_add("last", "Return the last item in a  list.", "[a,b,c,d]", "last([a,b,c,d])") ;
(%o1) done
```

19 Miscellaneous utilities

- `compile_lambda_verbose`

19.1 Variable: compile_lambda_verbose

mext package: defmfun1

default value `false`.
Description If this is true, then print translated code when automatically compiling lambda functions passed as arguments. This is done in the macro option-compile-lambda.

20 Options

Options to a function in the aex-maxima distribution are passed as follows:
funcname(x,y, [optname -¿ optval, optname2 -¿ optval2]) or funcname(x,y, optname -¿ optval, optname2 -¿ optval2)
The standard options described in this section are some options that are supported by many functions in the aex-maxima distribution.

- `adj`
- `compile`
- `foptions`
- `ot`

20.1 Option: adj

mext package: maxdoc

Description This option takes values of `true` or `false. If true`, then the output aex expression is adjustable, that is, the underlying array can be extended in size. If `false`, then the output aex expression is not adjustable. The non-adjustable array may have some advantanges in efficiency, but I have not observed them, and this may be lisp-implementation dependent.

20.2 Option: compile

mext package: maxdoc

Description If this option is true, then lambda functions passed as arguments to a function will be automatically translated or compiled. If it is false they will used as interpreted maxima code. Compiling lambda functions usually greatly deceases the execution time of the function if the lambda function is called many times.

20.3 Function: foptions

foptions(*name*)
mext package: defmfun1

Description Return a list of allowed options to `defmfun1` function *name*. I would prefer to call this `options`, but that name is taken by an unused, undocumented function.

Arguments `foptions` requires one argument *name*, which must be a string or a symbol.

20.4 Option: ot

mext package: maxdoc

Description With a value `ar` this option causes the function to return an array-representation expression. With a value `ml` a standard lisp list representation is returned. The array-representation is not a maxima array, but rather a more-or-less arbitrary maxima expression that is stored internally as an array. For certain operations, such as random access to elements of the expression, an array representation is faster than the standard list representation. One disadvantange of the array representations is that creating an array is relatively slow. For instance, execution time may be large if a function returns an expression with many small subexpressions that are in the array-representation. The majority of the maxima system does not understand array-representation, so conversion back to list-representation at may be necessary.