1 logic

1.1 Introduction to logic

This is a draft version of logic algebra package for Maxima. It is being developed by Alexey Beshenov (al@beshenov.ru). All source code is available uder the terms of GNU GPL 2.1.

List of recognized operators:

Operator	Type	Binding power	Description	Properties
`not`	Prefix	`70`	Logical NOT (negation)	
`and`	N-ary	`65`	Logical AND (conjunction)	Commutative
`nand`	N-ary	`62`	Sheffer stroke (alternative denial, NAND)	Commutative
`nor`	N-ary	`61`	Webb-operation or Peirce arrow (Quine's dagger, NOR)	Commutative
`or`	N-ary	`60`	Logical OR (disjunction)	Commutative
`implies`	Infix	`59`	Implication	
`eq`	N-ary	`58`	Equivalence	Commutative
`xor`	N-ary	`58`	Sum modulo 2 (exclusive or)	Commutative

1.2 TEX output

logic.mac assigns the following TEX output:

```
not       \neg
and       \wedge
nand      \mid
nor       \downarrow
or        \vee
implies   \rightarrow
eq        \sim
xor       \oplus
```

Examples:

```
(%i1) load ("logic.mac")$
(%i2) tex (a implies b)$
$$a \rightarrow b$$
(%i3) tex ((a nor b) nand c)$
$$\left(a \downarrow b\right) \mid c$$
```

```
(%i4) tex (zhegalkin_form (a or b or c))$
$$a \wedge b \wedge c \oplus a \wedge b \oplus a \wedge c \oplus b
  \wedge c \oplus a \oplus b \oplus c$$
(%i5) tex (boolean_form (a implies b implies c));
$$ \neg \left( \neg a \vee b\right) \vee c$$
(%i6) tex (a eq b eq c);
$$a \sim b \sim c$$
```

$$a \to b$$

$$(a \downarrow b) \mid c$$

$$a \wedge b \wedge c \oplus a \wedge b \oplus a \wedge c \oplus b \wedge c \oplus a \oplus b \oplus c$$

$$\neg \left(\neg a \vee b\right) \vee c$$

$$a \sim b \sim c$$

1.3 Definitions for logic

`logic_simp` (*expr*) [Function]

Returns a simplified version of logical expression *expr*.

Examples:

```
(%i1) load ("logic.mac")$
(%i2) logic_simp (a or (b or false or (a or b)));
(%o2)                                a or b
(%i3) logic_simp (b eq a eq false eq true);
(%o3)                           eq a eq b false
(%i4) logic_simp ((a xor true) xor b xor true);
(%o4)                                a xor b
```

The function applies only basic simplification rules without introducing new functions.

N.B. It should be merged somehow with the basic Maxima simplifier.

`characteristic_vector` (*expr*, *var_1*, ..., *var_n*) [Function]

Returns a list of size 2^n with all possible values of *expr*.

For example, `characteristic_vector (f(x,y,z), x, y, z)` is equivalent to list

```
[
  f (false, false, false),
  f (false, false,  true),
  f (false,  true, false),
  f (false,  true,  true),
  f ( true, false, false),
  f ( true, false,  true),
  f ( true,  true, false),
  f ( true,  true,  true)
]
```

If *var_1*, ..., *var_n* is omitted, it is assumed that

`[var_1, ..., var_n] = sort(listofvars(expr))`

Examples:

```
(%i1) load ("logic.mac")$
(%i2) characteristic_vector (true);
(%o2)                              [true]
(%i3) characteristic_vector (a xor b);
(%o3)                     [false, true, true, false]
(%i4) characteristic_vector (a implies b);
(%o4)                      [true, true, false, true]
(%i5) characteristic_vector (a implies b, a, b);
(%o5)                      [true, true, false, true]
(%i6) characteristic_vector (a implies b, b, a);
(%o6)                      [true, false, true, true]
```

`zhegalkin_form` (*expr*) [Function]

Returns the representation of *expr* in Zhegalkin basis `{xor, and, true}`.

Examples:

```
(%i1) load ("logic.mac")$
(%i2) zhegalkin_form (a or b or c);
(%o2) (a and b and c) xor (a and b) xor (a and c)
                               xor (b and c) xor a xor b xor c
(%i3) zhegalkin_form ((a implies b) or c);
(%o3) (a and b and c) xor (a and b) xor (a and c) xor a
                                                    xor true
```

`logic_equiv` (*expr_1*, *expr_2*) [Function]

Returns `true` if *expr_1* is equivalent to *expr_2* and `false` otherwise.

Examples:

```
(%i1) load ("logic.mac")$
(%i2) e : ((a or b) xor c) and d$
(%i3) zhegalkin_form (e);
(%o3) (a and b and d) xor (a and d) xor (b and d)
                                              xor (c and d)
(%i4) logic_equiv (%i2, %o3);
(%o4)                              true
(%i5) is (characteristic_vector(%i2) = characteristic_vector(%o3));
(%o5)                              true
(%i6) logic_equiv (x and y eq x, x implies y);
(%o6)                              true
```

`dual_function` (*expr*) [Function]

`dual_function (f (x_1, ..., x_n)) := not f (not x_1, ..., not x_n).`

Example:

```
(%i1) load ("logic.mac")$
(%i2) dual_function (x or y);
(%o2)                  not ((not x) or (not y))
```

```
(%i3) demorgan (%);
(%o3)                                x and y
```

self_dual (*expr*) [Function]

Returns `true` if *expr* is equivalent to `dual_function (expr)` and `false` otherwise.

Examples:

```
(%i1) load ("logic.mac")$
(%i2) self_dual (a);
(%o2)                                true
(%i3) self_dual (not a);
(%o3)                                true
(%i4) self_dual (a eq b);
(%o4)                                false
```

closed_under_f (*expr*) [Function]

`closed_under_f (f (x_1, ..., x_n)` returns `true` if `f (false, ..., false) = false` and `false` otherwise.

Examples:

```
(%i1) load ("logic.mac")$
(%i2) closed_under_f (x and y);
(%o2)                                 true
(%i3) closed_under_f (x or y);
(%o3)                                 true
```

closed_under_t (*expr*) [Function]

`closed_under_t (f (x_1, ..., x_n)` returns `true` if `f (true, ..., true) = true` and `false` otherwise.

Examples:

```
(%i1) load ("logic.mac")$
(%i2) closed_under_t (x and y);
(%o2)                                 true
(%i3) closed_under_t (x or y);
(%o3)                                 true
```

monotonic (*expr*) [Function]

Returns `true` if characteristic vector of *expr* is monotonic, i.e.

```
charvec : characteristic_vector(expr)
charvec[i] <= charvec[i+1],   i = 1, ..., n-1
```

where `a<=b := (a=b or (a=false and b=true))`.

Examples:

```
(%i1) load ("logic.mac")$
(%i2) monotonic (a or b);
(%o2)                                 true
(%i3) monotonic (a and b);
(%o3)                                 true
```

```
(%i4) monotonic (a implies b);
(%o4)                                false
(%i5) monotonic (a xor b);
(%o5)                                false
(%i6) characteristic_vector (a or b);
(%o6)                     [false, true, true, true]
(%i7) characteristic_vector (a and b);
(%o7)                    [false, false, false, true]
(%i8) characteristic_vector (a implies b);
(%o8)                     [true, true, false, true]
(%i9) characteristic_vector (a xor b);
(%o9)                    [false, true, true, false]
```

`linear` (*expr*) [Function]

Returns `true` if `zhegalkin_form`(*expr*) is linear and `false` otherwise.

Examples:

```
(%i1) load ("logic.mac")$
(%i2) linear (a or b);
(%o2)                                false
(%i3) linear (a eq b);
(%o3)                                true
(%i4) zhegalkin_form (a or b);
(%o4)              (a and b) xor a xor b
(%i5) zhegalkin_form (a eq b);
(%o5)                  a xor b xor true
```

Linear functions are also known as counting or alternating functions.

`functionally_complete` (*expr_1*, ..., *expr_n*) [Function]

Returns `true` if *expr_1*, ..., *expr_n* is a functionally complete system and `false` otherwise. The constants are essential (see the example below).

Examples:

```
(%i1) load ("logic.mac")$
(%i2) functionally_complete (x and y, x xor y);
(%o2)                                false
(%i3) functionally_complete (x and y, x xor y, true);
(%o3)                                true
(%i4) functionally_complete (x and y, x or y, not x);
(%o4)                                true
```

`logic_basis` (*expr_1*, ..., *expr_n*) [Function]

Returns `true` if *expr_1*, ..., *expr_n* is a functionally complete system without redundant elements and `false` otherwise.

Examples:

```
(%i1) load ("logic.mac")$
(%i2) logic_basis (x and y, x or y);
(%o2)                         false
```

```
(%i3) logic_basis (x and y, x or y, not x);
(%o3)                         false
(%i4) logic_basis (x and y, not x);
(%o4)                         true
(%i5) logic_basis (x or y, not x);
(%o5)                         true
(%i8) logic_basis (x and y, x xor y, true);
(%o8)                         true
```

All possible bases:

```
(%i1) load ("logic.mac")$
(%i2) logic_functions : { not x, x nand y, x nor y,
                          x implies y, x and y, x or y,
                          x eq y, x xor y, true, false }$
(%i3) subset (powerset(logic_functions),
              lambda ([s], apply ('logic_basis, listify(s))));
(%o3) {{false, x eq y, x and y}, {false, x eq y, x or y},
{false, x implies y}, {true, x xor y, x and y},
{true, x xor y, x or y}, {not x, x implies y},
{not x, x and y}, {not x, x or y},
{x eq y, x xor y, x and y}, {x eq y, x xor y, x or y},
{x implies y, x xor y}, {x nand y}, {x nor y}}
```

`logic_diff` (*f*, *x*) [Function]

Returns the logic derivative df/dx of f wrt x.

```
logic_diff (f (x_1, ..., x_k, ..., x_n),  x_k) :=
    f (x_1, ..., true, ..., x_n)  xor
        f (x_1, ..., false, ..., x_n)
```

Examples:

```
(%i1) load ("logic.mac")$
(%i2) logic_diff (a or b or c, a);
(%o2)          (b and c) xor b xor c xor true
(%i3) logic_diff (a and b and c, a);
(%o3)                       b and c
(%i4) logic_diff (a or (not a), a);
(%o4)                         false
```

`boolean_form` (*expr*) [Function]

Returns the representation of *expr* in Boolean basis `{and, or, not}`.

Examples:

```
(%i1) load ("logic.mac")$
(%i2) boolean_form (a implies b implies c);
(%o2)              (not ((not a) or b)) or c
(%i3) demorgan (%);
(%o3)                ((not b) and a) or c
(%i4) logic_equiv (boolean_form (a implies b implies c),
                   zhegalkin_form (a implies b implies c));
(%o4)                         true
```

demorgan (*expr*) [Function]

Applies De Morgan's rules to *expr*:

```
not (x_1 and ... and x_n) => (not x_1  or ...  or not x_n)
not (x_1  or ...  or x_n) => (not x_1 and ... and not x_n)
```

Example:

```
(%i1) load ("logic.mac")$
(%i2) demorgan (boolean_form (a nor b nor c));
(%o2)          (not a) and (not b) and (not c)
```

pdnf (*expr*) [Function]

Returns the perfect disjunctive normal form of *expr*.

Example:

```
(%i1) load ("logic.mac")$
(%i2) pdnf (x implies y);
(%o2) (x and y) or ((not x) and y) or ((not x) and (not y))
```

pcnf (*expr*) [Function]

Returns the perfect conjunctive normal form of *expr*.

Example:

```
(%i1) load ("logic.mac")$
(%i2) pcnf (x implies y);
(%o2)                    (not x) or y
```

Appendix A Function and variable index