Interpreter of simplified prolog

Author: Tomasz Kępa <tk359746@students.mimuw.edu.pl>

Constructs of prolog that I want to implement

1. Lists (with syntactic sugar)
2. Strings as lists of characters
3. Open data structures
4. Unification (Robinson's algorithm)
5. SLD resolution
6. Negation and SLDNF resolution
7. Integer arithmetics
8. If - then - else construction
9. Queries (?-)
10. `write` predicate
11. File loading ability: [module] (read file with prolog code)

Important parts of prolog than will be ignored

1. Definition of new operators
2. Grammar rules
3. Concurrency

Code snippets

```
% list(X) iff X is a list
list([]).
list([_Head | Tail]) :- list(Tail).


% head(E, L) iff E is the first element of the list L
head(E, [E | _Tail]).


% reverse(L, ?R) iff R is list L with reverse order
reverse(L, R) :- reverse(L, [], R).

reverse([], R, R).
reverse([Head | Tail], A, R) :- reverse(Tail, [Head | A], R).


% qsort(L, S) iff S is sorted list L
```

```
qsort(L, S) :- qsort(L, [], S).

qsort([], A, A).
qsort([E | L], A, S) :- partition(L, E, LL, LP),
                        qsort(LP, A, PS),
                        qsort(LL, [E | PS], S).



partition([], _, [], []).

partition([H | T], E, [H | LL], LP) :- H =< E,
                                       partition(T, E, LL, LP).
partition([H | T], E, LL, [H | LP]) :- H > E,
                                       partition(T, E, LL, LP).


% sum(L, S) iff S is a sum of elements of list L
sum(L, S) :- sum(L, 0, S).

sum([], S, S).
sum([X | Tail], A, S) :- AT is A + X,
                         sum(Tail, AT, S).


% even(X) iff X is an even number
even(s(X)) :- \+ even(X).


% odd(X) iff X is an odd number
odd(s(X)) :- \+ odd(X).


/* tree representation:
 * nil - empty node
 * tree(Left, X, Right) - non-empty node
 */


% depth(T, D) iff D is a depth of a tree T
depth(nil, 0).
depth(tree(LT, _, RT), D) :- depth(LT, LD),
                             depth(RT, RD),
                             max(LD, RD, MD),
                             D is MD + 1.
```

```
max(X, Y, Z) :- ( X =< Y
                  -> Z = Y
                  ;  Z = X ).


% print(L) - prints L to stdout
print([]) :- write('the list is empty'), nl.
print([E]) :- write(E),
              write('.'),
              nl.
print([H1, H2 | Tail]) :- write(H1),
                          write(','),
                          print( [H2 | Tail]).
```