Logic Programming, 16-17

Inês Dutra
DCC-FCUP
ines@dcc.fc.up.pt (room: 1.31)

December 6, 2016

Using repeat

```
% Version with repeat
  read_stdin :-
     repeat,
     read(X),
     write(X),
     nl,
     X=end_of_file, !.
```

```
% Recursive version
  read_stdin :-
     read(X),
     read_all(X).

  read_all(end_of_file) :- !.
  read_all(X) :-
     write(X), nl,
     read(Y),
     read_all(Y).
```

Handling errors

```
error_handling :-
    catch(do_something(X,Y),
          Error,
          write(error(do_something(X,Y),Error))).

do_something(X,Y) :-
    var(X), throw("Undefined variable").
do _something(X,Y) :- ...
```

Using global variables

- `assert, retract`
- `record, recorded`
- `set_value, get_value`

Interacting with the OS

- `system`
- `cd`
- `getcwd`

Using the alarm

```
loop :- loop.

:-   catch((alarm(10, throw(ball), _),loop),
        ball,
        format('Quota exhausted.~n',[])).
```

Profiling programs

```
list_profile :-
    % get number of calls for each profiled procedure
    setof(D-[M:P|D1],(current_module(M),
                      profile_data(M:P,calls,D),
                      profile_data(M:P,retries,D1)),LP),
    % output so that the most often called
    % predicates will come last:
    write_profile_data(LP).

list_profile(Module) :-
    % get number of calls for each profiled procedure
    setof(D-[Module:P|D1],(profile_data(Module:P,calls,D),
                           profile_data(Module:P,retries,D1)),LP),
    % output so that the most often called
    % predicates will come last:
    write_profile_data(LP).

write_profile_data([]).
write_profile_data([D-[M:P|R]|SLP]) :-
    % swap the two calls if you want the most often
    %  called predicates first.
    format('~a:~w: ~32+~t~d~12+~t~d~12+~n', [M,P,D,R]),
    write_profile_data(SLP).
```

More Profiling

```
yap_flag(call_counting,on), [-user].
   l :- l. end_of_file.
yap_flag(call_counting,off).

catch((call_count(10000,_,_),l),
      call_counter,format("limit_exceeded.~n",[])).
```

Statistics of Execution

```
?- statistics.
memory (total)          4784124 bytes
   program space        3055616 bytes:   1392224 in use,      1663392 free
                                                               2228132  max
   stack space          1531904 bytes:       464 in use,      1531440 free
     global stack:                           96 in use,       616684  max
      local stack:                          368 in use,       546208  max
   trail stack           196604 bytes:         8 in use,       196596 free

      0.010 sec. for 5 code, 2 stack, and 1 trail space overflows
      0.130 sec. for 3 garbage collections which collected 421000 bytes
      0.000 sec. for 0 atom garbage collections which collected 0 bytes
      0.880 sec. runtime
      1.020 sec. cputime
      25.055 sec. elapsed time
```

Programming with threads

```
% This program may not run in some versions of yap
% to run need to invoke both create_workers/2 and work/2
% create_workers(+Id, +N)
%
% Create a pool with given Id and number of workers.

create_workers(Id, N) :-
message_queue_create(Id),
forall(between(1, N, _),
       thread_create(do_work(Id), _, [])).

do_work(Id) :-
repeat,
  thread_get_message(Id, Goal),
  (   catch(Goal, E, print_message(error, E))
  ->  true
  ;   print_message(error, goal_failed(Goal, worker(Id)))
  ),
fail.

% work(+Id, +Goal)
%
% Post work to be done by the pool

work(Id, Goal) :-
    thread_send_message(Id, Goal).
```

Interfacing Prolog with C

```
// my_process.c
#include "YapInterface.h"

static int my_process_id(void)
{
     YAP_Term pid = YAP_MkIntTerm(getpid());
     YAP_Term out = YAP_ARG1;
     return(YAP_Unify(out,pid));
}

void init_my_predicates()
{
     YAP_UserCPredicate("my_process_id",my_process_id,1);
}

?- load_foreign_files(['my_process'],[],init_my_predicates).
```