Rheinische Friedrich-Wilhelms-Universität Bonn
Institut für Informatik III
Prof. Dr. A. B. Cremers

Advanced Logic Programming
Summer Semester 2010
Dr. Günter Kniesel

Assignment 10

Due: Sunday, 04.07.2010, 23:59:59 via SVN

For help, contact alp-staff@lists.iai.uni-bonn.de (staff only) or alp-course@lists.iai.uni-bonn.de (staff and participants).

Please start working on the exercises early enough so that you can contact us in time in case of problems. Don't expect us to be available during weekend!

Task 1. *Meta-interpreter* (3 Points)

For the purpose of this task use the following names for the meta-interpreter introduced in the lecture and the previous assignment.

- Basic Prolog Meta-Interpreter `base_solve(`**`Goal`**`)` - slides(5-39 to 5-46)
- Clause Tree Meta-Interpreter `tree_solve(`**`Goal`**`,`**`Trail`**`,`**`Tree`**`)`- slide(5-49)
- Loop Checking Meta-Interpreter `nonloop_solve(`**`Goal`**`,`**`Path`**`)`- slide(5-48)
- Iterative Deepening Meta-Interpreter `depth_solve(`**`Goal`**`,`**`Depth`**`)`- slide(5-52)
- Aspectual Meta-Interpreter `aspect_solve(`**`Goal`**`,`**`ToIntercept`**`,`**`ToCall`**`)`- Assignment 9 Task 3.

These Meta-interpreters can be used to interprete any prolog program. So they should also be able to interprete each other. For the following combinations, describe what happens and what you expect as an result.

a) `?- base_solve(base_solve(base_solve(G))).`

b) `?- aspect_solve(tree_solve(a(1),[],Tree),clause(G,Body),nonloop_solve(Body,Path)).`

c) `?- depth_solve(base_solve(tree_solve(a(1),[],Tree)),7).`

Hint: The implementations shown on the slides only illustrate principles but do not cover all special cases (for instance, some cannot handle the cut, "or" or built-ins properly). When answering the above question simply ignore these details and assume that each meta-interpreter is fully implemented.

Task 2. *Meta-interpreters (II)* (4 Points)

Given the following program for a basic meta-interpreter, a clause tree meta-interpreter and a fact:

```
base_solve(true):-!.
base_solve((G,R)) :- !,
      base_solve(G),
      base_solve(R).
base_solve(G) :-
      predicate_property(G,built_in),!,
      call(G).     % let Prolog do it
base_solve(G) :-
      clause(G, Body),
      base_solve(Body).

tree_solve(true,_,true):- ! .
tree_solve((G,R),Trail,(TG,TR)) :- !,
   tree_solve(G,Trail,TG),
   tree_solve(R,Trail,TR).
tree_solve(G,_,prolog(G)) :-
   predicate_property(G,built_in), !,
   call(G).
tree_solve(G,Trail,tree(G,T)) :-
   not( loop(G,Trail) ),
   clause(G, Body),
   tree_solve(Body,[G|Trail],T).

% predicate we want to solve
a(1).
```

Run the following queries and for each query list the result that you got. Then explain their different behavior.

a. `?- base_solve(a(1)).`
b. `?- tree_solve(a(1),[],Tree).`
c. `?- base_solve(tree_solve(a(1),[],Tree)).`

1. Explain the behaviour of a) versus the behaviour of b).
2. Explain the behaviour of b) versus the behaviour of c).
3. Explain the behaviour of a) versus the behaviour of c).

Task 3. *Misc. questions* (6 Points)

As a preparation for the oral exam, answer the following questions in your own words. Simply copying slides is not accepted.

a) Describe the difference between a predicate as data and as a program.
b) Write a short predicate `reverse_args(+Predicate,-NewPredicate)` which reverses the argument list of a given predicate. Explain how your solution works, including explanations of the basic term manipulation predicates that you used.
c) What is the idea behind the module concept of swi-prolog?
d) Explain the difference between `use_module/1` and `consult/1`.
e) Explain the difference between a red and green "cut".
f) Describe failure driven loops. Give a short example for your explanation.