Fundamentals of Artificial Intelligence and Knowledge Representation – Module 4

Prof. Federico Chesani – 16th of June, 2020

Available time: 25 min. + 10 min. for submitting the essay

Notice: when submitting your essay, please nema the file in the following way:

FamilyNameFirstName.date.pdf/.docx

For example, if you are John Smith, the file will be named:

SmithJohn.20200616.pdf

Exercise

The candidate is invited to present the Prolog "vanilla" meta-interpreter, and to shortly comment (in natural language) the meaning of the clauses.

After that, the candidate is invited to write a meta-interpreter **`solve(Goal, ListOfSubGoals)`** that is evaluated to true if **`Goal`** can be proved; moreover, in the parameter **`ListOfSubGoals`** the meta interpreter will return the list of the subgoals used to prove the **`Goal`**.

For example, given the program:

```
p(X) :- q(X), r(X).
p(X) :- s(X).
q(X) :- t(X).
r(1).
r(2).
r(3).
t(1).
t(2).
s(12).
```

and the query **`?- solve(p(X), Result)`**, the following outcomes are expected:

```
?- solve(p(X), Result).
X = 1,
Result = [p(1), q(1), t(1), r(1)] ;
X = 2,
Result = [p(2), q(2), t(2), r(2)] ;
X = 12,
Result = [p(12), s(12)].
```

Fundamentals of Artificial Intelligence and Knowledge Representation – Module 4

Prof. Federico Chesani – 16th of June, 2020

Available time: 25 min. + 10 min. for submitting the essay

Solution

The candidate is invited to present the Prolog "vanilla" meta-interpreter, and to shortly comment (in natural language) the meaning of the clauses.

It is expected that the candidate reports the "vanilla" meta-interpreter (three clauses), with three short comments (no more than three lines) for each clause. See the slides for the details.

After that, the candidate is invited to write a meta-interpreter ***solve(Goal, ListOfSubGoals)*** *that is evaluated to true if* ***Goal*** *can be proved; moreover, in the parameter* ***ListOfSubGoals*** *the meta interpreter will return the list of the subgoals used to prove the* ***Goal****.*

```
solve(true, []) :- !.
solve( (A,B), Result) :-
    !,
    solve(A, ListA),
    solve(B, ListB),
    append(ListA, ListB, Result).
solve(A, [A|Tail]) :-
    clause(A,B),
    solve(B, Tail).
```