# Logic Programming Question

Qualifier Examination

July 27, 2004

## Question 1 [40 pts]

In class we studied the notion of meta-interpreter and have seen how it can be used to modify the way Prolog programs are executed. For example, the simplest meta-interpreter is:

```
solve(true) :- !.
solve((G1,G2)) :- !, solve(G1), solve(G2).
solve(G) :- clause(G,Body), solve(Body).
```

**Point A [15 pts]**

Provide a simple meta-interpreter that executes Prolog program by always selecting the *rightmost* atom in the goal.

**Point B [25 pts]**

Assume that we modify the syntax of programs by attaching to each atom a *confidence level*. The syntax used is

$$Atom : Confidence$$

E.g., the element $p : 0.5$ indicates that we have 0.5 confidence about $p$. All the facts in the program are expected to be annotated with confidence levels. For example, we can have a program like

```
p:0.2 .
q:0.3 .
r :- p, q.
s :- r.
```

Given a rule:

$$Head : -B_1, \ldots, B_k$$

the confidence of the $Head$ will be obtained by multiplying together the confidence levels of the elements in the body. For example, if $p : 0.2$ and $q : 0.3$, and we have the rule

$$r : -p, q$$

then the confidence of $r$ is $0.2 \times 0.3 = 0.06$.

Write a meta-interpreter $solve(Goal, Confidence)$ which takes as input a $Goal$ and determines its confidence level $Confidence$.

## Question 2 [25 pts]

Consider the following program $P$:

```
p :- q, r.
p :- r, s.
q :- q.
q :- t.
s.
r :- s.
```

Answer the following questions:

1. Show the result of the application of the $T_P$ operator zero, one, two, and three times.
2. Describe the minimal model of the program $P$
3. Add the following rules to the program

   ```
   t :- not q.
   q :- not t.
   ```

   Describe the stable models of the new program.

# Question 3 [35 pts]

Use the constraint logic programming with finite domains that we have seen in class to solve the following problem. We have a factory with 2 machines of type A and 1 machine of type B. We have 3 tasks that have to be completed:

- task 1 requires first machine A for 2 units of time and then machine B for 3 units of time
- task 2 requires first machine B for 3 units of time and then both machines A for 1 unit of time
- task 3 requires one machine A and one machine B together for one unit of time.

Note that once we start a task the task cannot be interrupted (thus we cannot start one task, stop it, and then restart it later on). Write a constraint logic program that computes the schedules for the 3 tasks (at what moment in time each task start). Make sure to clearly separate the part where constraints are set up from the part where the labeling is conducted.