

# CSE-505 Computing with Logic

## Final Exam

Dec 20, 2006

Max: 45 points

Duration: 2 hours 30 minutes

---

### Instructions:

- Please write answers in the blue book provided.
  - Write your name clearly on the cover page of the blue books.
  - There are 6 questions in this exam. Attempt as many questions as you can. The total number of points in this exam is 55. The maximum score is 45.
  - Clearly mark the question number corresponding to each answer.
- 

1. [10 points] Finite state automata can be represented by the following set of facts:

- `trans(a, s1, α, s2)` to represent a transition in automaton *a* from state *s*<sub>1</sub> to state *s*<sub>2</sub> on symbol *α*;
- `init(a, s)` to represent initial state *s* of automaton *a*; and
- `final(a, s)` to represent a final state *s* of automaton *a*.

For example, the following facts represents a finite state automaton:

```
trans(a,1,x,2).
trans(a,1,y,3).
trans(a,2,x,2).
trans(a,2,y,4).
trans(a,3,y,4).
init(a,1).
final(a,4).
```

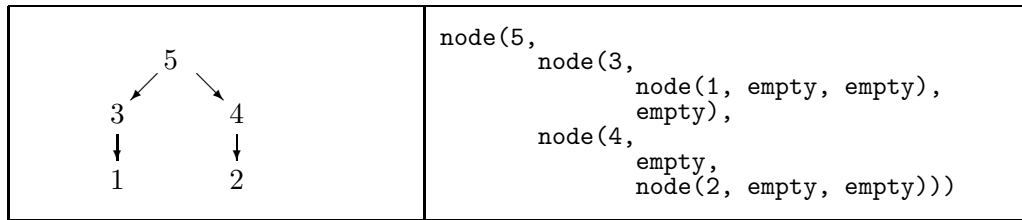
For the purposes of this question, assume that an automaton has a single initial state and a single final state; you may also assume that there are no  $\epsilon$  (silent) transitions.

- (a) [5 points] Write a Prolog predicate `accepts(A, L)`, that, given an automaton *A* and a list *L* of symbols, succeeds if and only if that automaton accepts the sequence of symbols. For instance, with the above example automaton, `accepts(a, [x,x,y])` succeeds but `accepts(a, [x,x])` and `accepts(a, [y,y,x])` fail.
- (b) [5 points] Let *L* be the language defined by a given automaton. We say that an automaton is partially palindromic if there is *some* string  $s \in L$  such that  $s^R$  is also in *L* ( $s^R$  is the reverse of *s*). For instance, the above example automaton is partially palindromic since *yy* is accepted by the automaton.

Write a Prolog predicate `partial(A)` that succeeds if and only if the language defined by automaton *A* is partially palindromic.

You may use tabling for this question; if you use it, make sure you say so.

2. [10 points] Consider binary trees represented by Prolog terms of the form `node(v, t1, t2)` to represent a tree with  $v$  as the value at the root with trees  $t_1$  and  $t_2$  as subtrees, and `empty` to represent an empty tree. For instance, the tree below (left) is represented by the term (right).



- (a) [5 points] A binary tree is said to be a *heap* if the largest value in any subtree occurs at its root. For instance, the tree above is a heap (but will no longer be so if values 2 and 4 are interchanged).  
Write a predicate to determine whether a given tree is a heap or not.  
You may assume, for convenience, that all values in a tree are positive and unique (i.e. two nodes do not have the same value).
- (b) [5 points] Write a Prolog predicate, using the DCG notation, to generate the list of all values in a given tree *in reverse post order*. For instance, reverse post order of the tree above is [5,4,2,3,1]. [Note that reverse post-order is different from pre-order since it reverses the order in which subtrees are visited].  
Full credit will be given only if you find the reverse post order list directly (i.e. without explicitly constructing a post-order list first and then reversing it).
3. [10 points] File systems typically use directories (folders) to organize files in a hierarchy since directories may contain other directories or files. One way to represent a file system is by a set of facts of the form `file(FileName, Directory)` where `FileName` is the name of the file and `Directory` is the name of its enclosing directory. For instance, the file system on the left, below, can be represented by the facts on the right, below.

<pre> / usr/   local/     xemacs     less   share/     tetex   etc/     init.d/       ifup       sshd     fstab </pre>	<pre> file(usr, '/'). file(local, usr). file(xemacs, local). file(less, local). file(share, usr). file(tetex, share). file(etc, '/'). file('init.d', etc). file(ifup, 'init.d'). file(ssh, 'init.d'). file(fstab, etc). </pre>
--	--

For the purposes of this question, assume that the names of all files and directories in a system are unique. Also assume that `'/'` is the unique root.

- (a) [5 points] The *full path name* of a file is a sequence of its containing directories (starting from the root). For example, the full path name of `ssh` in the above example is `['/', 'etc', 'init.d']`.  
Write a Prolog predicate `locate(FileName, FullPathName)` that takes a file name and returns a full path name (for a given a set of file/2 facts).
- (b) [5 points] Write a Prolog predicate `list(Directory, Contents)` that, given a directory name, returns the list of all files contained in that directory. For instance, `list(etc, X)` should return in `X` the value `['init.d', fstab]`.

4. [10 points] Consider the following Prolog program:

```
p(X, Y) :- q(X,X), !, r(X,Y).  
p(X, Y) :- q(X,Z), r(Z,Y), !.  
p(X, Y) :- q(X,Y).
```

```
q(a,b).  
q(b,c).  
q(c,c).
```

```
r(a,b).  
r(c,c).  
r(c,d).
```

- (a) How many answers are there for the query  $p(b,X)$ ? Explain briefly.  
(b) How many answers are there for the query  $p(c,X)$ ? Explain briefly.

[**Note:** For this question it may be useful to consider the SLD trees with Prolog's literal selection strategy.]

5. [10 points] For each of the following programs, state whether the program has zero, one, or more than one stable model. Justify each of your answers.

(a)

```
p ← q, r  
q ← r  
r ← r
```

(b)

```
p ← q, ¬r  
q ← ¬r  
r ← r
```

(c)

```
p ← p, ¬q  
q ← ¬p  
r ← r
```

(d)

```
p ← r, ¬q  
q ← r, ¬p  
r ← ¬s  
s ← s
```

(e)

```
p ← q, ¬r  
q ← ¬r  
r ← p
```

6. [5 points] Write a predicate `count(Goal, N)`, that takes a query `Goal` as an argument and returns (1) an answer of `Goal` by binding its variables, and (2) the length of the derivation for that answer (bound to `N`). Assume that the program over which `Goal` is evaluated is in Prolog's dynamic database.

For instance, let the following clauses be present in the dynamic database:

```
rev([], []).  
rev([X|Xs], Ys) :- rev(Xs, Zs), conc(Zs, [X], Ys).
```

```
conc([], Ys, Ys).  
conc([X|Xs], Ys, [X|Zs]) :- conc(Xs, Ys, Zs).
```

Then `count(rev([a,b,c], Q), N)` should return `Q=[c,b,a]` and `N=10`; and `count(rev([a,b,c,d], Q), N)` should return `Q=[d,c,b,a]` and `N=15`.

---

END OF EXAM