

CSE-505 Computing with Logic

Fall '05

Final Exam

(Take Home)

Due: Dec 21, 2005 (by noon)

Max: 100 points

Instructions:

- Please write answers down on paper.
- Write your name and USB ID number clearly on the first page. Make sure all sheets of paper are securely attached by a stapler pin.
- There are a total of 11 questions on the exam.
- Clearly mark the question number corresponding to each answer.
- For programs, you may use XSB's library predicates *unless expressly prohibited*. You may also use predicates from Bratko's book, but for each predicate, please remember to cite the page number in the text where the predicate is defined.
- You may leave the answer sheets in my mailbox in the faculty area of the department, or slide them under my door. Please make sure that I get the answer sheets by the deadline.

1. [10 points]

- (a) [5 points] Write a Prolog predicate `subterm(T1, T2)` that, given two terms T1 and T2 succeeds if and only if T2 is a subterm of T1.

For instance, `subterm(f(g(a), h(b)), g(a))`, and `subtermf(g(a),h(b)), b)` succeed, whereas `subterm(f(g(a),h(b)), c)` and `subterm(f(g(a), h(b)), h(a))` fail.

You may assume that T1 is a ground term.

For full credit, your predicate should be able to backtrack and return all subterms of a given term. For instance, `subterm(f(g(a),h(b)), X)` should return `X=f(g(a),h(b))` and upon backtracking, return `X=g(a)`, `X=a`, `X=h(b)` and `X=b`.

- (b) [5 points] Write a Prolog predicate `constants(T,L)` that, given a term T, returns the list of all constants in T in the order in which they appear. For instance:

- `constants(f(g(a), g(b)), L)` should return `L=[a,b]` and
- `constants(f(g(a), f(b,h(a))), L)` should return `L=[a,b,a]`.

For full credit your program should run in time linear in the size of the input term.

2. [12 points] A finite state automaton can be represented by the following set of facts:

- `trans(s1, a, s2)` to represent a transition in the automaton from state s_1 to state s_2 on symbol a ;
- `init(s)` to represent initial state s ; and
- `final(s)` to represent a final state s .

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

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

- (a) [5 points] Write a Prolog predicate `accepts(L)`, that, given a list L of symbols, succeeds if and only if the automaton accepts the sequence of symbols.

For instance, with the above example automaton, `accepts([a,a,b])` succeeds but `accepts([b,b,a])` fails.

- (b) [7 points] Write a Prolog predicate `gen(N, L)`, that, given an integer N, generates a list L of symbols of length N such that L is accepted by the given automaton.

For instance, `gen(2,L)` for the above example automaton should return `L=[a,b]` and `L=[b,b]`.

3. [8 points] *Lev* is a simple-minded, one-dimensional robot which works in an infinitely tall building. *Lev* obeys two commands:

- **up**: *Lev* goes to the next higher floor. Since the building is infinitely tall, there is always a next higher floor.
- **down**: *Lev* goes to the next lower floor. If *Lev* is already on the ground floor, it goes crazy and chases you shouting “Illegal! Illegal!!”.

At the beginning, *Lev* is on the ground floor.

Let L be a sequence of commands (represented, of course, as a Prolog list) that you plan to give to *Lev*. Write a Prolog predicate `upto(L, N)` that succeeds if and only if *Lev* will *not* go crazy when executing L . (Note that L may not leave *Lev* back on the ground floor). If `upto` succeeds, then N should be bound to the highest floor *Lev* reached when executing L .

For instance:

- `upto([down])`, `upto([up, down, down, up, up])`, should both fail.
- – `upto([], N)` succeeds with $N = 1$
- `upto([up, down], N)`, `upto([up,down,up,down], N)` `upto([up, up, down], N)` should all succeed with $N = 2$
- `upto([up, up, down, up, down,down], N)` succeeds with $N = 3$.

4. [10 points] Regular expressions can be represented using Prolog terms of the following forms:

Expression	Represents
<code>alpha(a)</code>	individual symbol a
<code>seq(r₁, r₂)</code>	concatenation of r_1 and r_2
<code>union(r₁, r₂)</code>	union of r_1 and r_2
<code>star(r₁)</code>	Kleene closure of r_1

For instance, the expression $(a + b)$ can be represented as `union(alpha(a), alpha(b))`; the expression $(a + b)^*$ can be represented as `star(union(alpha(a), alpha(b)))`; the expression $a(a + b)$ can be represented as `seq(alpha(a), union(alpha(a), alpha(b)))`.

Let r_1 and r_2 be Prolog terms that represent regular expressions. Write a predicate `nonempty_intersect/2` such that `nonempty_intersect(r1, r2)` succeeds if and only if there is some string s in the intersection of the languages defined by r_1 and r_2 . You may use tabling if necessary.

5. [10 points] Consider the following normal logic program:

```
t(0).
t(s(s(X))) :- t(X).
```

```
f(s(0)).
f(s(s(s(X)))) :- f(X).
```

- (a) Using SLDNF resolution, compute the first three answers (upon backtracking) for the query `t(X)`, `not f(X)`.
 (b) Can SLDNF answer the query `not f(X), t(X)`? Explain.

6. [12 points] Consider the following normal logic program:

```
vc(X) :- edge(X,Y), not vc(Y).
edge(1,2).
edge(1,3).
edge(2,3).
edge(2,4).
edge(3,4).
```

Enumerate all stable models of the above program.

7. [6 points] For each of the following programs, state whether or not

- a. The program is stratified or not,
 b. Whether the program has a stable model or not, and
 c. Whether the program has a well-founded model or not.

(a)

```
r ← ¬p, ¬q
q ← ¬p
p ← p
```

(b)

```
r ← ¬p, ¬q
q ← ¬p
p ← p, q
```

(c)

```
r ← ¬p, ¬q
q ← ¬r, ¬p
p ← r
```

8. [12 points] Find the well-founded model of the following general logic program. For full credit, show all the iterations.

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

9. [12 points total] Let $v/1$, $p/2$ and $q/1$ be three predicates defined in a program P .
- (a) [4 points] Extend P to define a new predicate $r/1$ such that $r(X)$ holds whenever

$$\exists Y.p(X, Y) \wedge q(Y)$$

The extended program must be a definite logic program.

For example, if the following is least Herbrand model of P :

```
v(1).    p(1,2).    q(2).
v(2).    p(1,3).    q(4).
v(3).    p(2,3)
v(4).    p(3,4).
```

then $r(1)$ and $r(3)$ are logical consequences of the extended program.

- (b) [8 points] Extend P to define a new predicate $s/1$ such that $s(X)$ holds whenever

$$v(X) \wedge (\forall Y.p(X, Y) \Rightarrow q(Y))$$

The extended program may be a general logic program.

For example, if the following is the least Herbrand model of P :

```
v(1).    p(1,2).    q(2).
v(2).    p(1,3).    q(4).
v(3).    p(2,3)
v(4).    p(3,4).
```

then $s(3)$ and $s(4)$ are logical consequences of the extended program (and $s(1)$ and $s(2)$ are *not* logical consequences).

Do not use any Prolog built-ins!

10. [10 points] A Datalog program is a set of Horn clauses where the alphabet consists of only constants, variables and predicate symbols (i.e. no function symbols).
- (a) [5 points] Show that every Datalog program has a finite least Herbrand model.
- (b) [5 points] Write a Prolog predicate `isDatalog` to determine if a program in a given file is a Datalog program or not. That is, `isDatalog(fileName)` should succeed if and only if the file `fileName.P` contains a program that has no function symbols.

11. [10 points] Let list A represent a sequence; a subsequence B of A is a list where the elements of B occur in the same order in A (but possibly separated by other elements of A). For instance, $[l,i,r,a]$ is a subsequence of $[l,o,g,i,c,p,r,o,g,r,a,m,m,i,n,g]$.

The following program computes the longest common subsequence of two lists:

```
:- table lcs/4.
% lcs/4: first two arguments are the two given lists.
%       the third argument is the length of the longest common subsequence
%       the fourth argument is a longest common subsequence.
lcs([], _, 0, []).
lcs(_, [], 0, []).
lcs([X|Xs], [X|Ys], N, [X|Zs]) :-
    lcs(Xs, Ys, Zs, M),
    N is M + 1.
lcs([X|Xs], [Y|Ys], N, Zs) :-
    X \= Y,
    lcs(Xs, [Y|Ys], M1, Zs1),
    lcs([X|Xs], Ys, M2, Zs2),
    longer(M1, M2, Zs1, Zs2, M, Zs).

longer(M1, M2, L1, L2, M, L) :-
    (M1 > M2
    -> L = L1, M = M1
    ;   L = L2, M = M2
    ).
```

- Consider evaluating the query `lcs(L1, L2, N, L)` for two given lists $L1$ and $L2$ using tabled evaluation. How many different calls will be placed in the call table during query evaluation? Give your answer as a function of the lengths of $L1$ and $L2$.
- For each call in the call table, how many different answers will be computed?
- How long does it take to compute a single answer, assuming that all other answers already exist in the tables?
- How long does it take to evaluate a query `lcs(L1, L2, N, L)` for two given lists $L1$ and $L2$? Give your answer in terms of worst case times, as a function of the lengths of $L1$ and $L2$.

END OF EXAM