Family name _____

Given name(s) _____

Student number _____

York University

Department of Computer Science & Engineering

CSE 3401 - Functional & Logic Programming

Solutions to Test 1

June 23, 2010

Instructions:

1. The test time is 120 minutes.
2. This is a closed book examination. No examination aids are permitted.
3. If a question is ambiguous or unclear then please write your assumptions and proceed to answer the question.
4. Return all examinations papers before leaving the exam room.
5. If needed, use extra papers at the end of the booklet. Do not use any other paper.
6. Keep all pages stapled.
7. Keep track of time. Your time is limited.

Good Luck!

Question	Max	Mark
1	10	
2	15	
3	15	
4	7	
5	8	
6	5	
7	10	
8	5	
9	5	
10	20	
Total	100	

Cheat Sheet
(Keep stapled to the your exam booklet)

Conversion to CNF:

1. Remove implications and equivalences
2. Move negations inward
3. Rename variables
4. Convert to Prenex normal form
5. Skolemize
6. Distribute or over and
7. Remove universal quantifiers

A few built-in predicates in Prolog (You don't need to use all of them):

close(+Stream)
current_input(-Stream)
current_output(-Stream)
number(+Term)
nonvar(+Term)
open(+SrcDest, +Mode, -Stream)
set_input(+Stream)
set_output(+Stream)
var(+Term)
?Term =.. ?List

Question 1. (10 marks) Convert the following formula to logic programming clauses. Clearly show ALL steps. Can it be done successfully? Why?

$(\forall x)((\exists y)lt(x,y) \wedge (\exists z)lt(z,x)) \rightarrow (\forall x)(\exists z)p(z,x)$

(Step1) $\neg((\forall x)((\exists y)lt(x,y) \wedge (\exists z)lt(z,x))) \vee (\forall x)(\exists z)p(z,x)$

(Step2) $(\exists x)((\forall y)\neg lt(x,y) \vee (\forall z)\neg lt(z,x)) \vee (\forall x)(\exists z)p(z,x)$

(Step3) $(\exists x)((\forall y)\neg lt(x,y) \vee (\forall z)\neg lt(z,x)) \vee (\forall u)(\exists v)p(v,u)$

(Step4) $(\exists x)(\forall y)(\forall z)(\forall u)(\exists v)(\neg lt(x,y) \vee \neg lt(z,x) \vee p(v,u))$

(Step5) $(\forall y)(\forall z)(\forall u)(\neg lt(g1,y) \vee \neg lt(z,g1) \vee p(g2(u),u))$

(Step6) same - nothing to do

(Step7) $\neg lt(g1,y) \vee \neg lt(z,g1) \vee p(g2(u),u)$

(As a logic programming clause) $p(g2(u),u) :- lt(g1,y), lt(z,g1).$

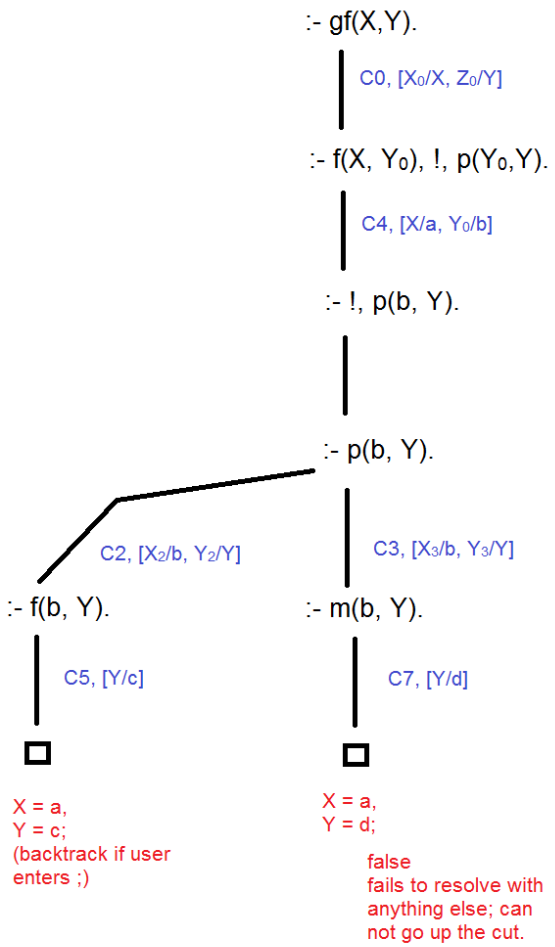
We obtained a Horn clause and the conversion to a logic programming clause is successful.

Question 2. (15 marks) Consider the following program in Prolog:

C0: gf(X, Z) :- f(X,Y), !, p(Y,Z).
C1: gf(X, _) :- w(X).
C2: p(X, Y) :- f(X, Y).
C3: p(X, Y) :- m(X, Y).
C4: f(a, b).
C5: f(b, c).
C6: f(u, v).
C7: m(b, d).
C8: m(v, w).

Draw the complete search tree for this query:
G0: :- gf(X,Y).

Label all branches. Under each leaf node, mention why backtracking occurs. If there are any outputs by Prolog, indicate them too.



Question 3. (15 marks) Write a program in Prolog for converting grams to kilograms (each kilogram is 1000 grams) in a way that the following responses are returned for the shown queries:

?- kilos(1000, K).
K = 1.

?- kilos(2000,2).
true.

?- kilos(G, K).
Error: unknown input!
false.

?- kilos(a,2).
Error: input not a number!
false.

?- kilos(3,3000).
false.

```
kilos(G,_):-    var(G), !,
                write('Error: unknown input!'), nl, fail.

kilos(G,_):-    \+number(G), !,
                write('Error: input not a number!'), nl, fail.

kilos(G, K):-   K is G/1000.
```

(Since the question only asked about converting grams to kilograms, you didn't have to consider that conversion)

Question 4. (7 marks) Are the following unifiable? If yes, give the mgu. If no, give your reason.

(a) due(book(prolog, Author, Year), Borrower) and due(Book, john)

Yes, mgu= [Book/book(prolog, Author, Year), Borrower/john]

(b) triangle(A, p(1,1), p(3,2)) and si(A, B, C)

No, not the same predicates triangle $\neq$si

(c) When does Prolog look for mgu's and why?

When given a query, Prolog does linear refutation and tries to resolve the leftmost subgoal with its knowledge base. In order to resolve, this subgoal must unify with a fact or the head of a rule. If a unifiable case is found, the mgu is used to find the resolvent given the resolution rule in predicate logic:

Given a Horn clause and a goal in the form:

A1:-B1...Bn. and :- D1...Dm.

If e is a unifier of A1 and D1 (i.e. e(A1)=e(D1))
Then the resolvent of the above two clauses is the following goal:

:-e(B1).. e(Bn) e(D2)... e(Dm).

The mgu is also used in back substitution to answer queries.

Question 5. (8 marks) Given a program as follows in Prolog:

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

What are the rules that Prolog follows for building its search tree? Is it true that the query **:- r.** on above program will create an infinite branch in the search tree? Why?

There are three rules:

R1) Linear refutation: always starting from the goal.
R2) Prolog's computation rule: starting from the leftmost subgoal
R3) Starting from the top and going down the knowledge base

Based on the (R1), the search starts with the goal :-r. Then based on (R2), the leftmost subgoal r is selected for matching and is matched with the head of r:-s, q, p. The resolvent is :- s,q, p. Again based on (R2), the leftmost subgoal is matched with s. resulting in the resolvent :-q, p. Another application of (R2), will get us to :-fail, p. which means this branch of the search tree will end due to fail. No other branches are possible to explore on backtracking.

As we can see, by applying R1, R2, R3 in Prolog, p:-p. will never be used in refutation of the query :-r. and therefore an infinite branch will NOT be created in the search tree.

Question 6. (5 marks) What is the answer to the following queries in prolog:

(a) **?- S =.. [a, b, c].**

S= a(b,c).

(b) **[a, b, c] =.. [H|T].**

H='.', T=[a, [b,c]].

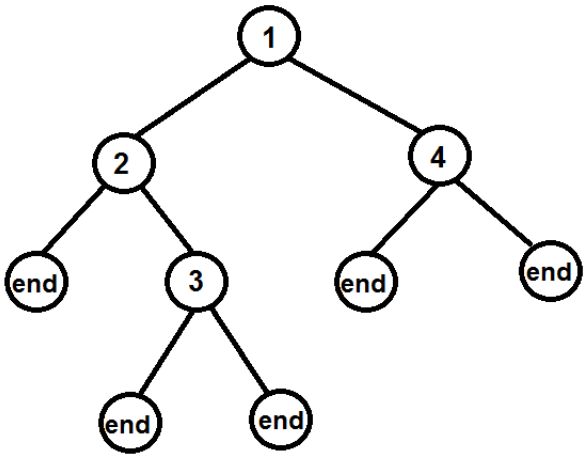
Question 7. (10 marks) (a) Given a tree structure t(Left, Root, Right) what does the following code do?

```
unknown(end, []).
unknown(t(Left, Root, Right), L):-   unknown(Left, L1),
                                     unknown(Right,L2),
                                     append(L1, [Root|L2], L).
```

This code will convert a tree into a list of the nodes. The order is [Left, Root, Right] applied recursively to the subtrees.

(b) What is the answer to the following query? Draw the tree matched with T.
?- T=t(t(end, 2, t(end,3,end)), 1, t(end, 4, end)), unknown(T, L).

It is:
T = t(t(end, 2, t(end, 3, end)), 1, t(end, 4, end)),
L = [2, 3, 1, 4].



Question 8. (5 marks) We have written the following query in Prolog:

:- op(500, yfx, '&&').

How is ***a && b && c*** evaluated, as ***(a && b) && c*** or as ***a && (b && c)***? Explain your answer.

Based on the specifier yfx, we have declare && to be a binary operator, for which the left argument can be a term with precedence class equal or lower to the precedence of the operator and the right argument can be a term with precedence class strictly lower to the precedence of the operator.

For a && (b &&c), the right argument is (b && c) which has the same precedence class as && and this is not permitted based on yfx specifier.

On the other hand for (a &&b) && c the left argument is (a && b) with the same precedence class as && and this is ok based on yfx specifier.

Therefore the operator && is left associative and the answer is (a && b) && c.

Question 9. (5 marks) Given the following code for finding the minimum of two numbers:

min(X, Y, X) :- X<Y, !.
min(X, Y, Y).

(a) Write three different types of queries with **min** that correctly work (based on the definition of minimum). Show the answers to the queries.

:- min(10, 20, 10).
true
:- min(10, 20, Z).
Z= 10.
:- min(20, 10, Z).
Z= 10.

(b) Write a query with **min** for which the answer is intuitively wrong. Why does this happen?
:- min(10, 20, 20).
true.

The query matches with min(X,Y,Y), therefore it can be refuted and the answer returned is true. Here using cut in the first line of code is not effective for this kind of query.

Question 10. (20 marks)The sales of a store have been stored in the file 'store.dat' as a linked list of structures containing serial number, description, and amount. For example for three sales described in the following table, the file contains:
llist(s(1003, item3, 15.99), llist(s(1002, item2, 21.99), llist(s(1001,item1, 10.99), end))).

Serial number	Description	Amount
1001	item1	10.99
1002	item2	21.99
1003	item3	15.99

As you can see, the sales are added in front of the linked list. Using an appropriate design (recursion, accumulators, or difference lists), write a predicate getA(L) in Prolog that reads the linked list from the file and returns the amounts in list L in the correct order. For example, for above linked list stored in the file, we will get:
?- getA(L).
L = [10.99, 21.99, 15.99].

Which design did you use and why do you think it is the appropriate one for the purpose of this question?

Since the sales are added in front of the linked list, reading from the front of the linked list to the end will be in the reverse order. If we use the accumulators, the resulting list will also be reversed which will give us the correct order as a result. This design has a lower complexity than the recursion method.

Using the accumulator design, we can write:

```
getA(L):- open('store.dat', read, SI),
          current_input(CI),
          set_input(SI),
          readAmounts(L),
          close(SI),
          set_input(CI).

readAmounts(L):-
        read(LL),
        getAmounts(LL, [], L).
getAmounts(end, L,L).
getAmounts(llist(s(Serial, Desc, Amount), Rest), A, L):-
        getAmounts(Rest,[Amount|A], L).
```