Final Examination, CMPUT 325

April 18, 2002
Section B2, instructor Martin Müller

Last name: Solutions____________

First name: All_________________

time(minutes(120)).
numberofquestions(4).
pages(10).
totalmarks(100).
marks(question1, 25).
marks(question2, 30).
marks(question3, 20).
marks(question4, 25).

This is a ‘closed book’ exam, you cannot use any notes or books or computers etc.

This exam is printed on the right side pages only, so you have some space on the empty left pages for temporary notes.
Write answers in the *space directly after each question* (preferred), or make it VERY clear what the answer for each question is, if you write it on the left side page. On the exam pages (right pages), cross out everything that you wrote that should not be part of your answer.

This is a long exam. Maybe you cannot finish all of it - but most of the questions have very short answers, so use your time wisely.

All programming examples have short solutions, which may include writing some small helper functions. So before you spend a lot of time on a complicated solution, think about the problem for a little bit longer.

For writing code, the correctness and clarity of your code are both important. You don’t need to write comments or test cases for your code in this exam.

Use only standard functions and predicates that we talked about in class.
Use names starting with an underscore such as _1, _2, _3,... for newly created variables in Prolog.

For true-or-false questions, circle either **true** or **false. Do not make a blind guess if you don’t know the answer. Giving too many wrong answers will reduce your marks.**

Examples:

0.0.0 [✓ true] false Today is the final exam for 325 (circle the right answer)

0.0.1 true false It will snow on April 18, 2003 (you don’t know, so do not circle anything)

Student ID ___________

Question 1 (25 marks) _________
Question 2 (30 marks) _________
Question 3 (20 marks) _________
Question 4 (25 marks) _________

Total (100 marks) _________

1. Lisp (25 marks total)

1.1 Understanding Lisp Code (10x1 mark)

We use pure Lisp, where *car* and *cdr* give an error if called for something that is not a list.

Consider the Lisp functions f and g:

```
(defun f (L)
     (cond
      ( (null L) nil)
      ( (eq (car L) 'a) (cons 'b (f (cdr L))))
      ( t (cons (car L) (f (cdr L)))))
)

(defun g (L1 L2)
      (cons (car L1) (cdr L2))
)
```

What is the result of evaluating the following expressions?

1.1.1 (f '(a b c)) (b b c)

1.1.2 (f '(a a (a))) (b b (a))

1.1.3 (f '(a)) (b)

1.1.4 (f '((a))) ((a))

1.1.5 (f 'a) Argument is invalid -- should be a list: a

1.1.6 (g '(2 5 9) '(3 4 5)) (2 4 5)

1.1.7 (g '((2) 5 9) '(3 (4 5)))((2) (4 5))

1.1.8 (g '(2 5 9) '((3 4 5))) (2)

1.1.9 (g '(2 5 9) '(a (3 4 5)))(2 (3 4 5))

1.2 Writing Lisp Code (9 marks)

We will call a list of exactly three integers such as (5 2 8) a *triple*.

1.2.1 (1 mark) Write a Lisp function **addtriple** as follows:
Given a triple, return the sum of the three numbers in the triple
Example:
(**addtriple** '(1 2 3)) should return 6

```
(defun addtriple (L)
        (+ (car L) (cadr L) (caddr L))
)
```

1.2.2 (2 marks) Write a Lisp function **addlist** as follows:
Given a list of triples, return a list containing the sums of the numbers in each triple.
Use the **addtriple** function in your code.
Example:
(**addlist** '((1 2 3) (0 0 5) (0 0 0) (9 9 9))) should return (6 5 0 27)

```
(defun addlist (L)
        (if (null L)
                nil
        (cons (addtriple (car L)) (addlist (cdr L))))
)
```

1.2.3 (3 marks) Write a Lisp function **rtriple** as follows:
Given a 3-argument function f and a triple,
return the result of applying f to the three numbers in the triple.
Example:
(defun f (x y z) (+ x y z))
(defun g (x y z) (* x z))
(rtriple 'f '(9 9 9)) should return 27
(rtriple 'g '(9 9 9)) should return 81

```
(defun rtriple (f L)
(funcall f (car L) (cadr L) (caddr L))
)
```

1.2.4 (3 marks) Write a Lisp function **rlist** as follows:
Given a 3-argument function f and a list of triples,
return a list with the results of applying f to the numbers in each triple.
Example (using functions f and g from 1.2.3):
(rlist 'f '((1 2 3) (0 0 5) (0 0 0) (9 9 9))) should return (6 5 0 27)
(rlist 'g '((1 2 3) (0 0 5) (0 0 0) (9 9 9))) should return (3 0 0 81)

```
(defun rlist (f L)
        (if (null L)
                nil
        (cons (rtriple f (car L)) (rlist f (cdr L))))
)
```

1.3 Higher-order and Lambda Functions (6 marks)

Given the following definitions of **f** and **s**, determine the result of the expressions below.

```
(defun f (x y) (cons x y))
(defun s (f) (function (lambda (x y) (funcall f y x))))
```

1.3.1 (2 marks)

(mapcar 'f '((a) (b)) '((1) (2)))

(((a) 1) ((b) 2))

1.3.2 (2 marks)

(mapcar (s 'f) '((a) (b)) '((1) (2)))

(((1) a) ((2) b))

1.3.3 (2 marks) This question uses the standard function **append**.

(mapcar (s 'append) '((a) (b)) '((1) (2)))

((1 a) (2 b))

2. Prolog (30 marks total)

2.1 Unification (8 marks)

What is the first result of the following queries in Prolog? Give the overall outcome (circle one of: **yes** if query succeeds, **no** if query fails, **infinite-loop**, or **other-error**) as well as the variable bindings. Use names such as _1, _2, _3,... for newly created variables, if necessary. Assume that no user-defined program has been loaded before executing your queries.

Example: **?- Z=0.**
outcome: yes
bindings: Z=0

2.1.1 (2 marks)

?- p(X) = p(5).

outcome: yes

bindings: X=5

2.1.2 (2 marks)

?- X = 20, f(Y) = f(g(Z)), Z = g(T).

outcome: yes

bindings: `X = 20`
```
Y = g(g(_1))
Z = g(_1)
T = _1
```

2.1.3 (2 marks)

?- X is 5+3, Y=X+X.

outcome: yes

bindings: `X = 8`
```
Y = 8+8
```

2.1.4 (2 marks)

?- X = 5+3, Y is X+Z.

outcome: other-error

bindings: (none)

2.2 Understanding a Prolog Program and Backtracking (8 marks)

Consider the following Prolog program:

```
tag(1).
tag(2).
label(a).
label(b).
label(c).

p([]).
p([T,L|R]) :- tag(T), label(L), p(R).
q([L|R]) :- label(L), p(R).
r(X) :- p(X).
r(X) :- q(X).
```

What is the first result for the following Prolog query, and what are the results of asking for four more answers by pressing semicolon **;** four times?

2.2.1 (2 marks)

?- p(X).

first: `X = []`

next four:

```
X = [1,a] ;
X = [1,a,1,a] ;
X = [1,a,1,a,1,a] ;
X = [1,a,1,a,1,a,1,a]
```

2.2.2 (2 marks)

?- q(X).

first: `X = [a]`

next four:

```
X = [a,1,a] ;
X = [a,1,a,1,a] ;
X = [a,1,a,1,a,1,a] ;
X = [a,1,a,1,a,1,a,1,a]
```

2.2.3 (2 marks)

?- r(X).

first: `X = []`

next four:

```
X = [1,a] ;

X = [1,a,1,a] ;
X = [1,a,1,a,1,a] ;
X = [1,a,1,a,1,a,1,a]
```

2.2.4 (2 marks) Assume that in the program above, the order of the two clauses for r(X) is swapped:

r(X) :- q(X).
r(X) :- p(X).

Now what are the results of the query **?- r(X).**

first: `X = [a]`

next four:

```
X = [a,1,a] ;
X = [a,1,a,1,a] ;
X = [a,1,a,1,a,1,a] ;
```

2.3 Understanding and Changing Prolog Programs (14 marks)

Consider the following Prolog program:

```
member(A,[A|_]).
member(A,[B|L]) :- A \== B, member(A,L).
a(X,L,L) :- member(X,L), !.
a(X,L,[X|L]).
```

2.3.1 (6x1 mark)
Determine the first answer for the following queries, and give the bindings for the variables.
If a query fails, explain why.

2.3.1.1 **?- a(3, [1,2], X).** `X = [3,1,2]`

2.3.1.2 **?- a(3, [3,1,2], X).** `X = [3,1,2]`

2.3.1.3 **?- a(3, [1,2,3,3], X).** `X = [1,2,3,3]`

2.3.1.4 **?- a(3, [1,[3],2], X).** `X = [3,1,[3],2]`

2.3.1.5 **?- a(3, X, X).** `X = [3|_1]`

2.3.1.6 **?- a(3, X, [2|Y]).** `X = [2,3|_1], Y = [3|_1]`

2.3.2 (5 marks)
Write a Prolog predicate **a1** that computes exactly the same first answer as **a** but does *not* use the cut. You must code all the helper functions that you may want to use, except for **member**.

```
not_member(X, []).
not_member(X, [Y|L]) :- not_member(X, L), X \== Y.

a1(X, L, L) :- member(X, L).
a1(X, L, [X|L]) :- not_member(X, L).
```

2.3.3 (1 mark)
What about the efficiency of the two predicates? Circle one of the three given answers.
a) Predicate **a** is more efficient.
a1) Predicate **a1** is more efficient. **a** is more efficient
same) Both are about the same.

2.3.4 (2x1 mark) answer true or false.

2.3.4.1 false Predicate **a** can generate more than one answer with backtracking.
2.3.4.2 true Predicate **a1** can generate more than one answer with backtracking.

3. Constraint Programming (20 marks total)

3.1 Completing a Constraint Program (10 marks)

Professor C.S.Train had to schedule five meetings on the same day, and solved this problem by using Sicstus Prolog and the CLPFD package.
She used the following 5 variables to represent the starting times of meetings:
S1 .. meet student1 (duration: 1 hour)
S2 .. meet student2 (duration: 2 hours)
R .. research group meeting (duration: 2 hours)
L .. lunch with colleagues (duration: 1 hour)
B .. breakfast with visitor (duration: 1 hour)

Here are the constraints:
- all meetings start at full hours (e.g. 8:00, but not 8:30)
- no meetings before 8:00
- all meetings must end at 19:00 at the latest
- order: breakfast first, meet student1 before lunch, student2 after lunch, research meeting last.
- lunch should start either at 11:00, 12:00, or 13:00.

3.1.1 (2x1 mark)
Compute two different solutions to the constraint problem by hand.

Solution 1: S1 =   S2 =   R =   L =   B =
many solutions, e.g. M = [9,13,15,12,8]

Solution 2: S1 =   S2 =   R =   L =   B =
many solutions, e.g. M = [9,13,16,12,8]

3.1.2 (8 marks)
Here is Professor Train's program, with some parts left out. Write the missing parts in the space provided after the program.

```
MISSING-PART-1    % load CLPFD package

todaysmeetings(M) :-
      M = [S1,S2,R,L,B],      % list of all meetings
      MISSING-PART-2,    % define variables and domains
      constrain(M),              % check given constraints
      MISSING-PART-3.    % solve constraint problem (use default settings)

constrain(M) :-
      MISSING-PART-4.    % check given constraints
```

Write the missing parts here:
3.1.2.1 (1 mark) MISSING-PART-1
```
:- use_module(library(clpfd)).
```

3.1.2.2 (2 marks) MISSING-PART-2
```
domain(M,8,17)
```

3.1.2.3 (1 mark) MISSING-PART-3
```
labeling([],M)
```

3.1.2.4 (4 marks) MISSING-PART-4
```
      M = [S1,S2,R,L,B],
B #< S1, S1 #< L, L #< S2, S2 #< R - 1, L #>= 11, L #=< 13.
```

3.2 Questions on Constraint Programming (10x1 mark)

Are the following statements true or false? Circle **true** or **false** only if you know the answer. In the questions, CLP stands for “constraint logic program” or “constraint logic programming”.

3.2.1	true		Prolog’s backtracking mechanism is useful for a CLP solver.
3.2.2		false	CLP is a more general problem-solving method than logic programming.
3.2.3		false	All variables in a constraint problem have the same domain.
3.2.4	true		A variable domain in CLPFD must be a range of consecutive integers.
3.2.5	true		In principle, constraint problems could be solved by using C or Java.
3.2.6	true		Current CLP systems are useful for industrial applications.
3.2.7		false	A problem with two contradictory constraints can still have a solution.
3.2.8		false	CLP is designed to replace all previous forms of logic programming.
3.2.9	true		A CLP can be 1 million times more efficient than Prolog backtracking.
3.2.10			false CLPFD contains one fixed strategy for assigning values to variables.

4 Foundations of Logic Programming (25 marks total)

4.1 Questions on Logic Programming (10 marks)

Are the following statements true or false? Circle **true** or **false** only if you know the answer. In the questions, let P be a program consisting of Horn clauses, let H be its Herbrand universe, and let B be the Herbrand base of P. Let the minus sign ‘-’ stand for logical negation.

4.1.1		false	A Horn clause has at most one negative literal.
4.1.2		false	$p \rightarrow (a \vee b)$ is a Horn clause.
4.1.3	true		$(a \vee b) \rightarrow p$ is a Horn clause.
4.1.4	true		An atom is a literal.
4.1.5	true		A negated atom is a literal.
4.1.6	true		Any subset of B is an interpretation of P.
4.1.7	true		B is a model of P.
4.1.8		false	The empty set is always a model of P.
4.1.9	true		$-\forall x\ p(x)$ is logically equivalent to $\exists x\ -p(x)$.

4.2 Herbrand (15 marks)
Let P be a program consisting of the following set of Horn clauses. In P, x and y are the variables.

p(a,b)
p(b,a)
p(x,y) → q(y,x)
q(x,y) → q(f(x),y)
p(x,y) → p(g(x),y)

4.2.1 (1 mark)
What are the constants in P? a,b

4.2.2 (1 mark)
What are the functions in P? f,g

4.2.3 (1 mark)
What are the predicates in P? p,q

Let H be the Herbrand universe of P and let B be the Herbrand base of P. Consider the following expressions:

1) f(b) in H
2) p(a,g(a)) in B
3) b in H
4) f(g(g(a))) in H
5) p(b) XXXX p needs two arguments
6) q(x,y) XXXX x and y are not constants
7) p(a,q(a,b)) XXXX q is not a function symbol
8) p(f, g(a)) XXXX f needs an argument

4.2.4 (2 marks)Which of the expressions 1)-8) are elements of the Herbrand universe H?

1),3),4)

4.2.5 (2 marks)Which of the expressions 1)-8) are elements of the Herbrand base B?

2)

4.2.6 (2 marks) Give a Herbrand interpretation I with exactly 5 elements. If none exists, why not?

many possible solutions. E.g. {p(a,b), p(a,a),p(b,a),p(b,b),q(a,a)}

4.2.7 (2 marks) Give a Herbrand model M with exactly 5 elements. If none exists, why not?

does not exist. any model must be infinite, contain all p(g(g(...g(a)..)),b) by rules 1 and 5 of P.

4.2.8 (2 marks) List 6 elements of the least model of P.

many solutions, e,g,
p(a,b), p(g(a),b), p(g(g(a)),b), p(g(g(g(a))),b), p(g(g(g(g(a)))),b), p(g(g(g(g(g(a))))),b)

4.2.9 (2 marks) Describe a model of P that is not a least model.