THE UNIVERSITY OF BRITISH COLUMBIA
CPSC 312: MIDTERM EXAM
OCTOBER 6, 2005

Instructor: Kurt Eiselt

Name: _____ Student #: _____

Signature: _____

Notes about this examination

1. You have 80 minutes to write this examination. When you have completed the exam, please wait quietly at your seat until the examination period has concluded.
2. No notes, books, or any type of electronic equipment is allowed including cell phones, laptop computers, and calculators.
3. Good luck!

Rules Governing Formal Examinations

1. Each candidate must be prepared to produce, upon request, a Library/AMS card for identification.
2. Candidates are not permitted to ask questions of the invigilators, except in cases of supposed errors or ambiguities in examination questions.
3. No candidate shall be permitted to enter the examination room after the expiration of one-half hour from the scheduled starting time, or to leave during the first half hour of the examination.
4. Candidates suspected of any of the following, or similar, dishonest practices shall be immediately dismissed from the examination and shall be liable to disciplinary action.
    a. Having at the place of writing any books, papers or memoranda, calculators, computers, audio or video cassette players or other memory aid devices, other than those authorized by the examiners.
    b. Speaking or communicating with other candidates.
    c. Purposely exposing written papers to the view of other candidates. The plea of accident or forgetfulness shall not be received.
5. Candidates must not destroy or mutilate any examination material; must hand in all examination papers; and must not take any examination material from the examination room without permission of the invigilator.

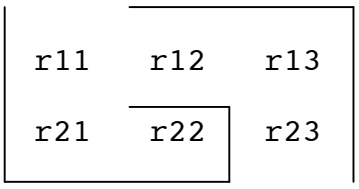
Question	Mark	Max
1		6
2		6
3		10
4		10
5		8
6		10
7		10
8		10
Total		**70**

Here's a familiar sight. It's the simple abstract interpreter from your textbook, but I've added a couple of print statements:

initialize resolvent to goal *G* (the query)
print resolvent
  while resolvent not empty *do*
    choose a goal *A* from the resolvent
    choose a (renamed) clause $A' :- B_1,\ldots,B_n$
     from program *P* such that *A* and *A'* unify
       with mgu Θ
       (if no such goal and clause exist, exit
       the while loop)
    replace *A* by $B_1,\ldots,B_n$ in the resolvent
    apply Θ to the resolvent and to *G*
    print resolvent
  If the resolvent is empty, *then* output *G*,
    else output *no*

And here is a very simple program, including a maze so simple that even a CPSC 111 student could navigate it:

```
connects_to(r11,r12).
connects_to(r11,r21).
connects_to(r21,r22).
connects_to(r12,r13).
connects_to(r13,r23).
```




```
path(X,Y) :- path(X,Z),connects_to(Z,Y).
path(X,Y) :- connects_to(X,Y).
```

You'll be using this information for both Questions 1 and 2 on the following page.

Question 1: [6 marks]

In the numbered spaces below, show what the nondeterministic simple abstract interpreter would print while trying to solve the query `?- path(r11,r23).` You might need all the spaces, or you might not. If you find that you need more than seven spaces, just stop after all seven spaces are used. I'll provide the result of executing the first print resolvent statment:

1. path(r11,r23)
2. _____
3. _____
4. _____
5. _____
6. _____
7. _____

Question 2: [6 marks]

Assume that the abstract interpreter on the previous page has been modified so that it is the Pure Prolog interpreter, and assume that the print statements remain. (Feel free to write those changes on the previous page if it helps, but it's not required.) Now show what the Pure Prolog interpreter would print as it tries to solve the same query: `?- path(r11,r23).` Again, you might need all the spaces, or you might not. If you find that you need more than seven spaces, just stop after all seven spaces are used. I'll provide the result of executing the first print resolvent statment:

1. path(r11,r23)
2. _____
3. _____
4. _____
5. _____
6. _____
7. _____

Question 3: [10 marks]

For each pair of terms below, give the most general unifier (Θ) if it exists.

a. `f(X,b,Z)` `f(a,Y,c)`

$\Theta =$

b. `f(X,Y)` `g(X,Y)`

$\Theta =$

c. `shuffle([H1|T1],[H2|T2],[H1|[H2|T3]])`
`shuffle([a,k,q],X,[a,j,k,10,q,9])`

$\Theta =$

d. `f(X,[4,5,6,7],Y,[5|M])` `f([A|B],[C|[D|E]],D,[C,K,L])`

$\Theta =$

e. `f(X,Y,[4,5,6,7],[5|M])` `f([A|B],[C|[D|E]],D,[C,K,L])`

$\Theta =$

Question 4: [10 marks]

Construct a procedure `decode(M1,M2)` which takes a secret coded message given by the list `M1` and proves that the message given by the list `M2` is the decoded version of `M1`. (Crypto freaks: yes, it's really a cipher, not a code. Forgive me.) You'll need a key for doing the decoding; here it is:

```
means(a,z).    means(z,a).
means(b,y).    means(y,b).
means(c,x).    means(x,c).
means(d,w).    means(w,d).
means(e,v).    means(v,e).
means(f,u).    means(u,f).
means(g,t).    means(t,g).
means(h,s).    means(s,h).
means(i,r).    means(r,i).
means(j,q).    means(q,j).
means(k,p).    means(p,k).
means(l,o).    means(o,l).
means(m,n).    means(n,m).
```

Here are some examples of correct behaviour for your `decode` procedure:

```
?- decode([r,o,l,e,v,k,i,l,o,l,t],X).

X = [i, l, o, v, e, p, r, o, l, o, g] ;

No
?- decode(Y,[i,l,o,v,e,p,r,o,l,o,g]).

Y = [r, o, l, e, v, k, i, l, o, l, t] ;

No
```

Question 5: [8 marks]

You might have noticed some symmetry in the key given in Question 4. In fact, you might think that you could delete half of the clauses in the key and replace them like this:

```
means(a,z).
means(b,y).
means(c,x).
means(d,w).
means(e,v).
means(f,u).
means(g,t).
means(h,s).
means(i,r).
means(j,q).
means(k,p).
means(l,o).
means(m,n).
means(X,Y) :- means(Y,X).
```

a. Assuming that your solution to Question 4 works, how does the change described above affect the behaviour of your `decode` procedure?

b. Would your `decode` procedure behave differently if the clause

```
means(X,Y) :- means(Y,X).
```

were moved from the bottom of the list of `means` predicates to the top of the list? Explain.

Question 6: [10 marks]

Construct a procedure `delete_at_index(Index,L1,L2)` which proves that list `L2` is the result of deleting the element from list `L1` indicated by the integer in `Index`. Here are some examples of `delete_at_index` in action:

```
?- delete_at_index(0,[a,b,c],X).

X = [b, c] ;

No
?- delete_at_index(1,[a,b,c],X).

X = [a, c] ;

No
?- delete_at_index(2,[a,b,c],X).

X = [a, b] ;

No
?- delete_at_index(3,[a,b,c],X).

No
?- delete_at_index(X,[a,b,c],[a,c]).
ERROR: Arguments are not sufficiently instantiated
```

[you don't need to worry about what happens in this last case]

Question 7: [10 marks]

Construct a procedure `sumall(L1,Sum)` which proves that the number `Sum` is the result of adding the values of all the numbers contained in the possibly-nested list `L1`. You may assume that the atomic (non-list) elements of `L1` are numbers. Here are some examples of `sumall` in action:

```
?- sumall([1,2,3,4],X).

X = 10 ;

No
?- sumall([1,[2,3,[4]]],X).

X = 10 ;

No
?- sumall([[[[4],3],2],1],X).

X = 10 ;

No
?- sumall(Y,10).
```

[you don't need to worry about what happens in this last case]

Question 8: [10 marks]

What does procedure `foo` do?

```
foo(X,[],0).
foo(X,[X|T],N) :- foo(X,T,N1),N is N1 + 1.
foo(X,[H|T],N) :- H\==X,foo(X,T,N).
```

_____