CSE 3401 – Functional and Logic Programming, Fall 2008
Department of Computer Science and Engineering
York University

Midterm (with Solutions)

Question 1

(10 points) Let P be the following set of predicate logic sentences:

$$\{\ \forall X \exists Y\, s(X,Y),\ \ \forall X \forall Y\, s(X,Y) \rightarrow g(Y,X)\ \},$$

and let g be the predicate logic sentence $\forall X \exists Y\, g(Y,X)$. Show that P logically implies g using resolution. Make sure to specify the m.g.u.'s in your resolution refutation.

Solution: first, convert the sentences in P and the *negation* of g to clauses. g must be negated because I asked to use resolution, and therefore the proof that P implies g must be done by contradiction.
First sentence (C_1)

$$\begin{aligned}&\forall X \exists Y\, s(X,Y)\\&\forall X s(X,f(X))\\&s(X,f(X)) :-\end{aligned}$$

Second sentence (C_2):

$$\begin{aligned}&\forall X \forall Y\, s(X,Y) \rightarrow g(Y,X)\\&\forall X \forall Y \neg s(X,Y) \vee g(Y,X)\\&g(Y,X) :- s(X,Y)\end{aligned}$$

Negation of the goal (C_3):

$$\begin{aligned}&\neg \forall X \exists Y\, g(Y,X)\\&\exists X \forall Y \neg g(Y,X)\\&\forall Y \neg g(Y,c)\\&:- g(Y,c)\end{aligned}$$

And now the resolution refutation:

1. $:- s(c,Y)$ (from C_2' and C_3 with $[Y'/Y, X/c]$)
2. $:-$ (from 1. and C_1 with $[X/c, Y/f(c)]$)

Question 2

(15 points) Give the Prolog definition of the predicate $proper\_suffix(L1, L2)$ which is true if the list $L2$ is a *proper* suffix of the list $L1$, that is, $L2$ is a suffix of $L1$, but not $L1$ itself. For example, $proper\_suffix([a, b], [])$, $proper\_suffix([a, b], [b])$ are true, but $proper\_suffix([a, b], [a, b])$ is not.
Give the recursive definition, and the definition using $append/3$. Explain the intended meaning of the clauses in your definitions.

(a) *(10 points)* Recursive definition.

Solution: $L2$ is a proper suffix of $L1$ if its either a tail of $L1$, or a proper suffix of a tail of $L1$, that is:

```
proper_suffix([H|T], T).
proper_suffix([H|T], L2) :- proper_suffix(T, L2).
```

Some people also had a clause

```
proper_suffix([X], []).
```

which is redundant, because its covered by the first one. Many people wrote this:

```
proper_suffix([X], []).
proper_suffix([H|T], L2) :- proper_suffix(T, L2).
```

If you were to trace the simplest example (e.g. `[a,b]`) you would realize that this gives only one solution, namely `[]`.

(b) *(5 points)* Definition using $append/3$. Reminder: $append(L1, L2, L3)$ is true if the list $L3$ is the result of concatenation of lists $L1$ and $L2$.

Solution: $L2$ is a proper suffix of $L1$ if $L1$ can be obtained by concatenation of some *non-empty* list, with $L2$:

```
proper_suffix(L1, L2) :- append([_|_], L2, L1).
```

Or, $L2$ is a proper suffix of $L1$ if the tail of $L1$ can be obtained by concatenation of some list (even empty) to $L2$ (in other words, $L2$ is a suffix of the tail of $L1$).

```
proper_suffix([H|T], L2) :- append(_, L2, T).
```

The $suffix/2$ was, by the way, one of the assigned exercises.

Question 3

(5 points) Write down Prolog's response to the following query:

```
?- f(2, [X,Y|T], [Y|R]) = f(Y, [1|[2,2]], T).
```

Solution:

```
X = 1
Y = 2
T = [2]
R = [] ;
No
```

Simply, run Robinson's algorithm to obtain this answer.

Question 4

(20 points) Write down Prolog's response to the query

```
?- p(X).
```

for each of the following four programs. If you believe that Prolog will respond with more than one answer, write down *all the answers in the order they will be returned by Prolog.* **Note:** programs in (b)-(d) are almost the same as in (a) – all changes are marked by comments.

(a) *(5 points)*

```
p(X) :- q(X,Y), Z is X + Y, r(Z).
p(X) :- r(X), X > 3.
q(X,X) :- r(X).
q(1,2).
r(2).
r(3).
r(4).
```

Solution:

```
X = 2 ;
X = 1 ;
X = 4 ;
No
```

Draw the resolution refutation tree.

(b) *(5 points)*

```
p(X) :- q(X,Y), Z is X + Y, r(Z).
p(X) :- r(X), X > 3.
q(X,X) :- r(X).
q(1,2).
r(2) :- !.                              % this clause differs from (a)
r(3).
r(4).
```

Solution:

```
X = 2 ;
X = 1 ;
No
```

Take the tree you drawn in (a) – the cut has no effect on the left subtree, because it only cuts off failed branches. However, on the right subtree (the one that corresponds to the second clause for $p(X)$), the cut eliminates a successful branch.

(c) *(5 points)*

```
p(X) :- q(X,Y), Z is X + Y, r(Z).
p(X) :- r(X), X > 3.
q(X,X) :- r(X), !.                      % this clause differs from (a)
q(1,2).
r(2).
r(3).
r(4).
```

Solution:

```
X = 2 ;
X = 4 ;
No
```

Take the tree you drawn in (a) – the cut removes only the subtree that corresponds to the clause $q(1,2)$.

(d) *(5 points)*

```
p(X) :- q(X,Y), Z is X + Y, \+ r(Z).  % this clause differs from (a)
p(X) :- r(X), X > 3.
q(X,X) :- r(X).
q(1,2).
r(2).
r(3).
r(4).
```

Solution:

```
X = 3 ;
X = 4 ;
X = 4 ;
No
```

Take the tree you drawn in (a) – every successful branch in the left subtree (the one that corresponds to the first clause for $p(X)$) becomes a failed branch, and visa versa. This gives the first two answers. The right branch is not affected by negation and gives the third answer, as in (a).