CSE 130: Programming Languages – Principles and Paradigms Fall 2000

Problem Set 4

Instructor: Daniele Micciancio Nov. 17, 2000

Due: Nov 29, beginning of class. No late submissions accepted.
Guidelines for the programming assignments: In this problem set you will use Prolog to write your programs. You can start the Prolog interpreter from ieng9 typing *sbprolog*. You can write your programs with any editor, and then load them from sbprolog with the command "*consult('filemane').*" . After the program is loaded, you can type any goal at the prompt. Together with your solutions you should submit a print out of the programs, and the result of executing them. You should also include a brief english description of how your programs work. Clarity of your programs (as well as the english description) is as important as correctness. In particular, in order to get full credit your programs should be properly formatted so to make them easy to read. No electronic copy of your programs is required.
Collaboration policy: As usual, you should do your homeworks individually. See collaboration policy on the web page for details.

Problem 1 (5 points)

For each of the following pair of terms, say if they can be unified, and if so, give a most general common instance, i.e. the most general term that is an instance of both.

- $g(X, f(X, Z))$ and $g(f(W, W), f(f(Y, a), Y))$
- $f(X, Y, g(X, Y))$ and $f(a, Z, g(Z, b))$
- $f(g(X, X, Y), V)$ and $f(g(W, a, Z), f(W, Z))$

Problem 2 (10 points)

Consider the following Prolog program

```
aa(san,sfo).
aa(nyc,bos).
aa(sfo,nyc).
us(san,nyc).
us(sfo,bos).
connected(X,X).
connected(X,Y) :- aa(X,Z), connected(Z,Y).
connected(X,Y) :- us(X,Z), connected(Z,Y).
```

where the predicates $aa/2, us/2$ *connected*$/2$ have the following informal interpretation:

- $aa(X, Y)$: there is a direct flight from airport X to airport Y operated by *American Airways*.

- *us*(X, Y): there is a direct flight from airport X to airport Y operated by *US Airlines*.
- *connected*(X, Y): it is possible to fly from X to Y on flights operated by *American Airways* or *US Airlines*.

Draw the search tree (a.k.a., execution tree) corresponding to the goal

```
connected(san,bos).
```

[Your tree should include both successful and unsuccessful paths. You can draw only the portion of the tree that is actually explored by Prolog, i.e., the set of nodes up to the point when the first successful path is found, or the computation enters an infinite loop. If the program does not terminates, you can of course draw only a finite portion of the search tree.] What is the route (i.e., the sequence of airports) corresponding to the first solution found by Prolog?

Now consider the following alternative definitions for the predicate "connected", obtained from the first definition either changing the order of the clauses, or changing the order of the predicates within each clause.

- Variant 1:

```
connected(X,X).
connected(X,Y) :- connected(X,Z), aa(Z,Y).
connected(X,Y) :- connected(X,Z), us(Z,Y).
```

- Variant 2:

```
connected(X,Y) :- connected(X,Z), aa(Z,Y).
connected(X,Y) :- connected(X,Z), us(Z,Y).
connected(X,X).
```

- Variant 3:

```
connected(X,Y) :- aa(X,Z), connected(Z,Y).
connected(X,Y) :- us(X,Z), connected(Z,Y).
connected(X,X).
```

For each of the above programs, give the search tree corresponding to the goal *connected*(san, bos), and say which route correspond to the first solution found (if any). If the program does not terminate, draw a finite portion of the search tree, and explain why the program goes into a loop. (A brief informal explaination is fine. Just write anything you would consider a convincing argument.)

Problem 3 (15 points)

In this problem you will define three different Prolog programs to sort a list of integer numbers, and compare their efficiency. In all three parts you are asked to complete the definition of a Prolog predicate $sortN(X, Y)$ (where $N = 1, 2, 3$) that is true if Y is a sorted version of X. So, for example,

- if you call $sortN([3, 2, 1], [1, 2, 3])$ you will get yes as an answer.
- if you call $sortN([1, 2, 3], [3, 2, 1])$ the answer will be no.
- if you call $sortN([3, 2, 1], X)$ you should get the solution $X = [1, 2, 3]$.

(a) Consider the sorting program defined in the textbook:

```
sort1(X,Y) :- permute(X,Y), sorted(Y).
sorted([]).
sorted([X]).
sorted([X,Y|List]) :- X =< Y, sorted([Y|List]).
```

The informal description of this sorting definition is the following: Y is the sorted version of X if Y is a permutation of X and the list Y is sorted (i.e., the elements of Y are in non-decreasing order.) Complete the above program defining Prolog predicate *permute*/2, where $permute(X, Y)$ is true if Y is a permutation of X, i.e., Y is a list containing the same elements as X, but in a possibly different order. [Hint: first define an auxiliary predicate $remove(X, Xs, Ys)$ where Ys is a list obtained from Xs removing a single occurrence of element X.]

Notice: the book uses the <= comparison operator, which is not defined in sbprolog. You should use =< instead.

(b) Another possible definition of predicate sort is the following:

```
sort2([],[]).
sort2([X|Xs],Zs) :- sort2(Xs,Ys), insert(X,Ys,Zs).
```

Describe (in english) which operation should be performed by predicate *insert*/3, in order to turn *sort*2 into a correct sorting procedure. Then give a Prolog implementation of *insert*/3.

(c) Finally, we consider the definition of the mergesort algorithm in Prolog:

```
sort3([],[]).
sort3([X],[X]).
sort3(Xs,Ys) :- shuffle(As,Bs,Xs),
                sort3(As,Cs),
                sort3(Bs,Ds),
                merge(Cs,Ds,Ys).
```

Give English descriptions and Prolog implementations of predicates $shuffle/3$ and $merge/3$ that turn $sort3$ into a correct sorting algorithm.

For each of the above algorithms (sort1, sort2, sort3), test if the algorithm is working correctly by applying it to some simple list (say at most 5 elements).

Once you have done some simple test, determine the maximum length of the lists that can be sorted by sort1, sort2, sort3 in reasonable time and space. (say, running time within a few seconds, and program terminating correctly without giving a heap overflow error message.)

You can generate test lists of arbitrary length (consisting of the number in reverse order) using the following predicate:

```
test(0,[]).
test(N,[N|T]) :- M is N-1, test(M,T).
```

For example you can test sort1 on the list $[5, 4, 3, 2, 1]$ asking the query:

```
test(5,X), sort(X,Y)
```

Your answers should read something like "Running sortX on lists of length N terminates correctly within S seconds. Running sortX on lists of length N' takes more than S' seconds, or terminates abnormally with a Heap overflow message."