COMP 280 : Assignment 9

due: Thursday, April 6, 2000

Write all programs in Prolog. Turn in a printout of your programs and your test cases. Be sure to test them before turning them in, testing both cases that should work and cases that should not. Write your programs declaratively (*e.g.* don't use the if-then-else construct). Document your programs with a comment describing the arguments and how they should relate.

Two Prolog tips for this assignment:

1. In class, we briefly discussed the table construct, which remembers the answers of previous queries to rules. To tell XSB to table the results from a rule my_rule of arity 2, write the following in your file:

   :- table my_rule/2.

   For this assignment, you should only table rules that need to be tabled to avoid infinite loops. Unnecessary tabling will result in lost points.

2. Prolog provides a built-in operator called setof for obtaining a list of all values that satisfy a given constraint. Here's an example of its use, using the following database:

   ```
   likes(bill, cider).
   likes(dick, beer).
   likes(harry, beer).
   likes(jan, cider).
   likes(tom, beer).
   likes(tom, cider).
   ```

   ```
   | ?-  setof(X, likes(X,Y), S).

   X = _h63
   Y = beer
   S = [dick,harry,tom],

   X = _h63
   Y = cider
   S = [bill,jan,tom],

   no
   | ?-
   ```

   setof(template, goal, list) returns a list of all values which satisfy goal, with the values formatted according to template. Here's another example.

   ```
   | ?- setof([beer, X], likes(X, beer), S).

   X = _h75
   S = [[beer,dick],[beer,harry],[beer,tom]],

   no
   ```

You should find both of these constructs useful for completing the assignment (starts on the next page).

1. (3 pts) Create a database of airline flight information. For each flight, you must store the airline, the flight number, the departure city, the arrival city, the departure time, and the arrival time. Represent the times as lists of two elements (hours and minutes) and use 24 hour/military time (*i.e.* [15,30] for 3:30pm). Your database should be sufficient to provide good test cases for the programs below.

2. (2 pts) Write a rule has_route(From_city, To_city), which is true when there exists a sequence of flights from From_city to To_city.

3. (2 pts) Write a rule produce_route(From_city, To_city, Route) which is true when Route is list of cities through which one can fly from From_city to To_city. Your flight database should contain a direct flight between each flight and the one that follows it in Route.

4. (3 pts) Write a rule produce_schedule(From_city, To_city, Schedule) which is true when Schedule is a list of details on the flights that one can take to get from From_city to To_city. Your rule should respect the following restrictions:

   (a) Schedule should be a list of lists, where each sublist contains the airline, flight number, arrival time and departure time for a flight on the route.

   (b) As in the previous program, there should be a direct flight between each city and the one that follows it in the schedule.

   (c) There should be at least one hour between the arrival time of each flight and the departure time of the next flight in the schedule.

5. (3 pts) Write a rule hub(Airline, City) which is true when City is a hub for Airline. A city is a hub if there are direct flights between the city and at least half of the cities serviced by the airline (an airline services a city if it has a flight either into or out of that city).

6. (3 pts) Write a rule simple_tour(Airline, City, Route) which is true when one can fly from City back to City following Route such that

   (a) All flights are on Airline,

   (b) No city appears more than once on the route other than City, which appears in the first and last positions on the route.

   (c) At least one city other than City appears in Route.

7. (2 pts) Write a rule full_tour(Airline, City, Route) which is true when one can fly from City back to City visiting every city which Airline services. This program is subject to the same restrictions as simple_tour. (This is known as the traveling salesman problem: can a salesman visit every city without repeating any visits.)