University of Toronto Mississauga
CSC 324 - Principles of Programming Languages, Fall 2016

Assignment 3

Due Monday December 5 at 11pm.
No late assignments will be accepted.

What to do

This assignment requires you to write queries and programs in Prolog. They should be written in good logic-programming style. Marks will be deducted for programs that look like translated Java or C. Simple programs are preferred, especially small collections of simple rules. Unless otherwise specified, you may assume that the inut to all predicates is correct, so that no error checking is required. *For full marks, all answers returned by Prolog must be correct; it is not sufficient for just the first answer to be correct.*

You should hand in five files: a listing of all your Prolog rules, a listing of all your Prolog queries, a transcript of a terminal session with the Prolog interpreter, a file of your solutions to the non-programming questions, and a signed cover sheet. These files should be submitted electronically using the UTORSubmit web page. The rules should be commented, and the terminal session should be short and should demonstrate that your rules and queries work correctly. Readability and well-written documentation are of great importance.

In this assignment, non-logical operations are rarely needed, and their use should be minimized. In particular, do *not* use `functor`, `arg`, `=..`, `assert`, `retract`, `fail` or `!` (cut). However, the built-in predicates `atom` and `number`, as well as negation, arithmetic operations (such as `+` and `-`), arithmetic predicates (such as `<`), and the `is` predicate may be needed from time to time, and you may use them when necessary. Unless specified otherwise, do not use any other built-in predicates. In particular, do not use the equality predicate, `=`, unless it reduces the size of your Prolog rules. Finally, do not use ; (semicolon), which stands for logical "or" in Prolog.

Note: The marker has a limited amount of time for each assignment, so it is your responsibility to provide documentation and testing that allows him to *quickly* evaluate your work. As with all work in this course, 20% of the grade is for quality of presentation.

No more questions will be added

1. **First-Order Logic (12 points).**

   This question uses the following predicates:

   - $company(x)$, which means that x is a company
   - $worker(x)$, which means that x is a worker.
   - $employs(x, y)$, which means that x employs y.
   - $person(x)$, which means that x is a person.
   - $sin(x, y)$, which means that x is the social insurance number of y.

   Using only these predicates, translate each of the English statements below into first-order logic. You may not use any function symbols, constant symbols, or arithmetic symbols, but you may use the equality predicate if necessary. You may use arbitrary formulas of first-order logic, not just Horn rules. You may also use he equality predicate, $=$, if necessary. (This is a paper-and-pencil problem, and no programming is required.)

   (a) Every worker is a person.

   (b) Every person has a social insurance number.

   (c) Different people have different social insurance numbers.

   (d) Every worker is employed by a company.

   (e) If a person is employed by a company, then he is a worker.

   (f) A company is not a person.

2. **Unification (8 points).**

   For each of the following pairs, state whether or not they will unify. In each case, state what the most general unifier is, or show that unification is impossible. If they unify, show the unification of the two atomic formulas. In this problem, $U.X, Y, Z$ are variables and a, b, c, d are constant symbols. (This is a paper-and-pencil problem, and no programming is required.)

   (a) $p(b, b, a)$ and $p(X, Y, Z)$.

   (b) $p(f(a, b), c, d)$ and $p(f(X, Y), X, Z)$.

   (c) $p(f(a, b), a, d)$ and $p(f(X, Y), X, Z)$.

   (d) $p(a, Z, Z)$ and $p(Y, Y, b)$.

   (e) $q(X, f(X, Y), Y)$ and $q(U, f(Z, c), Z)$.

   (f) $q(f(X, Y), Y, X)$ and $q(Z, U, U)$.

   (g) $q(X, Z, Z)$ and $q(U, U, Y)$.

   (h) $q(X, f(Y, g(X, a)))$ and $q(h(U, a), Z)$.

3. **Database Construction (5 points).**

Create a Prolog database for an airline company. (This database will be used in the next two questions.) The database consists of ground atomic formulas made from the following predicates:

```
plane(plane#,model,age)
flight(flight#,from,to,plane#)
passenger(sin,name,city,age)
booked(sin,flight#,date,price)
```

These predicates are interpreted as follows. Each plane has a unique plane number (`plane#`), a model, and an age. Each flight has a unique flight number (`flight#`) and flies from one city to another. A flight may fly between several cities (eg, from Toronto to Paris, and then from Paris to Frankfurt.) Each flight is assigned a unique plane, identified by its plane number. Each passenger has a name, a city, an age, and a unique social insurance number (`sin`). Many passengers may have the same name and city. Passengers are booked on a flight on a given date for a given price. A passenger may be booked on many flights, even on the same day.

Here are typical atomic formulas for each of the four predicates:

```
plane(p123,boeing747,8)
flight(f407,toronto,paris,p123)
passenger(s456_234_987,anna,toronto,26)
booked(s456_234_987,f407,jan17,500)
```

- The first formula means that plane p123 is a Boeing 747 and is 8 years old.
- The second formula means that flight f407 goes from Toronto to Paris and is assigned plane p123.
- The third formula means that Anna lives in Toronto, is 26 years old, and her social insurance number is s456_234_987.
- The fourth formula means that person s456_234_987 is booked on flight f407 on January 17, and she paid $500 for the booking.

Create a Prolog database of at least 40 atomic formulas using the four predicates above. Choose the data so that the queries in Questions 4 and 5 return interesting non-trivial answers (e.g., not always an empty answer, not always the same answer, not always a single answer).

4. **Database Queries (10 points).**

   Without adding any rules to your Prolog database, write Prolog queries that retrieve the the information below (and no more). Test your queries on the database you created in Question 3.

   (a) The model and age of plane p123.

   (b) The plane number and age of every boeing 747.

   (c) The flight number of every plane going from New York to Paris.

   (d) The flight number and price paid by every passenger going from Los Angeles to Paris on Jan 17.

   (e) The name and SIN of every passenger booked on flight f237 on Feb 23.

5. **Deductive Databases (15 points).**

   Add rules to your Prolog database that define the predicates below. Of course, your rules should work for any reasonable set of data, not just the particular data in your database. This property of a rulebase is called *data independence*.

   (a) `sn(sin,name)`. True iff a passenger with the given SIN has the given name.

   (b) `child(sin,name)`. True iff a passenger with the given SIN and name is less than 12 years old.

   (c) `nfd(name,flight#,date)`. True iff a passenger with the given name is booked on the given flight for the given data.

   (d) `sm(sin,model)`. True iff a person with the given SIN is booked on a flight that uses a plane of the given model.

   (e) `ccca(city1,city2,city3,age)`. True iff a passenger who lives in `city1` is booked on a flight from `city2` to `city3` on a plane that is more than the given age.

   Test your rules on the database you created in Question 3 by using them to answer queries. For example,

   – Find the names of all passenger booked on flight f407 on January 17.
   – Find all flights on which Anna is booked on February 23.

6. **Negation, Disjunction and Existence (15 points).**

Add rules to your Prolog database that define the predicates below. These rules all require the use of negation, disjunction or existence. As above, your rules should be data independent.

(a) `existsplane(planenum)`. True iff there is a plane with the given plane number (according to the `plane` predicate).

(b) `badflight(flightnum)`. True iff the plane assigned to the given flight number does not exist (according to the `plane` predicate).

(c) `noflights(sin1,sin2,date)`. True iff there are no flights on the given date from the city in which the passenger with `sin1` lives to the city in which the passenger with `sin2` lives.

(d) `serves(flightnum,city)`. True iff the flight with the given flight number flies into or out of the given city.

(e) `notserved(sin)`. True iff the city in which the passenger with the given sin lives is not served by any flights.

Test your rules on the database you created in Question 3 by using them to answer queries. For example,

– Find all bad flight numbers.
– Find the SINs of all passengers whose cities are not served by any flights.
– Find all `sin1` and `sin2` for which `noflights(sin1,sin2,jan17)` is true.

7. **Function Terms and Recursion (44 points total).**

In this question you will use function terms to represent trees. Each node of a tree stores a number, and each internal node has 1, 2 or 3 children. Specifically, the function term `node1(X,T)` represents a node that stores number `X` and has one child, `T`. The function term `node2(X,T1,T2)` represents a node that stores number `X` and has two children, `T1` and `T2`. The function term `node3(X,T1,T2,T3)` represents a node that stores number `X` and has three children, `T1`, `T2` and `T3`. Finally, the function term `leaf(X)` represents a leaf that stores number `X`.

Your logic programs below should traverse a tree at most once. Do *not* use the equality predicate, `=`, and do not define your own equality predicate.

(a) (4 points) Draw a tree and show how it is represented in Prolog. Your tree should have at least one node with one child, at least one with two children, and at least one with three children.

(b) (10 points) Define a Prolog predicate `treeMax(T,X)` that is true iff `X` is the largest number stored in tree `T`. Hint: define a predicate `max(X,Y,Z)` that is true iff `Z` is the maximum of `X` and `Y`. (You will need two simple rules for this.) Do *not* use the `is` predicate in this question.

(c) (10 points) Define a Prolog predicate `treeRev(T1,T2)` that is true iff tree `T2` is identical to tree `T1` except that each node has the order of its children reversed. Your program should work with either `T1` or $\widehat{\texttt{T2}}$ as input, and the other as output. For example,

```
| ?- treeRev(node2(7,leaf(1),node3(8,leaf(2),leaf(3),leaf(4))), T2).
     T2 = node2(7,node3(8,leaf(4),leaf(3),leaf(2)),leaf(1))

| ?- treeRev(T1, node2(7,node3(8,leaf(4),leaf(3),leaf(2)),leaf(1))).
     T1 = node2(7,leaf(1),node3(8,leaf(2),leaf(3),leaf(4)))
```

(d) (10 points) Define a Prolog predicate `countNodes(T,c(N0,N1,N2,N3))` that is true iff `N0` is the number of leaves in tree `T`, `N1` is the number of nodes with 1 child, `N2` is the number of nodes with 2 children, and `N3` is the number with 3 children.

(e) (10 points) Define a Prolog predicate `treeList(T,L)` that is true iff `L` is a list that contains all the numbers stored in tree `T` in nodes having 1 or 2 children. The numbers should appear in list `L` in the same order (and with the same frequency) as they appear in the tree. For example,

```
| ?- treeList(node2(7,node1(3,leaf(2)),node1(7,leaf(9)))),L).
     L = [7,3,7]
```

You may use the built-in Prolog predicate `append`.

University of Toronto Mississauga
Department of Mathematical and Computational Sciences
CSC 324 - Principles of Programming Languages

Cover sheet for Assignment 3

Complete this page and hand it in during class or tutorial.

Name: Sayantan Chattopadhyay
(Underline your last name)

Student number: 1000818145

I declare that this assignment is solely my own work, and is in accordance with the University of Toronto Code of Behavior on Academic Matters.

Signature: