

University of Toronto at Scarborough
CSC C24
Prolog Programming Project

Due Friday, March 30, 2001
12:30pm electronically and 1:00pm in the dropbox

This assignment will test your knowledge of Prolog. It is worth 10% of your final grade. Your code will be graded on correctness, readability, and the use of logic programming style. Your task is to write Prolog predicates for manipulating lists of players in the National Hockey League (NHL). You may use helper relations as needed in defining these predicates.

You may not use any of the following in any predicate/relation you write:

- ; (“or”)
- -> (“if-then”)
- ! (cut)
- Any other operator that is not part of “true” logic programming. If you have a question about something in particular that we haven’t covered in class, then ask me.

1 Problem Description

1. Copy the file `players.P` posted on the class website into your working directory. The file contains a database of facts about hockey players that you may use to test and debug your program. The following is a portion of the database illustrating the three relations that are defined:

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Database of NHL Players %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

position(gretzky, center).
position(kariya, leftwinger).
position(kiva, rightwinger).
position(coffey, defender).
position(hasek, goalie).

language(gretzky, english).
language(gretzky, french).
language(kiva, piglatin).

birthplace(gretzky, ontario).
birthplace(hasek, czechrepublic).
birthplace(kiva, usa).

```

Each player that the database knows about must be specified to play exactly one position (center, leftwinger, rightwinger, defender, or goalie). That is, every player X will have exactly

one predicate `position(X,P)` in the database for some value of `P`. A player participates in at most one `birthplace` relation, but may participate in more than one `language` relation (such as `gretzky` above).

You will implement additional predicates as described below, as well as any necessary “helper” predicates to support them, that use these facts to create collections of players with different constraints on them.

You will need to load the database file, or a similar database of your own creation, into Prolog, but **the file you turn in must not contain the database relations**. You will lose points on the project if it does. You must put the predicates that you implement into a separate file called `hockey.P`, as described in Part 3 below. Note that you must put **all** your predicates in the same file. Do **not** create a different file for each predicate.

2. Implement the predicate “`playerList(L,N)`.” The variable `L` is the list of players and the variable `N` is the number of players that are on the list.

The `playerList` predicate must obey the following conditions:

- (a) The list `L` must have exactly `N` members: $[P_1, P_2, \dots, P_N]$.
- (b) Each element of `L` must be a valid player. A player is valid if s/he participates in a `position` predicate.
- (c) Each element of `L` must be unique.

For this and each of the predicates described below, you should be able to specify queries with any combination of variables and constants. For example:

- `playerList(X,4)`.
- `playerList([gretzky,kiva,hasek],3)`.
- `playerList([coffey|Y],N)`.

NOTE: For this and remaining predicates, it is unwise to simultaneously specify the list argument as a variable and the number of players as anything greater than 6. Perhaps surprisingly, this is not the case if **both** arguments are variables, as in `playerList(X,N)`. Try it and see if you can figure out why.

(What will happen if you implement the predicate `playerList` (correctly) and type `playerList(X,12)` to the system? It will take a *very long time* to compute, and you’ll need to use control-c followed by `a` (for abort) to cut it off.)

3. Implement the predicate “`teamRoster(L,N)`.” The values of the variables `L` and `N` must meet **all** the conditions for “`playerList(L,N)`.” In addition, for this predicate, the list `L` must satisfy additional constraints to ensure that the list of players forms a valid team:
 - (a) The list `L` must have the same number of centers, rightwingers, and leftwingers. These form the forward line in hockey, and the team must consist of a certain number of forward lines.
 - (b) The list `L` must have a number of defenders that is a multiple of two. Two defenders form the defensive pair in hockey, and the team must consist of a certain number of defensive pairs.

- (c) The list L must have at least one forward line, at least one defensive pair, and at least one goalie.
- (d) The number of forward lines, defensive pairings, and goalies does not have to be the same.

So, for example, the list L may consist of two centers, two rightwingers, two leftwingers, six defenders, and one goalie.

Note that in meeting these constraints, it is not a requirement that the players that play the same position all be grouped together in the list L.

4. Implement the predicate “lockerAssignment(L,N).” As before, the variables L and N must meet **all** the conditions for “playerList(L,N).” In addition, for this predicate, the list L will be considered as an **ordered** list in which the ordering represents the assignment of those N players to a row of N lockers in the locker room. It is required in making this assignment that no two adjacent lockers are assigned to players that play the same position. (We don’t want “locker neighbors” to feel too competitive!)

Precisely, the predicate “lockerAssignment(L,N)” must satisfy the constraint that for every pair of players P_i and P_{i+1} in the ordered list L, P_i and P_{i+1} do not play the same position. NOTE: the lockers are in a row, not a circle, thus P_1 and P_N can play the same position.

5. Implement the predicate “banquetTalk(L,N).” Again, the variables L and N meet **all** the conditions for “playerList(L,N).” As in `lockerAssignment`, L is considered an ordered list that represents an assignment of the players in L to a sequence of N positions—in this case, the seats at a banquet table. Here, we must ensure that every player in L is adjacent (in table seating) to someone who speaks the same language. (That way every player can talk to one of their table neighbors at the end of season banquet.)

Precisely, the predicate “banquetTalk(L,N)” must satisfy the constraint that for every player P_i on the ordered list L, **either** P_i and P_{i+1} share a language, **or** P_i and P_{i-1} share a language. NOTES: Table seating is in a row, so this definition means that P_1 must share a language with P_2 , and P_{N-1} must share a language with P_N .

6. Implement the predicate “commonBackground(L,N).” Again, the variables L and N meet **all** the conditions for “playerList(L,N),” plus the additional condition that for every player on L, there must be at least one other (different) player on L that was born in the same place. This constraint ensures that every player on the list shares a common background with someone else on the list.

2 The Prolog Database

Your Prolog predicates should operate on a database of facts about NHL players, as illustrated above; a complete sample database called `players.P` is on the website. The database file `players.P` must be loaded into your Prolog session for you to have access to the facts defined in it. Your own predicates should be defined in a different file, `hockey.P`. **Do not copy the facts from `players.P` into the `hockey.P` file that you hand in. Do not create a different file for each predicate—place all your predicates in file `hockey.P`.**

We will test your program on our own test database of facts about NHL players. The test file will have the same relations describing players as the data base mentioned above. However, the

constants used in the relations in the test file—and therefore the actual facts about players—will be different from the facts we provide you for debugging.

3 To Be Handed In

3.1 On paper

Staple (no paperclips, no folded corners!) the following together with the cover sheet (available on the course web page) on top:

- A printout of all of your code.
- A printout of one or more script files containing your testing of each predicate. (See `man script` if you are unsure of how to create a script file.)
- A brief written report on your testing strategy.

3.2 Electronically

In addition to your paper submission, you must submit your code electronically.

- Put the definitions of the predicates “playerList,” “teamRoster,” “lockerAssignment,” “banquetTalk,” and “commonBackground,” as well as any supporting (“helper”) relations, in a file called `hockey.P`.
- Make sure the file `hockey.P` contains a comment at the top that gives your name, your login ID, your student ID number, and your tutorial section.
- **Do not** turn in any facts that are in the database file `players.P` that we provide to you.
- **Do not** create a separate file for each predicate—all your predicates must be in the same file, `hockey.P`.
- Submit the file `hockey.P` using the following command on `fissure.scar`:

```
submit -N p2 csc24s hockey.P
```

See `man submit` if you want more information about how the submit command works. Note that if you have already submitted a file and you wish to replace it with a different version, you can submit it again. But you must use the `-f` option to warn submit that you really mean to overwrite the old version.