CPSC 449 Assignment 3

Due: Friday December 8, 2017 at 12:00 noon

Individual Work:

All assignments in this course are to be completed individually. Students are advised to read the guidelines for avoiding plagiarism located on the course website. Students are also advised that electronic tools may be used to detect plagiarism.

Late Penalty:

Late assignments will not be accepted.

Errors in the Provide Code:

There is a fair bit of provided code in this assignment, and it's possible that it contains an error (or even more than one error). If you find an error, please let the course instructor know about it so that the error can be corrected for all students.

Background

In this assignment, you are going to complete a Prolog knowledge base that contains facts and rules about the prerequisite relationships between courses in Computer Science. The starter code that I have provided already includes the following information:

- Facts and rules that describe the prerequisites for non-computer science courses[1]
- Facts about consent of the department courses
- Facts about prerequisites for computer science courses that can only have their prerequisites satisfied in one way
- A rule that generates all of the different combinations of direct prerequisites for CPSC 331
- Facts about courses that are antirequisite to each other

You will make several extensions to this knowledgebase through the remaining parts of this assignment.

Part 1:

Add rules to the knowledgebase that encode the prerequisites for the computer science courses numbered between 217 (inclusive) and 599 (inclusive) that are not already listed in the knowledgebase (313, 319, 329, 335, 355, 359, 411, 413, 418, 433, 441, 449, 453, 457, 461, 471, 481, 491, 501, 518, 519, 521, 525, 530, 531, 535, and 583). Information about the prerequisites for the courses can be found at http://www.ucalgary.ca/pubs/calendar/current/computer-science.html.

Like the facts and rules already in the knowledgebase, your encoding for these courses should use the prereqFor functor. The first parameter is the course number, including its 4 letter department prefix, and its second parameter should be a list of courses that must be taken before a student is eligible to take the course provided as the first parameter. All of these courses have multiple combinations of courses that can be used to fulfill their prerequisites. Your rules should ensure that Prolog generates all

[1] The provided prerequisites have been simplified for a couple of math courses, and only consider university level (rather than high school level) requirements.

such combinations through backtracking using a strategy similar to that of the provided rules for amat219, math253 and cpsc331 (among others).

Once you have completed this part of the assignment you should be able to use Prolog to query the prerequisites for any undergraduate computer science course. For example, the query prereqFor(cpsc313, X) should return the following 12 responses (the order in which the responses are returned is not important).

X = [math271,phil279,cpsc219] ? ;
X = [math271,phil279,cpsc233] ? ;
X = [math271,phil279,cpsc235] ? ;
X = [math271,phil377,cpsc219] ? ;
X = [math271,phil377,cpsc233] ? ;
X = [math271,phil377,cpsc235] ? ;
X = [math273,phil279,cpsc219] ? ;
X = [math273,phil279,cpsc233] ? ;
X = [math273,phil279,cpsc235] ? ;
X = [math273,phil377,cpsc219] ? ;
X = [math273,phil377,cpsc233] ? ;
X = [math273,phil377,cpsc235]

Courses that do not have any prerequisites (such as CPSC 217) will instantiate the variable provided as the second parameter to the empty list, indicating that they do not have any prerequisites.

You must encode the prerequisites for the courses as rules rather than as long lists of facts. For example, while you could encode CPSC 319 using 4 facts:

```
prereqFor(cpsc319, [cpsc219]).
prereqFor(cpsc319, [cpsc233]).
prereqFor(cpsc319, [cpsc235]).
prereqFor(cpsc319, [encm339]).
```

Such an encoding is cumbersome, particularly when the number of combinations is large (CPSC 313 has a dozen different valid prerequisite combinations, and CPSC 413 has 24 – you don't want to have to list all of them off separately). Instead, the prerequisites for such courses can use the member predicate and backtracking to generate all combinations.

With judicious use of rules involving member, the prerequisites for all of the courses can be encoded in less than 75 lines of code (and one could write the code more compactly by placing more than one clause on each line if one wanted to). Note that you do not need to encode anything for notes like "Prior complete of CPSC XXX is strongly recommended".

Part 2:

Write a Prolog rule named allPrereqFor that identifies all combinations of courses that can be used to satisfy the prerequisites for a course (both its direct prerequisites and recursively, all of its prerequisites prerequisites). You should sort each generated list using the sort/2 built-in predicate, both because that

predicate merges duplicate elements so that you never see the same course listed twice, and because it will make it easier for you to compare your results to the provided test cases. Note that this rule will generate a large number of combinations for some courses. For example, allPrereqFor(cpsc413, X) generates over 600 possible combinations because of all the different ways that the math prerequisites can be satisfied. Applying it to cpsc331 generates the following 8 results:

```
| ?- allPrereqFor(cpsc331, X).
X = [cpsc217,cpsc219,math211,math271] ? ;
X = [cpsc231,cpsc233,math211,math271] ? ;
X = [consent235,cpsc235,math211,math271] ? ;
X = [encm339,engg233,math211,math271] ? ;
X = [cpsc217,cpsc219,math273] ? ;
X = [cpsc231,cpsc233,math273] ? ;
X = [consent235,cpsc235,math273] ? ;
X = [encm339,engg233,math273]
```

Applying it to cpsc501 generates the following 24 results:

```
| ?- allPrereqFor(cpsc501, X).
X = [cpsc217,cpsc219,cpsc319,cpsc449,phil279] ? ;
X = [cpsc231,cpsc233,cpsc319,cpsc449,phil279] ? ;
X = [consent235,cpsc235,cpsc319,cpsc449,phil279] ? ;
X = [cpsc319,cpsc449,encm339,engg233,phil279] ? ;
X = [cpsc217,cpsc219,cpsc319,cpsc449,phil377] ? ;
X = [cpsc231,cpsc233,cpsc319,cpsc449,phil377] ? ;
X = [consent235,cpsc235,cpsc319,cpsc449,phil377] ? ;
X = [cpsc319,cpsc449,encm339,engg233,phil377] ? ;
X = [cpsc217,cpsc219,cpsc331,cpsc449,math211,math271,phil279] ? ;
X = [cpsc231,cpsc233,cpsc331,cpsc449,math211,math271,phil279] ? ;
X = [consent235,cpsc235,cpsc331,cpsc449,math211,math271,phil279] ? ;
X = [cpsc331,cpsc449,encm339,engg233,math211,math271,phil279] ? ;
X = [cpsc217,cpsc219,cpsc331,cpsc449,math273,phil279] ? ;
X = [cpsc231,cpsc233,cpsc331,cpsc449,math273,phil279] ? ;
X = [consent235,cpsc235,cpsc331,cpsc449,math273,phil279] ? ;
X = [cpsc331,cpsc449,encm339,engg233,math273,phil279] ? ;
X = [cpsc217,cpsc219,cpsc331,cpsc449,math211,math271,phil377] ? ;
X = [cpsc231,cpsc233,cpsc331,cpsc449,math211,math271,phil377] ? ;
X = [consent235,cpsc235,cpsc331,cpsc449,math211,math271,phil377] ? ;
X = [cpsc331,cpsc449,encm339,engg233,math211,math271,phil377] ? ;
X = [cpsc217,cpsc219,cpsc331,cpsc449,math273,phil377] ? ;
X = [cpsc231,cpsc233,cpsc331,cpsc449,math273,phil377] ? ;
```

X = [consent235,cpsc235,cpsc331,cpsc449,math273,phil377] ? ;

X = [cpsc331,cpsc449,encm339,engg233,math273,phil377]

My solution to this part of the assignment is approximately a dozen lines of code, consisting of the allPrereqFor rule and a second rule to help generate the required results.

Part 3:

You may have noticed that some of results generated by allPrereqFor were contradictory insomuch as they included pairs of courses that can't be taken together, such as both cpsc231 and cpsc217, or both cpsc319 and cpsc331. For example, the first four results generated for cpsc441 by my implementation are:

| ?- allPrereqFor(cpsc441, X).

X = [cpsc217,cpsc219,cpsc319,cpsc355,cpsc359,phil279] ? ;

X = [cpsc217,cpsc219,cpsc231,cpsc233,cpsc319,cpsc355,cpsc359,phil279] ? ;

X = [consent235,cpsc217,cpsc219,cpsc235,cpsc319,cpsc355,cpsc359,phil279] ? ;

X = [cpsc217,cpsc219,cpsc319,cpsc355,cpsc359,phil377] ? ;

…

Notice that both the second and third results are contradictory lists – the second result includes both cpsc217 and cpsc231 (in addition to both cpsc219 and cpsc233), and the third list includes both cpsc235 and cpsc217 (and cpsc219).

Using the antirequisite information that I already included in the knowledgebase, create a Prolog rule named allPrereqFor_NoAnti. It will generate a sorted list (with duplicates removed) of all of the prerequsiites for a course (both its direct prerequisites, and recursively, all of its prerequisites prerequisites) such that the generated lists never contain a pair of courses that are antirequisite to one another. For example, allPrereqFor(cpsc457, X) generates 96 different lists of courses, many of which contain antirequisites. In contrast, allPrereqFor_NoAnti(cpsc457, X) generates the following 21 combinations:

| ?- allPrereqFor_NoAnti(cpsc457, X).

X = [cpsc217,cpsc219,cpsc319,cpsc355,cpsc359,phil279] ? ;

X = [cpsc217,cpsc219,cpsc319,cpsc355,cpsc359,phil377] ? ;

X = [cpsc231,cpsc233,cpsc319,cpsc355,cpsc359,phil279] ? ;

X = [cpsc231,cpsc233,cpsc319,cpsc355,cpsc359,phil377] ? ;

X = [consent235,cpsc235,cpsc319,cpsc355,cpsc359,phil279] ? ;

X = [consent235,cpsc235,cpsc319,cpsc355,cpsc359,phil377] ? ;

X = [admission_to_enel_or_ensf,cpsc319,encm339,encm369,enel353,engg233] ? ;

X = [cpsc217,cpsc219,cpsc331,cpsc355,cpsc359,math211,math271,phil279] ? ;

X = [cpsc217,cpsc219,cpsc331,cpsc355,cpsc359,math211,math271,phil377] ? ;

X = [cpsc231,cpsc233,cpsc331,cpsc355,cpsc359,math211,math271,phil279] ? ;

X = [cpsc231,cpsc233,cpsc331,cpsc355,cpsc359,math211,math271,phil377] ? ;

X = [consent235,cpsc235,cpsc331,cpsc355,cpsc359,math211,math271,phil279] ? ;

X = [consent235,cpsc235,cpsc331,cpsc355,cpsc359,math211,math271,phil377] ? ;

X = [cpsc217,cpsc219,cpsc331,cpsc355,cpsc359,math273,phil279] ? ;

X = [cpsc217,cpsc219,cpsc331,cpsc355,cpsc359,math273,phil377] ? ;

X = [cpsc231,cpsc233,cpsc331,cpsc355,cpsc359,math273,phil279] ? ;

X = [cpsc231,cpsc233,cpsc331,cpsc355,cpsc359,math273,phil377] ? ;

X = [consent235,cpsc235,cpsc331,cpsc355,cpsc359,math273,phil279] ? ;

X = [consent235,cpsc235,cpsc331,cpsc355,cpsc359,math273,phil377] ? ;

X = [admission_to_enel_or_ensf,cpsc331,encm339,encm369,enel353,engg233,math211,math271] ? ;

X = [admission_to_enel_or_ensf,cpsc331,encm339,encm369,enel353,engg233,math273] ? ;

My solution to this part of the assignment is approximately a dozen lines of code.

Part 4:

Create a Prolog rule named neededCourses that identifies sets of courses that can be taken in order to allow enrollment in a desired course. The rule will take a list of courses that the student has completed previously as its first parameter, and a single course that is desired as its second parameter. The final parameter will be an uninstantiated variable that will be instantiated with a list of courses that the student needs to take in order to take the desired course. The generated lists of courses should be minimal, meaning that they should not include courses outside of the prerequisite chain for the course in question. In many cases, there will be multiple minimal options. All such options should be generated via backtracking.

Note that the lists generated in this part of the assignment should not contain any courses that are antirequities to courses that the student has already taken. For example, if the student has already taken cpsc231 and cpsc233 and wants to take cpsc331 then your rule should generate results indicating that the student only needs to take either math211 and math271, or that the student needs to take math273. Don't return any results that include cpsc217, cpsc219 or cpsc235.

Consider the following examples:

| ?- neededCourses([cpsc231, cpsc233], cpsc457, X).

X = [cpsc319,cpsc355,cpsc359,phil279] ? ;

X = [cpsc319,cpsc355,cpsc359,phil377] ? ;

X = [cpsc331,cpsc355,cpsc359,math211,math271,phil279] ? ;

X = [cpsc331,cpsc355,cpsc359,math211,math271,phil377] ? ;

X = [cpsc331,cpsc355,cpsc359,math273,phil279] ? ;

X = [cpsc331,cpsc355,cpsc359,math273,phil377] ? ;

| ?- neededCourses([], cpsc219, Y).

Y = [cpsc217]

| ?- neededCourses([cpsc231, cpsc233, math211, math249, math271, cpsc331], cpsc585, Z).

Z = [consent585,cpsc453,math253] ? ;

Z = [consent585,cpsc453,math267] ? ;

Z = [amat219,consent585,cpsc453] ? ;

Hint: You may find it helpful to look at the list predicates in Section 8.20 of the gProlog manual located at http://www.gprolog.org/manual/gprolog.html#sec209. My solution to this part of the assignment was less than 10 lines of code.

Additional challenge:

Write a rule that identifies which courses a student still needs to take to complete the Concentration in Computer Graphics. The requirements for the concentration can be found near the bottom of this page: http://www.ucalgary.ca/pubs/calendar/current/sc-4-3-1.html (searching for graphics will help you find it quickly). The first parameter to the rule will be a list of courses that the student has already taken. The second parameter will be an uninstantiated variable. Your rule should instantiate the variable with minimal combinations of courses (through backtracking) that the student can use to complete the Concentration in Computer Graphics.

Grading:

A+: All parts of the assignments are completed successfully, included the additional challenge.

A: All parts of the assignment, except for the additional challenge, are completed successfully.

B: Parts 1, 2 and 3 of the assignment are completed successfully.

C: Parts 1 and 2 of the assignment are completed successfully.

D: Part 1 of the assignment is completed successfully.