Assignment 5
Due: July 27, 2023 at 11:59PM

1 Project setup

Follow the setup/import/export instructions from lab 7. The skeleton code at `https://classroom.github.com/a/_iNA7Uu3` is the start of this assignment.

2 Assignment 5 Specification

2.1 Overview

2.1.1 Goals

1. Practice writing Prolog to handle relational data
2. Practice extending a Prolog knowledge base to test predicates

Note: Your computer may recognize **.pl** files as Perl files, as they use the same file extension as Prolog files. Don't let the code highlighting distract you.

2.2 Example Database

For the first part of the assignment, refer to **a5a.pl**. This part of the assignment extends the database work that you did for a lab.

```
classtime(1000,10).          where(1000,dobbs102).          enroll(mary,1200).
classtime(1200,12).          where(1200,dobbs118).          enroll(john,3400).
classtime(3400,11).          where(3400,wentw216).          enroll(mary,3350).
classtime(3350,12).          where(3350,wentw118).          enroll(john,1000).
classtime(2350,11).          where(2350,wentw216).          enroll(jim,1000).
```

2.3 Predicates

Define the following predicates by writing a rule or rules for them. The examples and results use the facts given above. You will have to write some new facts to test your predicates correctly and completely.

2.3.1 roomconflict/2

A conflict exists if two **different** courses are using the same classroom at the same time. `?- roomconflict(X, 3350).` should return `false.` The query `roomconflict(X,Y).` should return all room conflicts in the database. The two arguments are course numbers.

2.3.2 meet/2

This predicate will decide if two **different** students (given as the two arguments) can meet each other, according to their schedules. There are two ways that two students can meet: either they can meet by being enrolled in the same course at the same time, or they can have different courses in the same classroom at adjacent times (off by an hour).

For this predicate, you need to return true for at least one ordering of the query. For example, `meet(mary, john).` might be true, or `meet(john, mary).` might be true. If your query returns true for both pairs, that is also fine. Assume that all classes start and stop exactly on the hour.

2.4 Extended equine relatives

For the second part of the assignment, refer to **a5b.pl**. The program already has a database of racehorses, including facts about their sex and parentage. Define four predicates as follows. You may write supporting rules if they help your definitions.

1. First of all, write a predicate called **grandparent/2** that returns true when the first argument is a grandparent of the second. If either of the arguments is a variable, it should return all possible values for a grandparent/grandchild.

2. Secondly, define an **aunt/2** predicate that says whether the first argument is an aunt of the second argument.

3. Third, define a **childless/1** predicate that returns true if the argument is a listed horse that has no child. You might find the **\+/2** predicate at the bottom of the `https://www.swi-prolog.org/pldoc/man?section=control` page to be helpful. If the argument is a variable, it should find all horses that have no children.

4. Lastly, define an **immediate/2** predicate that determines if two horses are in each other's immediate family. For this assignment, the following family relationships count as immediate family: parent, child, spouse (sharing a child), sibling (sharing at least 1 parent), self. If there are multiple solutions, the order does not matter.

Expected console output for some queries:

```
?- grandparent(northernDancer, X).
X = wellSpoken; X = galileo; X = danehill.

?- grandparent(pasDeNom, danehill).
true.

?- grandparent(X, cunco).
X = wellSpoken; X = danehillDancer; X = galileo; X = kind.

?- aunt(X, frankel).
X = wellSpoken.

?- aunt(galileo, danzig).
false.

?- childless(danzig).
false.

?- immediate(northernDancer, danzig).
true.

?- immediate(saintlySpeech, X).
X = sadlersWells; X = wellSpoken; X = saintlySpeech.
```

3 Testing

Both files have example databases. Test your predicates more robustly by changing or adding to the database. Make sure that your predicates work with both positive and negative results – they should not run forever in the absence of a true outcome. Please follow the document structure specified in the source file's comments.

4 Grading

Be professional. Make results easy to understand and grade. Include only those parts to be graded. Leave comments where necessary, especially if it aids in grading.

Each predicate is worth $\frac{1}{6}$ of the assignment grade.