CS 381, Spring 2024 — Homework 6 (Prolog)

Important Reminders!

1. Upload your solution as a single Prolog file `hw6.pl` to Canvas. **VERY IMPORTANT:** ***Use only lowercase letters for the file name!*** This file should contain the shown definitions of the `when`, `where`, and `enroll` predicates at the beginning. (It is probably a good idea to download the file `hw6.pl` and simply add your predicate definitions to it.)
   Also important: Do ***not*** include any extraneous entries for these predicates! If you want to test your predicate definitions with additional examples, please do this in a separate file. The file that you submit should have ***exactly*** the data shown in the exercise—no more, no less.
2. **Only submit files that compile without errors!** (Put all non-working parts in comments.
3. You must do all homework assignments by yourself, without the help of others. Also, you must not use services such as Chegg or Course Hero. If you need help, simply ask on Canvas, and we will help!
4. The homework is graded leniently, and we reward serious efforts, even when you can't get a correct solution.
5. You can work in teams of up to four students to create and submit homework assignments. Groups must be set up in advance and will have to be the same over the whole term. Each group submits one solution, and all group members receive the same grade for the homework.

Exercise 1. Database Application

Consider a database about classes, class times, classrooms, and student enrollment, given in the form of Prolog facts.

```
when(275,10).            where(275,owen102).          enroll(mary,275).
when(261,12).            where(261,dear118).          enroll(john,275).
when(381,11).            where(381,cov216).           enroll(mary,261).
when(398,12).            where(398,dear118).          enroll(john,381).
when(399,12).            where(399,cov216).           enroll(jim,399).
```

Define the following derived Prolog predicates by one or more rules. Note that the shown goals are just examples. You should define the predicates so that it is possible to formulate goals with variables or constants at any argument position. ***Hint***: The inequality of, say two variables `X` and `Y`, can be expressed using the subgoal `X \= Y`.

(a) Define a predicate `schedule/3` that gives for a student the classrooms and times of their taken classes, that is, if you evaluate the goal `schedule(mary,P,T)`, Prolog should give the following result.

```
?- schedule(mary,P,T).

P = owen102
T = 10 ;

P = dear118
T = 12 ;
```

As another application of the `schedule` predicate, consider the goal `schedule(S,cov216,T)` that shows all students that are in the classroom `cov216` together with the corresponding time.

```
?- schedule(S,cov216,T).

S = john
T = 11 ;

S = jim
T = 12 ;
```

(b) Define a predicate `usage/2` that gives for a classroom all the times it is used. For example, the goal `usage(cov216,T)` should yield the following result.

```
?- usage(cov216,T).

T = 11 ;

T = 12 ;
```

The goal `usage(X,11)` should list all classrooms that are used at `11`.

(c) Define a predicate `conflict/2` that can compute conflicts in the assignment of classes to classrooms. A conflict exists if two different classes are assigned to one classroom for the same time. The arguments of the `conflict` predicate are two class names. You can use the goal `conflict(275,X)` (or `conflict(X,275)`) to find out any classes that are in conflict with the class `275`.

```
?- conflict(275,X).
false.
```

The goal `conflict(X,Y)` determines all pairs of possible conflicts.

(d) Define a predicate `meet/2` that can determine pairs of students that can meet in a classroom by either attending the same class or by having classes that are back to back in one classroom. The last condition means that a student Jim can meet any student who has a class that is in the same classroom and immediately follows Jim's class. (Note that your definition of `meet` doesn't have to be symmetric, that is, if students A and B can meet, then your implementation has to return `true` for `meet(A,B)` or `meet(B,A)`, but not necessarily for both calls. You can ignore the case when students are enrolled in conflicting classes.)

Exercise 2. List Predicates and Arithmetic

Note: Do ***not*** use the predefined predicates `flatten` and `nth`. You are allowed to use predefined predicates, such as `append` or `member`.

(a) Define a Prolog predicate `rdup(L,M)` to remove duplicates from an ordered list `L`. The resulting list should be bound to `M`. Note that `M` must contain each element of `L` exactly once and in the same order as in `L`. You can assume that `L` is an ordered list.

(b) Define a Prolog predicate `flat(L,F)` that binds to `F` the flat list of all elements in `L` (where `L` can be a possibly nested list). For example, `flat([a,b,[c,d],[],[[[e]]],f],L)` should yield `L = [a,b,c,d,e,f]`.

(c) Define a Prolog predicate `project/3` that selects elements from a list by their position and collects them in a result list. For example, the goal `project([2,4,5],[a,b,c,d],L)` should produce the answer `L=[b,d]`. You can assume that the numbers in the first list are strictly increasing, that is, your implementation does not have to care about situations like `project([1,1,2],...)` or `project([2,4,3],...)`. You can also assume that the predicate is only used with a variable as third argument, that is, the list of positions and the list of elements are always given.