German University in Cairo
Media Engineering and Technology
Prof. Dr. Slim Abdennadher

Concepts of Programming Languages, Spring term 2016
Project Description
"Schedule My Events"

Every faculty at the GUC offers its students with different courses. Each course has a group of events associated to it. Such events include quizzes, assignments, project evaluations, ... etc. In this project you are required to implement a schedule for academic events for different study groups in the GUC.

The file `events.pl` posted on the MET website provides an example of a database of events to be scheduled. The data is organized using the following predicates:

a) `event_in_course(Course_Code, Event_Name, Event_Type)` encodes the fact that the course with code `Course_Code` has an event of type `Event_Type` named `Event_Name`. For example, the following fact is available in our database `event_in_course(csen403, quiz1, quiz).`

b) `studying(Course_Code, Group_Name)` provides the course catalogue. It encodes that the group `Group_Name` has to study `Course_Code` e.g. `studying(csen403, group4MET).`

c) `holiday(Week_Number,Day)` represents a holiday on the day: `Day` in the week numbered `Week_Number` e.g. `holiday(3,monday).`

d) `should_precede(Course_Code,Event1,Event2)` is available to make sure that the event named `Event1` should be scheduled before `Event2` for the course `Course_Code`
e.g. `should_precede(csen403,quiz1,quiz2).`

e) Every group has predefined slots in which academic events could be held. Such slots are encoded through the predicate `quizslot(Group_Name, Day, Slot_Number)`. For example `quizslot(group4MET, tuesday, 1)` encodes the fact that `group4MET` could have events on Tuesdays during the first slot. Each group could have more than one possible timing for events.

Properties of a Schedule

Assuming that there is an infinite number of locations, a produced schedule should have the following features:

1. No events should be scheduled on a holiday.
2. A group cannot have two events at the same time.
3. All events of any course are scheduled for any group taking the course.
4. Events of a group are scheduled only in the allowed slots.
5. No group can have two quizzes or assignments on the same day.
6. Two quizzes of one course should be separated by at least one weak.
7. The schedule generated should take into account that some courses have some events that should precede others.

Predicates to be added

The rest of the description will guide you through the predicates that you need to implement to be able to solve the scheduling problem.

precede/2

The predicate `precede(G,Schedule)` should succeed only if all of the `should_precede` facts of the courses studied by the group `G` are satisfied.

valid_slot/2 or valid_slots_schedule/2

You need to add **one** of these two predicates. The predicate to be added depends on how you decided to model and solve the problem. `valid_slot(G,Slot)` is true only if the group `G` does not have more than one event in `Slot`.
`valid_slots_schedule(G,Schedule)` is true only if the group `G` does not have more than one event in any slot of `Schedule`.

available_timings/2

`available_timings(G,L)` should only be true if `L` is the list of timings in which the group `G` could have an event scheduled.

group_events/2

`group_events(G,Events)` is true if `Events` is the list of events that should be scheduled for the group `G`.

no_consec_quizzes/2

`no_consec_quizzes(G,Schedule)` should succeed only if the group `G` does not have two quizzes for the same course in two consecitive weeks.

no_same_day_quiz/2

`no_same_day_quiz(G,Schedule)` is only true if group `G` does not have two quizzes scheduled on the same day in `Schedule`. According to your implementation, it could be replaced with `no_same_day_quiz(G,Week)` which succeeds if group `G` does not have two quizzes scheduled on the same day in `Week`.

no_same_day_assignment/2

Similar to `no_same_day_quiz/2` you should add `no_same_day_assignment(G,Schedule)` or `no_same_day_assignment(G,Week)` to make sure that group `G` does not have two assignments scheduled on the same day.

no_holidays/2

`no_holidays(G,Schedule)` should succeed only if `Schedule` has no events scheduled in any of the available holidays.

Scheduling

The main predicate the should activate the scheduling is `schedule(Week_Number, Schedule)` which should success if `Schedule` is the schedule of all of the available events satisfying all the constraints. `Week_Number` is the number of available weeks for the semester.

Note that you can use the accumulator technique or not. That is why in the previous section you were given alternative approaches for the predicates to be added.

Submission Guidelines

- The deadline of the project will be on **Saturday the 2nd of April**.
- You should submit the following form: `http://goo.gl/forms/pEuNPb0R18` with your team's information maximum by **Saturday 19/3/2016**. Each team should be of **2-3** members. If a team fails to submit this form by the 19th of March, they will be facing a zero. A list of teams with their corresponding generated *Team Name* will be posted on MET Website on Sunday 20/3/2016.
- Each team should submit a single *.pl* file, via the MET website submission link, containing the team's full project implementation. The submitted file **must** abide by the following rules:
  a) The file should be named according to your team's *Team Name* (from the to-be-posted teams list).
  b) You should include a clear documentation per implemented predicate. You should write each predicate's documentation above the predicate's implementation.
  c) Also, at the beginning of the file, you should provide the schedule representation that you have chosen as well as an example of it.
- It is the team's full responsibility to successfully submit a valid .pl file before the project's deadline.