Logic for Programming

Project

2022-2023

Timetables - Prolog searches

Contents

One fine day, you hear that the IST-Tagus room occupancy database has been attacked by a *hacker*. In an attempt to help, you study the data structures in question (Section 1), create a program in Prolog (first lines in Section 2) and solve the problem. You become a hero, but, as you know, with great powers come great responsibilities, and more requests for help start coming in (Section 3). Of course, you do your best! For the conditions under which the project was carried out, its evaluation and recommendations, see Sections 4 and 5.

1 Structures from data

There are two files - data.pl and keywords.pl - which are part of a freely modified version of a knowledge base kindly provided by the Academic and Building Management Area. The data.pl file contains facts about events, shifts associated with events and event times, defined as follows:

An event, `event(ID, NameCourse, Type, NumberStudents, Room)` is characterized by:

- An identifier;
- The name of the course associated with the event;
- The type of event (seminar, theoretical, etc.);
- The number of students associated with the event;
- The room where the event takes place.

An event has one or more shifts associated with it, `shift(ID, CourseSign, Year, ClassName)`, characterized by:

- An identifier (the ID of the associated event);
- The acronym of the course to which the event relates;
- The year in which the subject is offered on the course;
- The "name" of the shift.

An event also has a schedule associated with it, `horario(ID, DayWeek, StartTime, EndTime, Duration, Period)`, characterized by:

- An identifier (the ID of the associated event);
- The day of the week on which the event takes place;
- The start and end times of the event[1] ;
- The duration of the event (yes, it could be deducted from the previous figures);
- The period in which the event takes place.

By way of illustration, the following facts indicate that event 10 concerns a 'Digital Systems' laboratory, with 18 students and taking place in room 1-62. It takes place on Fridays between 8am and 10am, so it lasts two hours, and takes place on p2. This event is for the first year of LEE, classes lee0101 and lee0102.

[1] It is assumed that all events occur on the hour or hour and a half. Therefore, to make calculations easier, 12.5 will be used to indicate 12:30 (for example, by adding 1:30 to 12:30, we will be adding 1.5 to 12.5, giving us 14 (14h)).

```
event(10, 'digital systems', lab, 18, '1-62').
schedule(10, friday, 8.0, 10.0, 2.0, p2). shift(10,
lee, 1, lee0102).
shift(10, lee, 1, lee0101).
```

The "keywords.pl" file contains keywords that will be useful:

```
rooms(largeAmphitheaters, ['a1', 'a2']).
...
rooms(videoConf, ['0-19', '0-13']).
...
licenses(tagus,['lee', 'legi', 'leic-t', 'leti']).
masters(tagus,['mbmrp', 'mee', 'megi', 'meic-t', 'meti']).
```

2 The program at Prolog

The Prolog file (pl extension) that will be used in the project should have the following initial lines:

```
% Student number and name
:- set_prolog_flag(answer_write_options,[max_depth(0)]). % for complete lists
:- ['data.pl'], ['keywords.pl']. % files to import.

/*
Code
*/
```

3 Predicates to implement

3.1 Quality of data

The first request for help comes from the Secretariat: they ask you to help them find problematic events; in particular they ask you to identify/find them:

- Events without rooms;
- Events without rooms, given a day of the week;
- Events without rooms, given a period;

You roll up your sleeves and embrace the challenge with enthusiasm.

Knowing that events without rooms are identified by having the word 'withoutRoom' in the field relating to the room, implement the predicates `eventswithoutRooms/1`, `eventswithoutRoomsDayWeek/2` and `eventswithoutRoomsPeriod/2`, such that (respectively):

- `eventsWithoutRoom(EventsWithoutRoom)` is true if `EventsWithoutRoom` is a list, sorted and without repeated elements, of event IDs without a room;
- `eventsWithoutRoomsDayWeek(DayOfWeek, EventsWithoutRoom)` is true if `EventsWithoutRoom` is a list, sorted and without repeated elements, of IDs of events without rooms that take place on `DayOfWeek` (henceforth `Monday`, `Tuesday`, `Wednesday`, `Thursday`, `Friday`, `Saturday`);
- `eventsWithoutRoomPeriod(ListPeriods, EventsWithoutRoom)` is true if `ListPeriods` is a list of periods ($_{pi,i\in\{1,2,3,4\}}$) and `EventsWithoutRoom` is a list, sorted and without repeated elements, of IDs of events without rooms in the periods of `ListPeriods`. Events without rooms associated with semester courses (e.g. p1_2) should be accounted for. This is true for this predicate, but also for other predicates where information about periods is requested.

For example,

```
?- eventsWithoutSalas(Events).
Events = [14,88,191,311,312,342,343].
?- eventsSemSalasDiaSemana(Monday, Events). Events =
[191].
?- eventsWithoutPeriodSalas([p1], Events).
Events = [88,191,311,312,342,343].
?- eventsWithoutPeriodSalas([], Events).
Events = [].
```

In relation to the previous example, note that the third request is made on p1, but event 343 is returned, as it is an event without a semester room, which occurs in the first semester (p1_2), so it picks up p1:

```
event(343,'linear-algebra','theoretical-
practical',68,semSala).
schedule(343,thursday,8.0,10.0,2.0,p1_2).
```

3.2 Searches simple

You get a big thank-you from the Registrar for your excellent work and you're getting ready to go back to God of War/watch the latest episode of Arcane/Review Attack on Titan/Other (cross out what doesn't matter), when the Academic Area contacts you: they need help implementing a set of predicates. Once again, without a sigh, you roll up your sleeves and get back to work.

You start by implementing - **without using higher-order predicates, i.e. the predicates you define must use recursion (it doesn't matter if they generate recursive or iterative processes**) - the predicate `organizeEvents/3`, such that:

`organizeEvents(ListEvents, Period, EventsInPeriod)` é truth if `EventsInPeriod` is the list, sorted and without repeated elements, of event IDs of `ListEvents` that occur in the period `Period` for $_{pi,i\in\{1,2,3,4\}}$.

For example,

```
?- organizeEvents([23, 67, 89, 99, 6], p3, L).
L = [].
```

```
?- organizeEvents([23, 67, 89, 99, 6], p2, L).
L = [6,99].

?- organizeEvents([23, 67, 89, 99, 6], p1, L).
L = [23,67,89,99].
```

You also implement the predicate `eventsLessThan/2`, such that:

`eventsSmallerThan(Duration, ListEventsSmallerThan)` é truth if `ListMinorEventsWhich` is the sorted list with no repeated elements of the `events` identified in the list.
pain of events that have a duration less than or equal to `Duracao`.

For example:

```
?- eventsLessThan(0.5, ListEventsLessThan).
ListEventsLessThan = [4,7].
?- eventsSmallerThan(1.5, ListEventsSmallerThan).
ListEventsLessThan = [3,4,5,7,...,787,796].
```

Next, you implement the `eventsSmallerThanBool/2` predicate, such that:

`eventsLessThanBool(ID, Duration)` is true if the event identified by `ID` has a duration equal to or less than `Duration`.

```
?- eventsSmallerThanBool(45, 0.5).
false.
?- eventsMinorQueBool(4, 0.5). true.
```

You also implement the `searchDisciplines/2` predicate, such that:

`searchCourses(Course, ListCourses)` is true if `ListCourses` is the alphabetically ordered list of the names of the courses in the course `Course`.

For example[2] ,

```
?- searchCourses(leti, ListCourses). ListDisciplines
= [linear algebra,
    data analysis and statistical modeling, network architecture,
    differential and integral calculus i, differential and integral
    calculus iii, electromagnetism and optics, software engineering,
    fundamentals of programming, management,
    introduction to economics,
    introduction to telecommunications and computer engineering,
    introduction to electronic circuits and systems,
    mechanics and waves, programming with objects,
    propagation and antennas, communication systems,
    digital systems, operating systems].
```

Next, you implement - again **without using higher-order predicates, i.e. the predicates you define must use recursion (it doesn't matter if they generate recursive or iterative processes**) - the predicate `organizeDisciplines/3`, such that:

[2] For the sake of readability, a predicate is sometimes split into two lines. Also note that the knowledge base is not complete, so it is normal for subjects to be missing (example: calculus II).

`organizesCourses(CourseList, Course, Semesters)` is true if `Semesters` is a list with two lists. The list in the first position contains the courses in the `CourseDisciplines list` that take place in the first semester; the same goes for the list in the second position, which contains those that take place in the second semester. Both lists must be sorted alphabetically and have no repeated elements. The predicate fails if there is no course in `CourseList`. It can be assumed that there are no annual courses.

For example[3] ,

```
?- organizeDisciplines(['linear algebra','compilers'], 'leic-t', L). L
= [[linear algebra],[compilers]].
?- organizaDisciplinas(['linear algebra','business analytics', 'project
   evaluation', 'materials science'], legi, L).
L = [[linear algebra,business analytics,project evaluation], [materials
    science]].
?- organizaDisciplinas(['linear algebra','business analytics', 'project
   evaluation', 'materials science'], 'leic-t', L). false.
```

You then attack the predicate `hoursCourse/5`, such that:

`hoursCourse(Period, Course, Year, TotalHours)` is true if `TotalHours` is the total number of hours of events associated with the course `Course`, in the year `Year` and period `Period` = $_{pi,i\in\{1,2,3,4\}}$. Once again: don't forget semester courses.

Note that if several shifts share the same event, the number of hours of the event should only be counted once. For example, in the following case, only 2 hours should be counted:

```
event(78,'differential and integral calculus i','theoretical-
practical',86,a1). schedule(78,wednesday,8.0,10.0,2.0,p1_2).
shift(78,leti,1,leti0103).
shift(78,leti,1,leti0102).
shift(78,leti,1,leti0101).
```

For example,

```
?- hoursCourse(p1, leic-t', 1, TotalHours).
TotalHours = 50.0.
```

Finally, you implement the `evolutionHoursCourse/2` predicate, such that:

`evolucaoHorasCurso(Curso, Evolucao)` is true if `Evolucao` is a list of tuples in the form (`Year`, `Period`, `NumHoras`), where `NumHoras` is the total number of hours associated with the course `Curso`, in the year `Ano` and period `Periodo` ($_{pi,i\in\{1,2,3,4\}}$). `Evolution` should be sorted by year (ascending) and period.

Tip: use the previous predicate. For

example,

[3] Note that leic-t has to take plicas. Also note that if you copy&paste the Prolog objectives from the pdf, you'll get a syntax error in Prolog.

```
?- evolutionCourseHours('leic-t', Evolution).
Evolution = [(1,p1,50.0),(1,p2,59.0),(1,p3,0),(1,p4,0),
(2,p1,47.0),(2,p2,77.0),(2,p3,0),(2,p4,20.0),
(3,p1,32.0),(3,p2,32.0),(3,p3,39.0),(3,p4,19.0)].
```

3.3 Critical occupancies of rooms

The Academic Area has loved you forever! You take a deep breath and get ready to play LOL/Rocket League/COD/Manic Miner/Minecraft/Other (cross out what doesn't matter), but it's not yet time! The Building Management team needs your help: some types of rooms - for example, amphitheaters - are heavily occupied and we need to identify which ones and when. They're asking you to implement a set of predicates that will allow them to calculate the occupancy percentages of the various room types, with critical occupancies being those that exceed a given value (*threshold*). Since it's already cold, don't roll up your sleeves. In fact, you can't help but let out a little sigh. But then a surge of energy goes through you and off you go, to help the Building Management team: LET'S GO!!!!!!!!!!!!!

An event has a start time and an end time. If a *slot* is given with a start time and an end time, it may or may not fall entirely or partially on the event. So, you implement the predicate `occupySlot/5`, such that:

`occupySlot(StartEventTime, EndEventTime, StartEventTime, EndEventTime, Hours)` is true if `Hours` is the number of overlapping hours (remember that 0.5 represents 30 minutes) between the event that starts at `StartEventTime` and ends at `EndEventTime`, and the *slot* that starts at StartEventTime and ends at `EndEventTime`. If there are no overlaps, the predicate must fail (false).

The following example illustrates four scenarios (in the first, the event is completely contained in the *slot*; in the second, the event completely contains the *slot*; in the third, the overlap is at the beginning of the event; in the fourth, the overlap is at the end of the event).

```
?- occupySlot(8.5, 11, 9, 10.5, Hours).
Hours = 1.5.
?- occupySlot(9.5, 10, 9, 10.5, Hours).
Hours = 0.5.
?- occupySlot(8.5, 9.5, 9, 10.5, Hours).
Hours = 0.5.
?- occupySlot(10, 11, 9, 10.5, Hours).
Hours = 0.5.
?- occupySlot(10, 11, 8, 9, Hours).
false.
```

Next, implement the predicate `numHorasOcupadas/6`[4], such that:

`numHorasOcupadas(Periodo, TipoSala, DiaSemana, HoraInicio, HoraFim, SomaHoras)` is true if `SomaHoras` is the number of hours occupied in the rooms of type `TipoSala`, in the time interval defined between `HoraInicio` and `HoraFim`, on the day of the week `DiaSemana`, and in the period `Periodo` = $_{pi,i\in\{1,2,3,4\}}$. Don't forget semester subjects.

For example,

[4] The possible room types are the first arguments of the `room` predicate in the keywords.pl file (example: largeAmphitheatres, labsElectro).

```
?- numHoursOccupied(p1, largeAmphitheaters, wednesday, 8.0, 12.0, S). S
= 6.0.
numOccupiedHours(p1, largeAmphitheaters, wednesday, 8.0, 10.0, S).
S = 2.5.
```

Regarding the last example, he notes that there is a 'Fundamentals of Programming' class in a2 (one of the large amphitheaters) that takes up only 30 minutes of the 8.0-10.0 slot (event 78), to which is added the two hours of the 'Differential and Integral Calculus I' class (event 566), in a2.

You also implement the `occupationMax/5` predicate, such that:

`occupationMax(RoomType, StartTime, EndTime, Max)` is true if `Max` is the number of possible hours to be occupied by rooms of `RoomType` (see above), in the time interval defined between `StartTime` and `EndTime`. In practical terms, `Max is` assumed to be the given time interval (`EndTime - StartTime`), multiplied by the number of rooms in play of the `RoomType` type.

For example (since there are two large amphitheaters),

```
?- occupancyMax(largeAmphitheaters, 8, 12.5, Max).
Max = 9.0.
```

Then you implement the `percentage/3` predicate, such that:

`percent(SumHours, Max, Percent)` is true if `Percent` is the division of `SumHours` by `Max`, multiplied by 100.

For example,

```
?- percentage(5, 9, Percentage). Percentage =
55.55555555555556.
```

Finally, you implement the `occupationCritical/4` predicate, such that:

`occupancyCritical(StartTime, EndTime, Threshold, Results)` is true if `Results` is an ordered list of tuples of type `casesCritical(WeekDay, RoomType, Percentage)` where `WeekDay`, `RoomType` and `Percentage` are, respectively, a day of the week, a room type and its occupancy percentage, in the time interval between `StartTime` and `EndTime`, and assuming that the occupancy percentage relative to these elements is above a given critical value (`Threshold`). In the tuple representation, it uses the `ceiling` predicate to round the percentage value to the next integer, i.e. `Percentage` must be the first largest integer relative to the percentage value used in the calculations (but only in the tuple representation; in the calculations it must use the percentage value without any rounding).

For example,

```
?- occupationCritical(8, 12.5, 85, Results).
Results = [casesCritical(Monday,largeAmphitheatres,89),
      casesCritical(Monday,largeAmphitheatres,95),
      casesCritical(Monday,smallAmphitheatres,93),
      casesCritical(Friday,labsChemistry,89)].
```

3.4 And now for something completely different...

After receiving countless compliments and thanks from Building Management, you arrive home. You think this is the day you're going to see "Wednesday/Enola Holmes2/Umbrella Academy/The Boys/Altered Carbon/Other (delete as appropriate). However, just as you're opening the door to the street, your next-door neighbor, Maria by the name of Maria, appears:

– Young man," she begins, "I've heard that you're a programming deity and I need your help to organize my family for Christmas dinner. We have a table of 8 people and there will be 8 of us. My João and I occupy the two headboards, but I have to stay at the headboard closest to the fireplace because I'm very cold. Aunt Guga, who is almost 100 years old, comes and has to stand to my João's right. Then my daughter Ana has to stand next to my grandson Manelito, who is only 3 years old, and my son Miguel has to stand next to Pedrito. My son-in-law Jorge gets on very well with Miguel and I'd like them to be face to face at the table. Do you think there's a solution to this? Oh, I forgot to say that it's very important that Manelito and Pedrito don't stand exactly opposite each other and end up throwing potatoes and peas at each other.

You swallow, remembering that when you go on vacation, this lady takes care of Darwin, your little orange fish, and respond with your best smile:

– Right, I'll get you a program that checks all those requirements.

You go into the house thinking that the best thing to do is to implement something generic, lest the neighbor come and ask you for solutions every time she gives a dinner party. However, you decide to assume (Figure 1) that: a) the dining table is rectangular, with 8 seats in total, one seat at each head and 3 at each side, with the headboards differentiated; b) there will be exactly 8 guests.

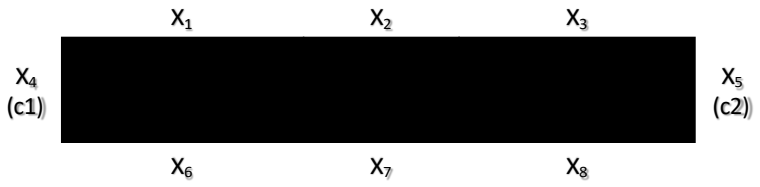

Figure 1: Table design

You then decide to implement the `occupationTable/3` predicate, such that:

`occupationTable(ListPeople, ListRestrictions, OccupationTable)` é truth if `ListPeople` is the list with the names of the people to be seated at the table, `ListRestrictions` is the list of restrictions to be checked (see below) and `OccupationTable` is a list with three lists, the first of which contains the people on one side of the table (X_1 , X_2 and X_3), the second the people at the head of the table (X_4 and X_5) and the third the people on the other side of the table (X_6 , X_7 and X_8), so that these people are exactly those in `ListPeople` and check all the restrictions in `ListRestrictions`. You can assume that there will be one and only one solution.

Assume that the possible constraints are (examples in Figure 1):

- `cab1(NamePerson)`: true if `NamePerson` is the person in bed 1 (the one near the fireplace) - X_4 ;
- `cab2(NamePerson)`: true if `NamePerson` is the person at head 2 - X_5 ;
- `honor(NamePerson1, NamePerson2)`: true if `NamePerson1` is in one of the headers.

and `NamePerson2` to your right - X_3 or X_6 , depending on which headboard is occupied;

- `side(NamePerson1, NamePerson2)`: true if `NamePerson1` and `NamePerson2` are side by side in the table[5] - for example, X_7 and X_8 ;
- `notSide(NamePerson1, NamePerson2)`: true if `NamePerson1` and `NamePerson2` are not side by side in the table - for example, X_1 and X_3 ;
- `front(NamePerson1, NamePerson2)`: true if `NamePerson1` and `NamePerson2` are exactly opposite each other in the table[6] - for example, X_7 and X_2 ;
- `notFront(NamePerson1, NamePerson2)`: true if `NamePerson1` and `NamePerson2` do not face each other across the table - for example, X_7 and X_3 .

For example,

```
?- occupationTable([maria, joao, pedrito, jorge, ana, manelito, miguel, guga],
   [cab1(maria), cab2(joao), honor(joao, guga), side(ana, manelito),
   side(miguel, pedrito), front(miguel, jorge),
   nonFrente(pedrito, manelito)], L).
   L = [[miguel,pedrito,guga],[maria,joao],[jorge,ana,manelito]] ;
   false.
?- occupationTable([a, b, c, d, e, f, g, h], [cab1(e), honor(e, b), nonFront(a, b),
   side(f, g), side(a, c), nonSide(f, c), nonFront(f, c), front(g, d)], L). L
   = [[c,a,d],[e,h],[b,f,g]] ;
   false.
?- occupationTable([a, b, c, d, e, f, g, h], [cab1(e), honor(e, b), cab2(c),
   honor(c, a), nonFront(a, b), nonSide(b, f), side(f, g), front(b, h)], L). L
   = [[h,d,a],[e,c],[b,g,f]];
   false.
```

Figure 2 illustrates the solution for Maria's supper.

Figure 2: Neighbor Maria's Christmas dinner solution

You also remember that with Prolog you can generate and test solutions. You just have to make sure it doesn't explode in memory. You think you might have to use functors. Or maybe not. Hmm...

When you've finished, you go to tell your neighbor about the solution and she's delighted (later she'll bring you some "filhoses" as a thank you). You return home and turn off your cell phone, *just in* case....

[5] You don't think that those at the head of the field are "next to" anyone.
[6] You consider that the person at the head of the table is not in front of the person at the other end of the table.

4 Delivery and evaluation

4.1 Conditions and deadlines

The project is carried out individually. The project code must be submitted electronically by **11:59 p.m. on January 13, 2023,** via the Mooshak system. Projects will not be accepted after this time under any circumstances[7] . Please note:

- You must submit a .pl file containing the code for your project. The code file must contain the number and name of the student in a comment on the first line;
- No accented characters or any character that does not belong to the ASCII table should be used, even in comments;
- Don't forget to remove/comment on messages written on the screen;
- The evaluation of the execution of the project's code will be done automatically through the Mooshak system, using various tests configured in the system. The execution time of each test is limited, as is the memory used. Typically, you can only make a new submission 15 minutes after the previous submission[8] . Only 10 submissions are allowed simultaneously in the system, so a submission may be rejected if this limit is exceeded. In that case, try again later;
- The tests considered for evaluation may or may not include the examples provided, as well as a set of additional tests. It should be noted that the knowledge base used in the tests is an extension of the one given.

The necessary instructions for submitting the code to Mooshak will be published on the chair's page and electronic submission will be possible from then on. Until the submission deadline, you can make as many submissions as you wish (please do not use Mooshak for debugging), and the last submission will be used for evaluation purposes.

There may or may not be an oral discussion of the project and/or a demonstration of how the program works (this will be decided on a case-by-case basis).

4.2 Quotation

An extended version of the data.pl file will be used in the assessment. The project grade will be based on the following:

- Correct execution - 16 values distributed as follows:
  1. Data quality (2.0 values)
     (a) `eventsWithoutHalls`: 0.5 values;
     (b) `eventsWithoutHallsDayWeek`: 0.5 values;
     (c) `eventsWithoutHallsPeriod`: 1.0 values;
  2. Simple search (7.5 values)

[7] Please note that the limit of 10 simultaneous submissions in the Mooshak system means that if there are a high number of submission attempts over the deadline, some students may find it impossible to submit the code on time.

[8] Please note that if you make a submission to Mooshak less than 15 minutes before the deadline, you will not be able to to make any `further` submissions.

   (a) `organizesEvents`: 1 value
   (b) `eventsLessThan`: 0.75 values;
   (c) `eventsLessThanBool`: 0.75 values;
   (d) `searchDisciplines`: 1 value;
   (e) `organizesDisciplines`: 1.5 values;
   (f) `hoursCourse`: 1.5 points;
   (g) `evolutionCourseHours`: 1 value;
3. Critical room occupancy (4.5 points)
   (a) `occupationSlot`: 1.5 values;
   (b) `number` of `hours worked`: 1.0;
   (c) `occupancyMax`: 0.75 values;
   (d) percentage: 0.25;
   (e) `occupationCritical`: 1.0 value;
4. And now for something completely different... (2 values)
   (a) `occupationTable`: 2.0 values.

- Programming style and ease of reading - 4 values distributed as follows:
  - Comments (1.0 value): should include comments for the user (summary description of the predicate); should also include, where appropriate, comments for the programmer.
  - Good practices (3.0 points):
    * Integration of knowledge acquired during the course (1.0 value).
    * Implementation of predicates that are not excessively long. The fact of using a recursive predicate, provided it is done well, is not penalized in relation to the use of functional predicates; however, poor procedural abstraction and code duplications will be penalized (1.0 value).
    * Choice of names for auxiliary predicates and variables (1.0 value).

The presence of *warnings* will be penalized (-2 points). Predicates 4 and 8, if they do not respect the instructions given in their implementation, will be penalized by 1 and 1.5 points respectively.

4.3 Copies

Very similar projects will lead to failure in the subject and disciplinary proceedings. The subject faculty will be the sole judge of what is considered cheating.

5 Recommendations

- We recommend SWI PROLOG, which will be used to evaluate the project.
- When developing the program, don't forget Murphy's Law:
  - All problems are harder than they seem;
  - Everything takes longer than we think;
  - If anything can go wrong, it will go wrong at the worst possible time.