Prolog HW: CS101a/Aut08

Tim Hickey
due Tue 11/18

November 10, 2008

1 Overview

The goal of this assignment is to give you the opportunity to learn how to use Logic Programming (and Prolog in particular) to solve some simple natural language and scheduling problems.

You will be given a program that allows users to ask questions about the current classes at Brandeis for Spr09. It translates those questions into an internal representation using DCGs and then answers the questions by querying a database consisting of all courses for next year that have been assigned a block. Actually, some classes (science labs in particular), do not fit nicely into the block system and so don't appear in this database.

2 Prolog resources

You should use SWI Prolog for this assignment. You can download it for free from `http://www.swi-prolog.org/` This is a professional quality Prolog implementation distributed under an open source license and should run on most Mac, Linux, and Windows platforms. The SWI website also contains a Prolog manual `http://gollem.science.uva.nl/SWI-Prolog/Manual/`

3 The course knowledge base

The main predicate in this prolog program is a collection of facts (stored in the file "current.pl") about the courses offered at Brandeis for the Spr09 semester. Here are a few facts from the DB

```
course(3212,'cosi',2,'a',1,'Introduction to Computers','f','Hickey','Timothy J').
course(4080,'cosi',113,'b',1,'Machine Learning','p','Pollack','Jordan').
course(4081,'cosi',128,'a',1,'Modern Database Systems','j','Cherniack','Mitch').
course(4501,'econ',57,'a',1,'Environmental Economics','b','Bui','Linda').
```

The course predicate has 9 fields as follows:

```
course(ID, Dept, Num, Suffix, Section, Title, Block, LastName, FirstName).
```

where the ID is a number uniquely identifying that course among all courses for Spr09. The Dept is a 3 or 4 letter abbreviation for the department, the Num is the course number, the suffix is one of the standard suffixes (a,b or d), The Title is the official course title, and LastName/FirstName are the name of the instructor.

This database only contains classes which are scheduled for standard blocks (and so doesn't contain any of the Science lab courses...)

3.1 Accessors and Setof

. The first step is to write some accessors which allow one to get the fields for a particular course quickly, e.g. to find the suffix for a particular course num we can use the following predicate:

```
course_suffix(ID,Suffix) :-
  course(ID,_Dept,_Num,Suffix,_Sec,_Title,_Block,_LN,_FN).
```

Note that the underscores at the beginning of variable names are indicators to the Prolog compiler that you, the author, know that the variable `_Dept` only appears once in the procedure and hence is not being used to pass values between the head of the procedure and the predicates in the body. If you don't add the underscores the compiler will complain.

You can then find the set of all suffixes in the database using the following query:

```
?- setof(S,C^course_suffix(C,S),L),length(L,N).
L = [a, b],
N = 2.
?-
```

The `setof(T,Q,L)` operator generates a list `L` of terms which are instantiations of `T` obtained by solving the query `Q`. The `C^` prefix of the query `C^course_suffix(C,S)` is an existential quantifier that causes the value of `C` to be reset after each solution of the query is found. If you don't do this, then it will return a different list `L` for each possible value of `C`,

3.2 Problem 1

. Write accessors for all of the other fields and use a setof query it to determine how many different values there are for each field.

4 Scheduling

From the description of the blocks, it is easy to create a predicate which is true if two blocks conflict. It starts off like this

```
% Here are all the conflicting blocks....
conflicts(X,X). % any course conflicts with itself!
conflicts(k,s1).
conflicts(l,s1).
conflicts(s1,k).
conflicts(s1,l).
conflicts(k,s3).
conflicts(l,s3).
conflicts(k,s3).
conflicts(l,s3).
% and so on...
```

We can use this to write a predicate which is true if two courses do not conflict:

```
courses_no_conflict(N1,N2) :-
  course_block(N1,B1),course_block(N2,B2),not(conflicts(B1,B2)).
```

This can be generalized to show that a course N1 does not conflict with any of a list `L` of courses

```
course_doesnot_conflict(N1,[]).
course_doesnot_conflict(N1,[N|Ns]) :-
  courses_no_conflict(N1,N),
  course_doesnot_conflict(N1,Ns).
```

and finally, that a list of courses in non-conflicting ..

```
no_conflicts([]).
no_conflicts([B|Bs]) :-
  course_doesnot_conflict(B,Bs),
  no_conflicts(Bs).
```

We will be working with lists of class ids and so it is nice to have a procedure which will printout all of the class information for those classes in a nice form. Here is one way to do this ..

```
% writeclasses(+L) where L is a list of class ID numbers
writeclasses([]).
writeclasses([A|As]) :-
  course(A,Dept,Num,Suf,Sec,Title,Block,Last,First),
  block(Block,Time),  write(Time),write(': '),
  write(course(A,Dept,Num,Suf,Sec,Title,Block,Last,First)),nl,
  writeclasses(As).
```

We can use this to help create schedules as follows. If you want to take any four courses, the following query will work:

```
?- L=[A,B,C,D], no_conflicts(L),writeclasses(L).
```

but if we wanted the first two classes to be COSI classes numbered 100 or more the third to be an econ, and the fourth to be an english or history class, we could modify the query as follows:

```
?- L=[A,B,C,D],
   course_dept(A,cosi),course_num(A,N1),N1>100,
   course_dept(B,cosi),course_num(B,N2),N2>100,
   course_dept(C,econ),
   member(X,[eng,hist]),course_dept(D,X),
   no_conflicts(L),writeclasses(L).
```

Even better is to store this query as a procedure which can be loaded into the interpreter and evaluated with a simple call.

```
schedule1(L) :-
   L=[A,B,C,D],
   course_dept(A,cosi),course_num(A,N1),N1>100,
   course_dept(B,cosi),course_num(B,N2),N2>100,
   course_dept(C,econ),
   member(X,[eng,hist]),course_dept(D,X),
   no_conflicts(L),writeclasses(L).
```

4.1 Problem 2

Sometimes students want to keep an entire day free. The hw4code.pl program has a predicate, course_free_on_day(courseID,day), which is true if the given block does not meet on the given day. Use this to create a procedure `schedule2` by modifying `schedule1` above to find a schedule that does not meet on Monday. How many such schedules are possible? (use setof and length to figure this out).

5 A Natural Language Front End

We can now create a simple natural language front end that will let the user make English-like queries to the database. The top level will look like this:

```
interact :-
  write_prompt,
  read_request(UserRequest),
  write(UserRequest),nl,
  question(Tree, UserRequest, []),
  write(Tree),nl,
```

```
  findanswers(Tree,Answer),
  writeclasses(Answer),nl.
```

where `question(T,U,[])` is implemented using DCGs. Here is an initial version of the DCGs which accept queries asking about courses in particular departments, blocks, by particular instructors, etc.

```
question(show_courses(Q)) --> [show,courses],qualifiers(Q).

qualifiers([]) --> [].
qualifiers([dept(Ds)|Qs]) --> [in],depts(Ds),qualifiers(Qs).
qualifiers([block(B)|Qs]) --> [in,block],blocks(B),qualifiers(Qs).
qualifiers([notblock(B)|Qs]) --> [not,in,block],blocks(B),qualifiers(Qs).
qualifiers([instr(I)|Qs]) --> [taught,by],[I],qualifiers(Qs).
qualifiers([not_onday(Ds)|Qs]) --> [not,meeting,on],days(Ds),qualifiers(Qs).

depts([D]) --> [D].
depts([D|Ds]) --> [D],[or],depts(Ds).

days([D]) --> day(D).
days([D|Ds]) --> day(D),[or],days(Ds).


question(show_courses(Q)) --> [show,courses],qualifiers(Q).

qualifiers([]) --> [].
qualifiers([dept(Ds)|Qs]) --> [in],depts(Ds),qualifiers(Qs).
qualifiers([block(B)|Qs]) --> [in,block],[B],qualifiers(Qs).
qualifiers([notblock(B)|Qs]) --> [not,in,block],[B],qualifiers(Qs).
qualifiers([instr(I)|Qs]) --> [taught,by],[I],qualifiers(Qs).
qualifiers([not_onday(Ds)|Qs]) --> [not,meeting,on],days(Ds),qualifiers(Qs).

depts([D]) --> [D].
depts([D|Ds]) --> [D],[or],depts(Ds).

days([D]) --> [D].
days([D|Ds]) --> day(D),[or],days(Ds).

% The curly braces allow you to call a prolog procedure without absorbing any tokens....
% the day procedure matches any single token which is in the list of five weekdays
day(X) --> [X],{member(X,[mon,tue,wed,thu,fri])}.

blocks([B]) --> block_id(B).
blocks([B|Bs]) --> block_id(B), [or], blocks(Bs).
```

```
block_id(B) --> [B],{block_time(B,_)}.
```

This generates a list of constraints on courses which is sent to the findanswers predicate. We can define it as below to provide a list of courses that meet all of these requirements.

```
findanswers(show_courses(Qs),L) :-
  setof(N,satisfies(N,Qs),L).

satisfies(_,[]).
satisfies(N,[Q|Qs]) :-
  satisfies(N,Q),
  satisfies(N,Qs).
satisfies(N,dept([D|_])) :- course_dept(N,D).
satisfies(N,dept([_|Ds])) :- satisfies(N,dept(Ds)).
satisfies(N,not_onday(Ds)) :- course_free_on_day(N,Ds).
```

Here is an example of its use ... The first three lines after the query are debugging lines showing intermediate values.

```
?- interact.
>> show courses in anth or fa not meeting on tue

MWT 10-11   : course(1911, fa, 18, b, 1, History of Art II: From the Renaissance to the Mod
MWT 10-11   : course(4026, anth, 157, a, 1, Families and Households, c, Jacobson, David)
MWT 11-12   : course(4024, anth, 132, b, 1, Representing Ethnography, d, Jacobson, David)
MWT 11-12   : course(4632, fa, 71, a, 1, Modern Art and Modern Culture, d, Kalb, Peter R)
MWT 12-1    : course(1051, anth, 5, a, 1, Human Origins, e, Urcid, Javier)
MWT 1-2     : course(4631, fa, 45, a, 1, St. Peter's and the Vatican, f, McClendon, Charles
MW 3:30-5   : course(4108, fa, 39, b, 1, Islamic Art and Architecture, l, Karimi, Zahra Pam
MW 3:30-5   : course(4396, anth, 33, b, 1, Crossing Cultural Boundaries, l, Parmentier, Ric
MW 5-6:30   : course(4110, fa, 58, b, 1, High and Late Renaissance in Italy, m, Unglaub, Jo
MW 5-6:30   : course(4397, anth, 114, b, 1, Verbal Art and Cultural Performance, m, Parment
MW 5-6:30   : course(4823, anth, 116, a, 1, Human Osteology, m, Urcid, Javier)
true

?-
```

5.1 Problem 3

Extend the parser so that it allows users to specify course number constraints

```
show courses in cosi or math above 100
show courses in cosi below 40
show courses in cosi above 100 below 120
```

This will require modifying the DCGs and also the `satisfies` predicate so that it will solve these new constraints.

5.2 Problem 4

Some students like to avoid morning classes or evening classes. Modify the program to allow the user to specify morning, afternoon, or evening classes. For example,

```
show afternoon courses in cosi not meeting on fri
show morning and afternoon courses in cosi numbered < 40
```

Look at how the code for `not meeting on ...` was implemented and model your code after that.

6 How to submit

You should submit a complete program `hw4.pl` which successfully solves problems 1-5. Also, submit a transcript of a Prolog session that shows your program working with a wide variety of queries. If you can't get part of the program working just say so. You can use the `script` command in Unix to create a transcript of a session. Email both the homework and the script to tjhickey@brandeis.edu with the subject line `HW4 CS101`.

You should also include a paragraph explaining who you talked to about this assignment and what resources you used (websites, books, etc.).

7 Making change at Brandeis!

If anyone is interested in pursuing this further we could look into creating a web-based application for use next semester by Brandeis students picking courses for Fall09. The real-life system would help generate class schedules taking into account more information (e.g. prerequisites, the classes the student has already taken, non-western courses, quantitative reasoning courses, major/minor information on core courses and other course requirements, courses that don't fit into the block system such as lab courses, etc.). Send email to tjhickey@brandeis.edu if you're interested in pursuing this further.