

May 6, 2003

Computational Logic- Project

This file belongs to **Ramakrishnan Venkitaraman** (*rxv024000@utdallas.edu*)

Problem: *Figuring out when a graduate student in Computer Science will be able to graduate. This program could work in 3 ways: (i) Given a schedule it should be able to tell whether the student is eligible to graduate; (ii) Given a partial schedule it should be able to tell all possible ways in which a student can graduate (iii) Tell a new student (who has decided his/her major) all possible schedules that will lead to graduation.*

Solution:

Modeling the problem

Prolog was chosen to implement the solution as the problem can be essentially modeled as a search problem in a huge database that's given to us. What we essentially have is a database which denotes the set of courses that the student has taken in the case of an existing student and the set of courses that the student can take in the case of a new student. This database in our program is represented in the form of prolog facts.

Design and Implementation

In the implementation the user has to give as input to the system the unique identification number of the student who is to be advised regarding graduation and a variable say "L" (list) that will hold the output after the computation.

The system takes this input and then searches the database and first finds out the department to which this student belongs to. Then it scans through the database and gathers information regarding the set of courses he has taken. Then it compares the number of courses/credit hours that he has already taken to the number of credit hrs that he needs to take to graduate.

After the check has been made and if the system finds out that the student has taken all the courses he needs to graduate and has satisfied the core courses, independent study and other requirements it says "YES" meaning that the student can graduate.

After the check has been made if it finds out that the student needs to take more courses to graduate then it goes to the database that denotes the courses that this student can take and then compares it with the courses that he has already taken and then lists the courses that he can take to graduate. This listing which advises the student about the courses he can take gives a list. The number of elements in the list corresponds to the number of courses the student needs to take to graduate. The system first gives the initial list and if the user is not satisfied with the answer that prolog gave, then he can type ";" and the system will give another answer if possible until he is satisfied. If the user is satisfied with the answer he can just type "Enter key".

For now I have done the project by assuming certain things like for the courses that the system advises that the student take it assumes that all the courses are for 3 credit hrs. The generic case is a straight forward extension of this case. This just needs the addition of more facts. The system can be extended to support more majors and more choices in courses just by adding more facts and no addition to code is necessary. CLP can be used to make the search more effective. I chose prolog as SWI Prolog (Free!) does not support CLP.

Source Code

The prolog code that was used to implement the system is given below.

```
% Database for students and graduation info -Expert System

/* Syntax: grad(SSN) to check if the student can graduate or not */

%student (name, ssn, address, gpa)

student('st1',1000,'street1, city1, state1, zip1',3.75).
student('st2',1001,'street2, city2, state2, zip2',4.0).

%coursetaken(ssn, course#, sem, year, grade)

coursetaken(1000,no1,fall,2002,'A').
coursetaken(1000,no2,fall,2002,'B').
coursetaken(1000,no3,fall,2002,'A').

coursetaken(1000,in1,fall,2002,'A').
coursetaken(1000,in2,fall,2002,'A').
coursetaken(1000,in3,fall,2002,'A').
coursetaken(1000,no7,fall,2002,'A').

coursetaken(1000,c1,spring,2002,'A').
coursetaken(1000,c2,spring,2002,'A').
coursetaken(1000,c3,spring,2002,'A').

coursetaken(1001,no8,fall,2002,'A').
coursetaken(1001,no9,fall,2002,'B').
coursetaken(1001,no10,fall,2002,'A').
coursetaken(1001,no4,fall,2002,'A').
coursetaken(1001,no5,fall,2002,'A').
coursetaken(1001,no6,fall,2002,'B').

%course(course#, coursename,type,creditHrs).. type = core or elective or
thesis or independent

course(c1,_,core,3).
course(c2,_,core,3).
course(c3,_,core,3).

course(no1,cs1,core,6).
course(no2,cs2,elective,3).
course(no3,cs3,thesis,6).

course(no7,cocs7,core,3).
```

```

course(no8,cocs8,core,3).
course(no9,cocs9,core,3).
course(no10,cocs10,core,3).

course(in1,inn1,independent,3).
course(in2,inn2,independent,3).
%course(in3,inn3,independent,3).

course(no4,cs4,core,3).
course(no5,cs5,elective,3).
course(no6,cs6,thesis,3).

%thesis(ssn#) means that the person with the particular ssn# is in the
thesis option

thesis(1000).

dept(1000,cs).
dept(1001,cs).
dept(1002,cs).

%=====

%list_courses(ssn#, outputList) -Output list contains the list of
courses that are taken by ssn#

list_courses(Sid, L):-list_courses(Sid, [],L).
list_courses(Sid, Acc, L):-coursetaken(Sid,Cno,_,_,_),
\+member(Cno,Acc), append([Cno],Acc,L1), list_courses(Sid,L1, L), !.
list_courses(_, Acc,Acc).

%collect_hrs(Sid,N):- N is the list of the courses that the student
with Sid has already taken
collect_hrs(Sid,N):-collect_hrs(Sid, [],N).
collect_hrs(Sid,L,N):-coursetaken(Sid,Cno,_,_,_),
\+member(Cno,L), append([Cno],L,L1), collect_hrs(Sid,L1,N), !.
collect_hrs(_, Acc,Acc).

%sum_hrs(ssn#,result) -result contains the sum of the credit hrs of the
courses taken by ssn#

sum_hrs(Sid,N):-sum_hrs(Sid,0, [],N).
sum_hrs(Sid,Acc,L,N):-
coursetaken(Sid,Cno,_,_,_), course(Cno,_,_,Hrs), \+member(Cno,L), append([
Cno],L,L1), A is Acc+Hrs, sum_hrs(Sid,A,L1,N), !.
sum_hrs(_, Acc,_, Acc).

%sum_hrs_type(ssn#,type, result) -result contains the sum of the credit
hrs of the courses taken by ssn# of the gvn type = thesis, independent,
core, elective

sum_hrs_type(Sid,Ty,N):-sum_hrs_type(Sid,0, [],Ty,N,Li).
sum_hrs_type(Sid,Acc,L,Ty,N,Li):-
coursetaken(Sid,Cno,_,_,_), course(Cno,_,Ty,Hrs), \+member(Cno,L), append(
[Cno],L,L1), A is Acc+Hrs, sum_hrs_type(Sid,A,L1,Ty,N,Li), !.
sum_hrs_type(_, Acc,Li,_, Acc,Li).

```

```

%calculate GPA -gpa(Sid,G) stores the gpa of Sid in G
gpa(Sid,G):-gpal(Sid,0,[],N,Len), G is N/Len.
gpal(Sid,Acc,L,N,Len):-coursetaken(Sid,Cid,_,_,G), \+member(Cid,L),
append([Cid],L,L1), G='A', Acc1 is Acc+4,gpal(Sid,Acc1,L1,N,Len),!.
gpal(Sid,Acc,L,N,Len):-coursetaken(Sid,Cid,_,_,G), \+member(Cid,L),
append([Cid],L,L1), G='B', Acc1 is Acc+3,gpal(Sid,Acc1,L1,N,Len),!.
gpal(_,Acc1,L,Acc1,Len):-length(L,Len).

%ind_study_ok(sid):- checks if the student with "Sid" has taken more
than 2 independent studies or not.
ind_study_ok(Sid):-
sum_hrs_type(Sid,0,[],independent,_,Li),length(Li,Le),Le<=2.

%core_study_ok(sid)- checks if the student has taken 5 core courses or
not
core_study_ok(Sid):-
sum_hrs_type(Sid,0,[],core,_,Li),length(Li,Le),Le>=5.

%thesis_study_ok- checks to see if the student has taken less than <6
or >6 hrs as thesis hrs.
thesis_study_ok(Sid):-sum_hrs_type(Sid,thesis,N),N<=6.

%grad(ssn) : can the student with ssn graduate
grad(Sid,_):-
thesis(Sid),sum_hrs(Sid,N),N>=30,thesis_study_ok(Sid),ind_study_ok(Sid)
,gpa(Sid,Gpa),Gpa>=3,core_study_ok(Sid).
grad(Sid):-
\+thesis(Sid),sum_hrs(Sid,N),N>=33,ind_study_ok(Sid),gpa(Sid,Gpa),Gpa>=
3,core_study_ok(Sid).
% The remaining courses are assigned assuming that all the courses are
for 3 credit hrs.. Just for simplicity
grad(Sid,L):- dept(Sid, Dept), sum_hrs(Sid,Num), collect_hrs(Sid,
Taken), Num1 is 33-Num, Num1\==0, Num2 is Num1/3, print('To graduate #
courses the student needs to take is '), print(Num2), nl,
print('Courses already taken ==> '), print(Taken), nl, print('For
graduation the student has the following choice of
courses'),nl,chooseN(Dept, Num2, L, Taken).

subsequence([X|Xs],[X|Ys]):-subsequence(Xs,Ys).
subsequence(Xs,[Y|Ys]):-subsequence(Xs,Ys).
subsequence([],Ys).

%=====

% this program aims at finding the # choices that are possible i.e N C
K (N choose K)

% cantake(dept, course) denotes that the student of this dept can take
this course

cantake(cs, no1).
cantake(cs, no2).
cantake(cs, no3).
cantake(cs, no4).
cantake(cs, no5).

```

```

cantake(cs, no6) .
cantake(cs, no7) .
cantake(cs, no8) .
cantake(cs, no9) .
cantake(cs, no10) .
cantake(cs, no11) .
cantake(cs, no12) .

%assign(cs,6,0,[],L,[]). gives L = [no6, no5, no4, no3, no2, no1]

assign(Dept,N,N,L,L,_).
assign(Dept,N,A,L,R,Taken):- cantake(Dept,Course), \+member(Course,
Taken), \+member(Course, L), append([Course], L, L1), A1 is A+1,
assign(Dept, N, A1, L1, R,Taken) .

%chooseN(Dept, N, L, Taken) where N is the # courses that we are trying
to select and L is the List that will contain the list of the courses
(of length N) that the student can take and Taken is the list of
courses that the student has already taken
chooseN(Dept, N, L, Taken):- assign(Dept,N,0,[],L,Taken) .

% R will store a list of lists which will contain all the possible
combinations in which the "N" courses can be taken
%my_setof(Dept,N,R):- setof(L,chooseN(Dept,N,L),R) .

```

/*

The output for a sample run of the program is shown below

Syntax of the Input: grad(SSN,List)

In our case the following SSN# correspond to the following students

SSN#1000 - a student who is eligible to graduate

SSN#1001 - a student who has taken a few courses and needs to take some more to graduate

SSN#1002 - a new Student

In the output “no#” (example no1, no2...) corresponds to the course numbers and the list L is the list of courses the student needs to take to graduate and that’s what our system advises him to take. By typing “;” we get all the possible ordering of the remaining courses that the student can take.

*/

?- grad(1000,_).

Yes

?- grad(1001,L).

'To graduate # courses the student needs to take is '5

'Courses already taken ==> '[no6, no5, no4, no10, no9, no8]

'For graduation the student has the following choice of courses'

L = [no11, no7, no3, no2, no1] ;

L = [no12, no7, no3, no2, no1] ;

L = [no7, no11, no3, no2, no1] ;

L = [no12, no11, no3, no2, no1] ;

L = [no7, no12, no3, no2, no1] ;

L = [no11, no12, no3, no2, no1] ;

L = [no11, no3, no7, no2, no1] ;

L = [no12, no3, no7, no2, no1] ;

L = [no3, no11, no7, no2, no1] ;

L = [no12, no11, no7, no2, no1]

Yes

?- grad(1002,L).

'To graduate # courses the student needs to take is '11

'Courses already taken ==> '[]

'For graduation the student has the following choice of courses'

L = [no11, no10, no9, no8, no7, no6, no5, no4, no3|...] ;

L = [no12, no10, no9, no8, no7, no6, no5, no4, no3|...] ;

L = [no10, no11, no9, no8, no7, no6, no5, no4, no3|...] ;

L = [no12, no11, no9, no8, no7, no6, no5, no4, no3|...] ;

L = [no10, no12, no9, no8, no7, no6, no5, no4, no3|...] ;

L = [no11, no12, no9, no8, no7, no6, no5, no4, no3|...]

Yes

?-