EECS 344

Project Report

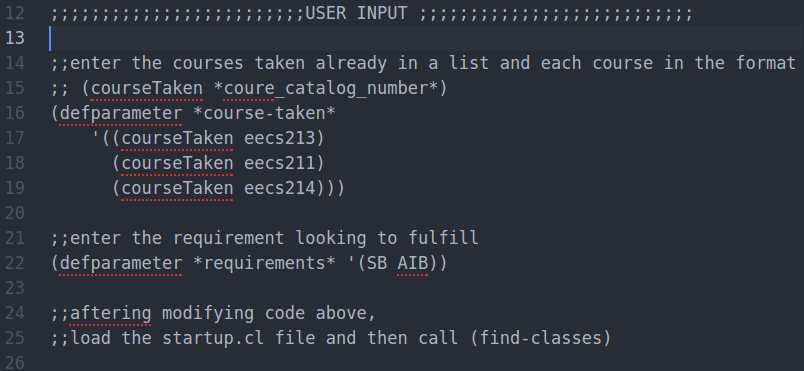
Qiang Bi

**Motivation**

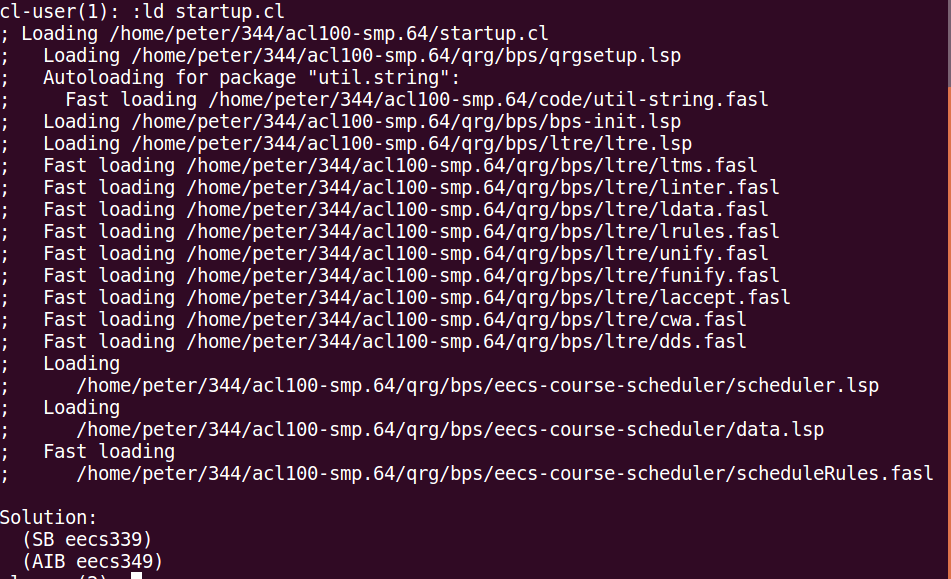
For most computer science undergraduate students, especially underclassmen, it can be frustrating to plan courses for registration. Most of the time, one has to go through almost all courses offered and check if he/she fulfills the pre-requites and if there are time conflicts among the courses. Even for a sophomore like myself, I frequently have to go through many courses and make a preliminary list, only later to realize either there is a time conflict in my course scheduling or I do not fulfill the pre-requisites. I would like to simplify the process and write a program as my term project. As a stretch goal, the project upon completing may be scalable and apply to course scheduling of every department, given the accurate class data is provided.

**Approach**

Using the techniques of building a solver for constraint satisfaction problems, such as homework 3, I desgined a solver using LTRE to take as inputs the ID of courses taken already and the requirement looking to fulfill (such as core, which breadth, depth, or project). Then the solver, with all the rules/constraints I write in, will look for one or several solutions of the class schedule that satisfy the user’s need.

I first wrote a parser in python since it is hard to get data from JSON format database. The parser, file named courseParser.py, parses all class data obtained into LTRE recognized facts (lisp expression). I also had to hard code most the class data into the system using disjunctive logics. Then, I wrote rules to enforce pre-requisite constraint and no-time-conflict constraints. As a concrete example, the student will put in all courses he/she has taken, such as (eecs213, eecs211, eecs214) and the requirement interested in fulfilling, such as (AIB, SB). Then the solver will look for combination of EECS classes offered this spring quarter that satisfy the “pre-requisite constraint” and the “no-time-conflicts constraint”, and output every feasible solution for the user to pick, which in this case will return eecs339 satisfy system breadth (SB) and eecs349 satisfy AI breadth (AIB).

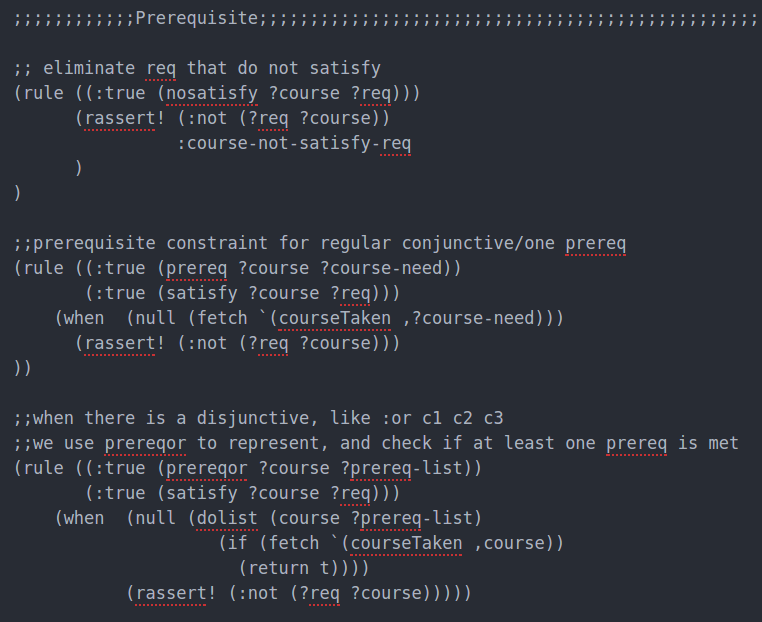
(scheduler.lsp)

 (command line output)

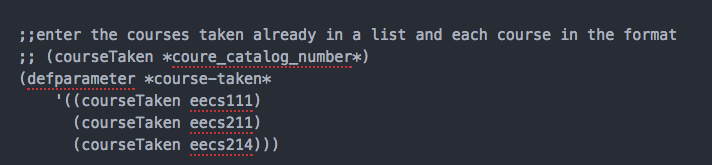
**Highlight/Overview of Code**

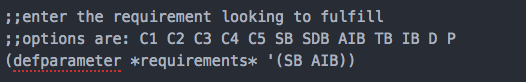
The solver consists of a couple files: scheduler.lsp, data.lsp, scheduleRules.lsp, and startup.cl. The scheduler.lsp file serves as the driver of the entire program. It consists of the dd-search call using LTRE after loading data and rules file. The data file has all the facts about the courses, and the rules file has all the rules I wrote. The startup.cl file has all the command needed to get the solver started.

In the rules file, I simply used a rule of “pairwise-nogood” between every pair of different requirements and make sure each course can only count for one requirement. For the prerequisite constraint, I rassert! negation on the course-requirement pair whenever a course is not taken. (whenever a courseTaken fact cannot be fetched). I did run into problem representing the disjunctive prerequisites, i.e. course1 and (course2 or course3 or course4). I decided to put each conjunctive part in one independent prerequisite fact, and use “prereqor” fact representation form which takes a list of disjunctive prerequisite. In the rule, “prereqor” is evaluated by looping through the prerequisites list and make sure at least one of the course in the list is taken.

****

**How to use the solver**

To use the solver, there are a couple simply steps to do. The solver is developed under LTRE, so you must have a working LTRE engine. First, the user inputs the courses that they have taken already by adding to the parameter \*course-taken\* in the format (courseTaken \*course\_catalog\_number\*). See the example screenshot.

Tthe the user needs to input the requirements the he/she intends to fulfill, such as SB, SDB, AIB, TB, IB, D, P, C1, C2, etc. (Each one ends with B is a short for breadth requirement, i.e. SB = system breadth. D = depth P = project C1-C6 = core 1 to 6) Note this list of the requirements to fulfill will be “all or nothing”, meaning that the solver will find a course schedule that fulfill all the requirements the user asks to fulfill. If the solver cannot find such solution, it will not produce a solution. Therefore, we ask the user to input at most 4 or 5 requirements, since that is the maximum number of courses you can take. It is advised that you start with one requirement and slowly add to list more requirements. 

Next, the user can save the file and open Acl100.smp64 and load the startup.cl file. Note you need to first revise the directory of each file loaded in the startup.cl file. The solver should output classes in the form of one or multiple (requirement course). If there are multiple ways to schedule courses, it will output all possible solutions, each separated with keyword solution.

**Conclusion**

The complexity of the problem leads me to simplify much of the constraint. The prerequisite constraint rules worked really well as I found a way to accurately represent and query the prerequisites of each course. (using a list) I initially wanted to have the solver being able to take as input the number of courses interested and the courses taken and produce the courses that match any requirements. However, I found out that it is really hard to implement how many “objects” dd-search assigns, and it seems like it either produce the solutions that satisfy all the attribute-object match or none. If I were to start over again, I should have noticed this potential problem earlier and seek help to resolve it earlier.