University of Washington, Bothell

CSS 482: Expert Systems

Fall 2010

In-Class Exercises

Weeks 1–3

This document outlines a problem domain that we will use in class to build up successively more sophisticated knowledge representations using facts and rules, eventually resulting in a working expert system. The problem domain is that of scheduling courses for the year. The goal is to have an expert system that receives as input the courses a student has already taken and then outputs a legal plan of study for the year. We will build this a step at a time.

Expertise You can find the course schedule for the 2010–11 academic year at `http://www.uwb.edu/css/courses/time-schedule` and the set of prerequisites for each course at `http://www.uwb.edu/css/courses/course-prereqs`.

Facts We need to identify the types of declarative knowledge in this problem domain. Then, we need to develop a representation for this knowledge. Luckily, expert systems lend themselves to experimentation.

1. Identify one kind of declarative knowledge contained in the expertise above. Propose a representation using simple *ordered facts*.

2. Does your fact representation contain enough information that you could write a simple rule to infer something new? What kind of knowledge is this new fact you inferred (or is it even a fact)? Does it require a new representation as an ordered fact? If your factual knowledge is incomplete and doesn't allow anything interesting to happen, then skip to the next step. Otherwise, state your rule and a fact that it could infer.

3. There are two basic kinds of information available from the web pages above: course schedule (what courses are to be offered in which quarter) and course prerequisite (what courses must be taken before the given course can be taken). Actually, there is a third kind of information — co-requisites — but we will ignore that for our purposes here. This suggests two different forms of ordered facts, something like "course X is scheduled quarter Y" and "course X has prerequisite Z". Develop a set (it could be a subset of the full course list; just enough to test) of such facts, then implement simple rules that list which courses are offered in fall and what the prerequisites are for each course. (Basically, these rules are just outputting the knowledge base.)

4. Clearly, this is not enough information to plan a student's course schedule because we have no information about the student! We need to know what courses a student has already taken. Define an ordered fact that contains a list of courses already taken. Using this fact, plus the facts previously defined, write a rule that will list all classes that a student is able to take *right now*. Don't worry about what quarter it is.

5. These ordered facts will quickly get cumbersome, if they haven't already. Let's define some unordered facts:

   ```
   (deftemplate course (slot name)
                       (slot quarter)
                       (multislot prerequisites))
   (deftemplate quarter (slot name) (multislot courses))
   (deftemplate student (slot name) (multislot courses))
   ```

   In these cases, the `course` fact contains schedule and prerequisite information for a course, the `quarter` fact contains the list of courses scheduled for the student in a particular quarter, and the `student` fact contains information about a student. Implement these three `deftemplate`s, with documentation strings, and use a `deffacts` construct to encode the actual course information for this year. Re-implement the preceding rule using these new facts.

Rules Now we are ready to start writing rules. As our work progresses, what we will find is that we need to either add more information to our existing facts or additional facts. That is OK; at each point in the development process, we will have a partially working expert system and we will generally find that each modification to our knowledge base will be an addition to it, rather than a re-implementation.

1. Let's start out with the set of facts you have from the last step of the previous section. Assume you will use a greedy algorithm to plan the schedule for the year, and (for the moment) that we aren't worrying about time conflicts. What information do you need to plan a quarter's schedule? Do you have enough information to do that? How do you know that you're done with planning the quarter? How can you move on to the next quarter?

2. At this point, you've likely identified that you need more facts. For one thing, you need to know how many courses a student should schedule each quarter. We'll do that by stating the number of credits desired for each quarter, plus including the number of credits for each course. That takes care of one part of the problem. The other part is keeping track of the state of the problem solution. For that we need to know the *current quarter* and the *remaining credits to schedule*. Let's make these separate, ordered facts. Why do it this way? Well, it's simple, and these facts, representing solution state information, will be used primarily by *reasoning control rules*: a separate set of rules that have as their purpose controlling when the other rules can be activated (reasoning *phases*).

3. Write a set of rules that will schedule a single quarter. Besides the course, quarter, and student specific information, plus the remaining credits to schedule, your rules should just use the current quarter name (without changing it). Initialize the current quarter to `Fall` and the credits to 15. Test your rules with a variety of quarter names, initial credits to schedule, and set of courses the student has already taken.

4. Now that you can schedule a single quarter reliably, consider the rules that start the whole process, detect when a quarter is done, advance reasoning to the next quarter, and detect when the scheduling is complete and output the final result. To do this, answer the following questions: How do I know what the quarter names are, and in what sequence they are planned? How do I know when a quarter's schedule is done? What do I need to do when one quarter is ended and I need to start on the next? Test your resultant program thoroughly.

5. For a program extension, consider how you might prevent scheduling conflicts. What additional information do you need to add to the `course` fact? How do you need to modify the reasoning process to prevent time conflicts? Could you do this by adding rules and more steps to the reasoning process, or do you need to modify many rules substantially?

6. Is this all that seems practical to do, or do you think you could push this project further? Could you incorporate degree planning? Heuristics gleaned from the CSS advisors regarding which courses to take first? Do you think it might be feasible to create an "online advisor" that could help students think about what courses to take each year? What sort of capabilities would such a system need? How could it be designed so that students understand its limitations? Do you think it could be made useful by prospective students thinking about coming to UWB?