Homework Assignment 1: Semantics, Knowledge Bases and Proofs

CSC 384 – Winter 2003

Out: January 22, 2003
Due: February 5, 2003: in class

Be sure to include your name and student number with your assignment. If your handwriting is even *possibly* illegible, be sure to hand in your assignment in some typed form.

1. Consider a simple language to compare currencies. It consists of three constant symbols `c1`, `c2`, `c3` one binary predicate symbol `higher_than` and no function symbols. Using this language, we may build the following knowledge bases:

   - KB$_1$ = {`higher_than(c1,c2).`}
   - KB$_2$ = {`higher_than(X,Y).`}
   - KB$_3$ = {`higher_than(X,Z) <- higher_than(X,Y), higher_than(Y,Z).`}

   Let's analyze the possible interpretations $I = \langle D, \pi, \phi \rangle$ where $D = \{\text{CDN\$}, \text{US\$}, \text{EURO}\}$.

   (a) How many interpretations of the above language exist using the domain D?

   (b) Are the interpretations where $\phi(c1) = \text{CDN\$}$, $\phi(c2) = \text{US\$}$ and $\pi(higher\_than, \text{CDN\$}, \text{US\$}) = true$ models of KB$_1$? Explain why.

   (c) How many interpretations are *models* of KB$_1$? Give a brief justification for your answer.

   (d) Give an interpretation that is *not* a model of KB$_2$

   (e) How many interpretations are *models* of KB$_2$? Give a brief justification for your answer.

   (f) How many interpretations are *models* of KB$_3$? Give a brief justification for your answer.

2. Imagine a new online university offering courses over the web. The university would like an automated tool to determine when a student is ready to graduate. In other words, this tool can verify whether a student satisfies all the requirements of her study program. In this question, you will design such a tool by axiomatizing the domain (i.e., specifying a knowledge base) in Prolog.

   The university would like this tool to be generic enough to work for any study program and any student. Given the requirements of a program and the transcript of a student, the tool will determine whether the student can graduate. Here are a set of predicates to specify the requirements of a program and the transcript of a student.

   - `allCourses(List)` – asserts that `List` is the list of all courses available for a study program. This list consists of a set of `[Course_id,No_credits]` pairs indicating the name of each course and its associated number of credits.

     ```
     allCourses([[c1,2],[c2,3],[c3,1],[c4,2],[c5,2],[c6,3],[c7,2],[c8,3],[c9,2]]).
     ```

- `requiredCredits(No_credits)` – asserts the minimum number of credits that must be earned for graduation.

  ```
  requiredCredits(30).
  ```

- `group(Group_id, N, CourseList)` – asserts that `N` courses in `CourseList` refered to by `Group_id` must be passed (grade of at least d-) in order to graduate.

  ```
  group(g1, 3, [c1, c2, c3]).
  group(g2, 1, [c5, c6]).
  group(g3, 2, [c7, c8, c9]).
  ```

  When the number of courses required in a group is equal to the cardinality of the group, then all courses of that group are required. This is the case of `g1`. Note also that some courses may not be part of any group, which means that they are electives. This is the case of `c4` and `c5`.

- `groupList(GroupList)` – asserts that `GroupList` is a list of group ids identifying groups of courses (see above).

  ```
  groupList([g1, g2, g3]).
  ```

- `equivalent(CourseList)` – asserts that all courses in `CourseList` are equivalent. In other words, the credits of only one course in `CourseList` may count towards graduation. Similarly, only one course in `CourseList` may count towards a group requirement.

  ```
  equivalent([c7, c9]).
  ```

  Here we can assume that equivalent courses are always part of the same group and have the same number of credits. In the above example, taking `c7` or `c9` yields 2 credits either way and counts as one course towards `g3`.

- `transcript(List)` – asserts that `List` is a list of `[Course_id,Grade]` pairs that are part of a student transcript. This list contains only the courses the student registered for.

  ```
  transcript([[c1,a], [c2,b_plus], [c3,a_minus], [c4,f], [c4,c], [c5,w]]).
  ```

  The possible grades are `a_plus`, `a`, `a_minus`, `b_plus`, `b`, `b_minus`, `c_plus`, `c`, `c_minus`, `d_plus`, `d`, `d_minus`, `f` and `w`. Here `f` means “fail” and `w` means “withdraw”. To pass a course, the grade must be at least `d_minus`. Note also that it is possible for a student to take the same course twice. In the above example, the student first failed `c4` and then passed it with grade `c`.

Using the above predicates, we can summarize in a knowledge base the requirements for a study program and the transcript of a student. Here is an example:

```
allCourses([[c1,2],[c2,3],[c3,1],[c4,2],[c5,2],[c6,3],[c7,2],[c8,3],[c9,2]]).
requiredCredits(30).
group(g1, 3, [c1, c2, c3]).
group(g2, 1, [c5, c6]).
group(g3, 2, [c7, c8, c9]).
groupList([g1, g2, g3]).
equivalent([c7, c9]).
transcript([[c1,a], [c2,b_plus], [c3,a_minus], [c4,f], [c4,c], [c5,w]]).
```

A student is eligible for graduation when she has passed enough courses in each group and when she has accumulated enough credits. We define the predicate `graduate` accordingly:

```
graduate :- transcript(L), enoughCourses(L), enoughCredits(L).
enoughCredits(L) :- totalCredits(L,TotCred), requiredCredits(ReqCred),
                    TotCred >= ReqCred.
```

Your task is to define the predicates `enoughCourses` and `totalCredits`. For parts (a) and (b) you should hand in a listing of your Prolog program with all defined predicates. Be sure to document your definitions. You should also hand in a clean script (i.e., free of extraneous information) of a Prolog session in which you run each test case provided. A set of test cases will be posted to the course website within a week of the assignment being handed out.

(a) Define a predicate `enoughCourses(L)` that is true iff the student has passed enough courses in each group. Be careful not to count courses that were failed or withdrawn from. Also, make sure to count only one course in each equivalent list. Finally do not double count courses taken more than once.

(b) Define a predicate `totalCredits(L,TotCred)` that is true iff the student has accumulated `TotCred` credits. Again, do not count courses that were failed or withdrawn from. Also, make sure to count only one course in each equivalent list. Finally do not double count courses taken more than once.

(c) We are currently storing in the knowledge base the requirements of a single program and the transcript of a single student. This allows us to formulate the query `?graduate.` with respect to the student and program stored in the kowledge base. How should we modify the predicates defined above to allow *simultaneous* storage of requirements for several programs and many student transcripts? In particular explain what argument(s) need to be added to each predicate. How should we formulate a query asking: who's ready to graduate? You don't need to implement your solution.

3. Consider the following knowledge base:

```
1. q(Y) <- s(Y,Z) & r(Z).
2. p(X) <- q(f(X)).
3. s(f(a),b).
4. s(f(b),b).
5. s(c,b).
6. r(b).
```

Show the set of (ground atomic) consequences derivable from this KB. Assume that a bottom-up proof procedure is used and that at each iteration (when a clause is selected), the clause selected is the *first* applicable clause in the order shown (e.g., if clauses 3 and 4 are applicable, we select clause 3 and derive its head first). Furthermore, applicable substitutions are chosen in "alphabetic order" if more than one applies to a given clause (e.g., if X/a and X/b are both applicable for clause 2 at some iteration, derive $p(a)$ first). In what order are consequences derived?

4. Consider the following knowledge base:

```
1. student(william).
2. student(mary).
3. prof(diane).
4. prof(ming).
5. parent(diane, karen).
6. parent(diane, robyn).
7. parent(susan, sarah).
8. parent(susan, ariel).
9. parent(karen, mary).
10. parent(karen, todd).
11. has_access(X, office) <- has_keys(X).
12. has_access(X, gym) <- prof(X).
13. has_access(X, gym) <- student(X).
14. has_access(X, gym) <- has_access(Y, gym) & parent(Y, X).
```

(a) Provide a top-down SLD derivation of the fact `has_access(todd, gym).` You can follow the style of Example 2.32 in the textbook.

(b) The fact `has_access(mary, gym)` has two straightforward, but quite distinct, SLD derivations. Give both of them.

(c) Does there exists an SLD derivation for `has_access(ariel, gym)`? Briefly, why or why not?

(d) Argue that the set of answers to the query `?has_access(X,office)` is empty. If the clause `has_keys(X) <- prof(X)` is added to KB, what is the set of answers to this query?