Thanks for helping me out on my research. Let's get straight to the task: I am asking you to write first-order statements which describe our department. First, let me describe the domain. The domain contains the following predicates:

Professor	(Person)	
Position	(Person, Position)	
Student	(Person)	
Phase	(Person, Phase)	
YearsInProgram	(Person, Integer)	
AdvisedBy	(Person X, Person Y)	"X is advised by Y"
TempAdvisedBy	(Person X, Person Y)	"X's temporary advisor is Y"
Publication	(Paper P, Person X)	"Person X is an author of paper P"
TaughtBy	(Course, Person, Quarter)	
TA	(Course, Person, Quarter)	
CourseLevel	(Course, Level)	
SamePerson	(Person, Person)	
SamePaper	(Paper, Paper)	
SameCourse	(Course, Course)	
SameProject	(Project, Project)	
SameQuarter	(Quarter, Quarter)	
SamePosition	(Position, Position)	
SamePhase	(Phase, Phase)	
SameInteger	(Integer, Integer)	

In general, for most of these predicates, you should not need to refer to specific ground constants. That is, I don't want statements referring to Publication(T,mattr) or things like that. There are some exceptions. Here are some constants you may or may not find useful:

Phase: Refers to the student's phase in grad school. There are three phases:
- post_Quals
- pre_Quals
- post_Generals

Level: Refers to the level of the course. There are four levels:

level_100	(intro courses 100, 142, 143)
level_300	(undergrad courses – junior level)
level_400	(advanced undergrad courses – senior level)
level_500	(grad level courses)

Position: Refers to the type of faculty for various professors

faculty
faculty_affiliate
faculty_adjunct
faculty_emeritus
faculty_visiting

Integer: Just used to say how many years a person has been in the program. They are year_1, year_2, …, year_12. (no jokes about YearsInProgram(mattr…)!)

The remaining types are pretty obvious. Here's a short description of each:
Person: includes professors and students
Paper: a conference or journal publication.
Course: Classes offered by the department
Quarter: identifier for the year and quarter (like autumn_0203).

Note, when you are doing the task, if you think there is some other predicate that would be useful for you, let me know. I may be able to add it to the domain pretty easily.

The Task

Please write down a first-order theory describing the domain. For simplicity, write it as a bunch of first-order statements, which are implicitly conjoined together. A statement may be anything you want which uses the above predicates. If not specified, variables are assumed to be universally quantified.

What I am looking for is a set of statements that roughly characterize our (or any) computer science department. The rules you write do not need to always be true. As long as they are generally more true than not, they will be useful.

Let me give you an example based on a "friendship" domain. You can write a simple statement like:

Friend(X,Y) => Friend(Y,X)

which means that if X is a friend of Y, then Y is a friend of X. This is not always true, but is probably true in many cases. Here's another example:

exists(Y) Friend(X,Y)

which means, for every X, there is at least one Y such that Friend(X,Y) is true. Don't worry about exact notation, as long as I can figure out what it means.

Some more examples:
Smokes(X) and Smokes(Y) => Friend(X,Y) or Friend(Y,X)
Friend(X,Y) and Smokes(X) => Smokes(Y)

Friend(X,Y) and !Smokes(X) => !Smokes(Y)
!Friend(X,Y) or !Friend(Y,Z) or Friend(X,Z).

Sometimes you have to be a little careful. Note that if I write “Friend(X,Y) and Smokes(X) => Smokes(Y)”, that does not say anything about what happens if X does not smoke, hence the need for another rule explicitly explaining what happens when “Friend(X,Y) and !Smokes(X)”. Also, of course it is not always true that a smoker’s friend smokes, but it might be more true than just 50%, so it can provide useful information/evidence to a system trying to figure out who smokes.

I would like as complete a theory as possible, but in order to help you focus your time, I would like you to pay particular attention to statements that involve the AdvisedBy() predicate, because that is probably the predicate I will be using for testing.

After you have written as many statements as you can, I would like you to go back and give each two numbers. The first (from 0-1) is the fraction of time that you think this statement is true. Don’t forget that a rule like “A => B” is true whenever A is false. The second number (from 0-1) is sort of a +/- which tells your confidence in first number. A value of 0 means it is absolutely right, and a value of 1 means you basically have no idea what it should be. For example:

0.8 0.4 Friend(X,Y) and Smokes(X) => Smokes(Y)

Means I think that maybe roughly 80% of smoker’s friends will smoke. But I’m not sure, it might be quite a bit more or less than 80%.

The rules themselves are more important to me than the numbers, so don’t stress over the numbers. Feel free to take whatever amount of time you want. Hopefully it will be a bit fun coming up with these statements. I’m hoping you can spend at least half an hour, but more time is certainly not discouraged!