Logical Foundations of Computer Science

Syllabus for the course CS 5303, Spring 2016

Instructor: Vladik Kreinovich, email vladik@utep.edu, office phone (915) 747-6951

Class time: MW 3:00-4:20 pm

Office hours: Mondays 1:30-3 pm and 4:30-5:30 pm, Wednesdays 1:30-3 pm, or by appointment

MAIN OBJECTIVE: to teach logical foundations of computer science.

MOTIVATION: How can we make sure that a program computes what the user want? Of course, we can (and we should) test the program on several examples. However, if the program works well on several example, there is no guarantee that it will always work well. Such a guarantee cannot come from testing, it has to come from *reasoning* about the program. The science of reasoning is called *logic*. So, to make sure that our programs satisfy the user specifications, we must use logic. In this class, we will study different logical methods and techniques.

Reasoning about complex objects -- such as complex programs -- is not easy. Just as we human make mistakes when computing by hand and when coding, we often make mistakes when reasoning. Computing becomes easier (and less error-prone) when we make computers help us compute; for that, we need to describe the corresponding computational tasks in precise terms -- the terms that a computer can understand. Similarly, to make reasoning easier and less error-prone, we need to describe the corresponding logical problems in precise terms, terms a computer can understand and process. Thus, a significant part of this class will be devoted to a formal description of different logics, and to algorithms that help solve the corresponding logical problems.

CONTENTS: We start with the simplest logic -- *propositional* logic, a logic of "and", "or", and "not" operations used in if-statements. We will learn how to translate appropriate English phrases into this language, and how to reason in propositional logic. Specifically, we will first study so-called natural deduction -- a commonsense way of reasoning about propositional formulas. As often happens in computer science, the natural way is not always the most efficient one, so we will study different ways to reason about such formulas, some of which related to digital design (and its normal forms), some related to Artificial Intelligence (resolution).

Interestingly, often, once we have formulated the specifications in terms of propositional logic, we can automatically generate the corresponding program. In this class, we will describe the main ideas behind such logic-based automated program synthesis.

Not all statements can be expressed in propositional logic: often, to check the program's correctness, we need to make sure that it works correctly *for all* possible inputs. Statements containing *quantifiers* such as "for all" and "there exists" form *predicate* (*first-order*) logic. Again, we will learn how to translate appropriate English statements into this logic, and how to reason in this logic.

The predicate logic is much more complex than the propositional logic: In propositional logic, there is an algorithm that solves all the problems; this algorithm may require too much computation time, but it exists. In contrast, for predicate logic, no such general algorithm is possible. It is therefore desirable to find practically important classes of problems which can be algorithmically solved. In this class, we will learn two such classes, that correspond to temporal logic (reasoning about changes) and modal logic (reasoning about what is possible and what is not). We will show how these logics can help, correspondingly, in program verification and in dealing with uncertainty (in particular, with interval uncertainty).

All the above techniques deal with situations where the statements have an absolutely precise meaning. However, often, the users are not precise. For example, if it is not possible to always produce an exact solution to an optimization problem, a user may want a program that will produce a solution which is, in the vast majority of cases, *close* to the optimum -- without specifying what "close" or "vast majority" mean. The last topic that we will study -- multi-valued logics (such as fuzzy logic) -- help to describe such statement inside the computer, and to have a computer process such statements. Interestingly, such logics also help to describe how we think -- and thus, help computers emulate our informal reasoning.

PROJECTS: An important part of the class is a project. There are three possible types of projects:

- If, as part of your current research project, you are interested in doing some research related to logical foundations, a research that will be useful for your future thesis, project, or dissertation, this will be an excellent project for the class.
- Another possibility is to participate in a hands-on advanced logic-related project, usually in collaboration with someone who is already working on a related topic. A list of such projects will be given shortly after the class starts. These projects will range from relatively easy to truly challenging ones -- ones which could eventually lead to a publication.

THERE IS NO TEXTBOOK: we will use handouts and links

TESTS AND GRADES: There will be two tests and one final exam. Each topic means home assignments -- both theoretical and programming. Maximum number of points:

- first test: 10
- second test: 25
- home assignments: 10
- final exam: 35
- project: 20

(smart projects with ideas that can turn into a serious scientific publication get up to 40 points).

A good project can help but it cannot completely cover possible deficiencies of knowledge as shown on the test and on the homeworks. In general, up to 80 points come from tests and home assignments. So:

- to get an A, you must gain, on all the tests and home assignments, at least 90% of the possible amount of points (i.e., at least 72), and also at least 90 points overall;
- to get a B, you must gain, on all the tests and home assignments, at least 80% of the possible amount of points (i.e., at least 64), and also at least 80 points overall;
- to get a C, you must gain, on all the tests and home assignments, at least 70% of the possible amount of points (i.e., at least 56), and also at least 70 points overall.

STANDARDS OF CONDUCT: Students are expected to conduct themselves in a professional and courteous manner, as prescribed by the Standards of Conduct. Students may discuss programming exercises in a general way with other students, but the solutions must be done independently. Similarly, groups may discuss project assignments with other groups, but the solutions must be done by the group itself. Graded work should be unmistakably your own. You may not transcribe or copy a solution taken from another person, book, or other source, e.g., a web page). Professors are required to - and will - report academic dishonesty and any other violation of the Standards of Conduct to the Dean of Students.

If you feel you may have a disability that requires accommodation, contact the The Center for Accommodations and Support Services (CASS) at 747-5148, go to Room 106 E. Union, or e-mail to cass@utep.edu. For additional information, please visit the CASS website.