THEORY OF COMPUTATION

Syllabus for the course CS 5315, Spring 2016

Instructor: Vladik Kreinovich, email vladik@utep.edu, office CCSB 3.0404, office phone (915) 747-6951

Class time: MW 12-1:20 pm, in Room CCSB 1.0204

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

Prerequisite: CS 3350 (Automata)

MAIN OBJECTIVES:

- to teach the foundations of computing, and
- to enable students to prove (not only to program).

CONTENTS

1. *Turing's snakelike machine:* not very fancy, but it can compute anything (you just wait and wait and wait, ...). Finally: something purely theoretical (and not real machines): *recursive functions*. Church's bold statement: if anyone can compute anything on any machine, I can compute it on a Turing snake! Universal Turing machine. Can anyone really beat Church? We'll discuss the attempts (Gandi, Kreisel, etc) if time allows.
2. You are accustomed to the fact that everything is computable, and if your program does not work, that means a bad grade. Finally! Only in this course! Computational problems that cannot be solved! (and so you get a bad grade, if your program solves them - just kidding).
3. If a program requires a billion years to finish its computations, only a crazy theoretician can call it an algorithm. So, to sound more reasonable, we will talk about computational complexity, realistic (polynomial-time) computations, P and NP, NC, limitations on space and on the number of processors, etc. "P=NP?" as a challenge to mankind. Will science ever stop? Again, we will find here lots of undecidability results. And maybe, as a project, you will be able to prove that some problem that you were planning to solve is undecidable.
4. What to do if your problem turned out to be undecidable? For sure: not to give up. It can be still decidable in some sense: for almost all cases, by a Monte-Carlo algorithm that gives an answer with probability close to 1, etc. Few results and lots of open problems.
5. Turing machine was invented in the 30s, P=NP problem appeared when many of you guys were too young to count. What is the modern state of the Theory of Computation? We'll try to cover briefly:
    - parallelism,
    - algebraic computations,
    - computations with real data (including the idea of interval mathematics),
    - fuzzy algorithms,
    - neural computing,
    - chemical computing,
    - quantum computations, etc.

PROJECTS. After a few lectures, you will be given a list of projects to choose from (or you may be smart enough to propose something on your own). These projects will include mainly theory; you'll

have to read, analyze and discuss some theoretical paper, and ideally come out with some new ideas (don't be afraid; Nobel prize is desirable for an A, but not necessary).

LEARNING OUTCOMES

1. **Knowledge and Comprehension**

a. Describe the practical need for theory of computing: to know which computational problems are solvable and which are not, and which problems can be, in principle, solved within given computation time, to avoid wasting resources on trying to solve problems in such a general context that they become unsolvable

b. Describe different models of computing, including recursive functions, different versions of Turing machine, etc.

c. Describe how computability in a programming language is related to formal models of computing, on the example of primitive recursive and recursive functions

d. Understand Church-Turing thesis and understand the status of this thesis - that it is, in effect, a statement about the physical world

e. Define decidable and recursively enumerable (r.e., semi-decidable) sets

f. Understand the notion of an oracle and of computing relative to an oracle

g. Define classes P, NP, the notions of polynomial time reduction, NP-hardness, and NP-completeness; understand the motivations behind these definitions: P means feasible, NP stands for a problem; understand the difference between the formal notion of polynomial time and the practical notion of feasibility

i. Understand the difference between a proof and a sequence of reasonable arguments which does not constitute a proof

j. Know several NP-hard problems

k. Define complexity classes from the absolute and relative polynomial hierarchy; give examples of problems

l. Understand the main idea of parallelization

m. Define the class NC of parallelizable P problems, know the relation between NC and P, and an example of a P-complete problem

n. Understand the difference between the formal computation time of a parallel program and the actual time which takes communication time into account

o. Define Kolmogorov complexity, understand motivations behind this definition

p. Be aware of the main ideas behind different physical schemes of computing beyond traditional Turing machines, such as quantum computing

2. **Application and Analysis**

a. Trace the computation of a primitive recursive or recursive function on a numerical example

b. Translate a formal description of a primitive recursive or recursive function into a program

c. Trace a Turing machine on a given input

d. Prove that satisfiability is NP-hard, and that one more problem is NP-hard

e. Know how to parallelize standard simple parallelizable algorithms (e.g., search, or dot product of two vectors)

f. Use the definition of Kolmogorov complexity K(x) to provide reasonable upper bounds for K(x) of a given string x

3. **Synthesis and Evaluation**

a. Prove, from the definition, that a given function (e.g., a given polynomial or propositional function) is primitive recursive or recursive

b. Design a Turing machine that computes a given function

c. Synthesize Turing machines that compute two functions into a Turing machine for computing their composition

d. Prove that not every problem is computable - e.g., that the halting problem is undecidable

e. Apply diagonalization to prove results similar to what we had in class

f. Prove that given simple sets are decidable and/or r.e.

g. Prove that the union, intersection, and complement of decidable sets are decidable

h. Prove that the union and intersection of r.e. sets is r.e.; prove that the complement to a r.e. set is not always r.e.

i. Prove that not every r.e. set is decidable and that not every set is r.e.

j. Prove, for a given software requirement, it is algorithmically impossible to check whether a given program satisfies this requirement

k. For problems similar to ones considered in class, prove their NP-hardness

l. Be able to parallelize simple algorithms

m. Be able to understand and present a research paper in Theory of Computing - with a minor help from a professor

MAIN SOURCE: Michael Sipser, Introduction to the Theory of Computation, PWS Publishing Co., 2nd or later edition

TESTS AND GRADES: There will be three tests and one final exam. Each topic means home assignments (mainly on the sheets of paper, but some on the real computer). Some of them may be graded. Maximum number of points:

- first test: 10
- second test: 10
- third test: 15
- 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.

DISABILITIES: 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.

SEE YOU IN THE CLASS!