Theoretical Computer Science Bridging Course

-

Introduction / General Info

Summer Term 2016

Fabian Kuhn

About the Course

Topics

- Foundations of theoretical computer science
- Introduction to logic

No lectures

- There are recordings which you are supposed to watch

Exercises

- There will be weekly exercises which you need to do
  - 50% of the points are needed to be admitted to the exam
- You can work in groups of 2 or 3 students

Exam

- A written exam at the end of the term
  - Details will be published on the course web page a.s.a.p.

About the course

What is the purpose of the course?
Who is it targeted to?

- The course is for incoming M.Sc. students who do not have the necessary theory background required by the M.Sc. program.
  - E.g., students who did not study computer science or students from more applied schools, ...

Website

- All necessary information about the course will be published on

  http://ac.informatik.uni-freiburg.de/teaching/ss_16/tcs-bridging.php

  - Or go to my group's website: http://ac.informatik.uni-freiburg.de
  - Then follow teaching – summer term 16 – TCS bridging course

- Please check the website for
  - Recordings and slides
  - Exercises and sample solutions
  - Pointers to additional literature
    (e.g., written lecture notes from an older version of this lecture)
  - Information about the exam
  - …

Exercises

There will be weekly exercise sheets:

- Exercise sheets are published at the latest on Monday on the website
- Exercises are due after one week on the coming Monday by midnight
- Hand in your exercises electronically (by email) to your tutor

- For the exercises, you can build groups of at most 3 students
- Each group needs to hand in one solution
  - Make sure that all students in the group participate in the writing equally!
- After handing in the exercises, you need to meet and discuss the exercises with your tutor

Exercise Tutorials

Tutors for the course:

- Chaodong Zheng, chaodong@cs.uni-freiburg.de
- Anisur Rahaman Molla, anisur.rahaman@cs.uni-freiburg.de

Weekly Tutorials:

- You need to meet your tutor once a week
  - Each group can schedule the weekly meeting individually
- In the tutorial, you need to explain your solutions to your tutor!
- Also ask your tutor if you have any questions!

Exercises

The exercises are the most important part of the course!

- To pass the exam, it is important that you do the exercises
- If you feel comfortable with all the exercises, you should also be able to pass the exam

- You therefore need to achieve 50% of the points to be admitted to the exam!

- When working in groups, make sure that you all participate in solving the questions and in writing the solutions!
  - You will all need to explain your solutions to your tutor.

Course Topics

Foundations of Theoretical Computer Science

- Automata theory
- Formal languages, grammars
- Turing machines
- Decidability
- Computational complexity

Introduction to Logic

- Propositional logic
- First order logic

Purpose of the Course

Goal: Understand the **fundamental capabilities** and **limitations** of **computers**

- What does it mean to “compute”?
  - Automata theory
- What can be computed?
  - Theory on computability/decidability
- What can be computed efficiently?
  - Computational complexity

Meaning of “Computing”

Mathematical Models

- Turing machines 1930s
- Finite state automata 1940s
- Formal grammars 1950s

Practical Aspects

- Compute architectures 1970s
- Programming languages 1970s
- Compilers 1970s

Is My Function Computable?

Write an algorithm / computer program to compute it

- Can it compute the right answer for every instance?
- Does it always give an answer (in finite time)?
- Then you are done.

Otherwise, there are two options

- There is an algorithm, but you don't know it
- There is no algorithm → the problem is unsolvable

Formally proving computability is sometimes hard!

- But you will learn how to approach this…

Is My Function Computable?

- Many “known” problems are solvable
  - Sorting, searching, knapsack, TSP, …
- Some problems are not solvable
  - Halting problem
  - Gödel incompleteness theorem
- Don’t try to solve unsolvable problems!

Can I Compute My Function Efficiently?

- Some problems are “easy”
  - Can we formally define what this means?
- **Complexity theory** is about this
  - Complexity classes, tools for checking membership
- It is important to know how hard a problem is!

- **Feasible problems:**
  - E.g., sorting, linear programming, LZW compression, primality testing, …
  - Time to solve is polynomial in the size of the input
- **Problems that are considered infeasible**
  - Some scheduling problems, knapsack, TSP, graph coloring, …
  - Important open question: “Is $\mathrm{P} = \mathrm{NP}$”?
- **Unfeasible problems**
  - Time exponential in input, e.g., quantified Boolean formula

Questions?