Overview

Theoretical Computer Science

- What is the theory of computer science?
- What are the theoretical foundations of computer science?

When we talk about computer science, we need to ask ourselves these questions:

- What is a computer?
- What is a problem?
- Are there problems that have no solution?
- Are there problems that cannot be solved?
- Do the last two questions mean the same thing?
- If a problem can be solved: How long does it take until the solution will be found?
- And by the way: What exactly does “solving a problem” mean? What does “computing” mean?

These are the questions that need to be answered when we talk about the theoretical foundations of computer science. We are not talking about writing programs, we are not talking about silicon chips, and we are not talking about any specific applications. We are diving deep into the fundamental ideas of computer science, and we will also have to think about machines that can never be built, because we can learn interesting things from these imaginary machines.

The simple answers:

A problem is a general question that needs to be answered. A problem can also be a task that needs to be done. In everyday life, we use the word “problem” for things that cause trouble, but “trouble” is not a category of science.

Our course is divided into 5 chapters:

Modeling

This is about how to extract the abstract and naked question that needs to be answered if you want to solve a problem. We want to find the fundamental idea behind a problem. If we can describe that idea in a formal way, detached from the original problem, we may see things we would not have seen before. Modeling means to describe something in mathematical terms or by using a description language like UML (Unified Modeling Language) or by drawing a map, etc.

Formal languages

When we talk about formal languages, we also talk about automata, and that means we talk about machines that can compute. After this chapter, you will look at languages (any kind of language, including programming languages or even natural languages like English) from a completely different perspective. We will find 4 classes of languages and 4 kinds of machines that can process these languages, and we will see that each language corresponds to a class of problems, and that solving a problem always means finding the right answer to the question “does the word W belong to the language L?” You will also learn that the number of problems is infinitely large, and that also the number of solutions is infinitely large, but these infinities still have different sizes.

Computational Complexity

We assume that a problem can be solved, and we are interested in how quickly we can achieve the solution. We will take a closer look at a problem that is very common when dealing with computers: You have to sort things. There are several ways to do this, and we will see what happens if we first sort 1000 things, then 2000, then 3000, and so on. How does the time it takes us to sort change when we change the number of things to sort? And what can we learn from this behavior for any other type of problem?
By the way, you can win $1,000,000 if you find an answer to a particular complexity question. (No kidding!)

Decidability

We will find answers to these three questions here:

- Are there problems for which there is no solution?
- Are there problems that cannot be solved?
- Do the last two questions mean the same thing?

Spoiler alert: The answers are (in the same order as the questions): Yes, yes and no.
We will see that there are many problems for which there is no solution, and we'll also see that when a problem has a solution, sometimes it's still impossible to find it. And we will see why that is.

Logic

We will talk about statements that are true and how to create new true statements. We will see that there are different ways to define logic, and we will take a closer look at some of them that have proven useful in certain situations. You should know that these types of logic exist and how to use them.