Theory of Computation, the Base of Disciplines in Computer World

The theory of computation contains a wide spread of concepts in computer world. When we are learning in the computer world, for instance, computer science and technology, artificial intelligence and so on, it is not rare that the knowledge contacts the theory of computation closely in many ways, as one says, “if electronics constructs the body of computer, then the theory of computation forms the spirit”. The theory of computation includes several topics: automata theory, formal languages and grammars, computability, and complexity.

During the learning, we cannot get rid of the theory of computation. Sometimes we might not realize that some knowledge is associated with the theory of computation, but it is in the field of the theory of computation. The most common situation is the analysis of time/space complexity of algorithms when learning data structure and algorithm analysis, one of the most basic courses. Besides, In the study of cryptography, the complexity is significant. It specifically requires computational problems that are hard, rather than easy. Secret codes should be hard to break without the secret key or password, that is, the encryption method should be reliable. Complexity theory has pointed cryptographers in the direction of computationally hard problems around which they have designed revolutionary new codes.

Computability and complexity are parts of the theory of computation. The difference is that computability studies that “if there is a computer program that can provide a solution of mathematical function, no matter the amount of time and space available to carry out the computation, even infinite”. However, complexity theory studies that “the ‘practical’ aspects of computability; that is, for a computable function, it answers the question: How much time and space will be needed for the computation?”. These two concepts help people to categorize the mathematical problems into some categories such that people can know what problems are solvable and what are not, and what are easy to solve and what are hard. First, we need to define precisely the notion of a computable function, though it may be difficult. However, it is clear for some problem. Only when it is computable, can we do a

deeper work in it. And then we need to find an effective solution, the shorter time and smaller space, the better, which is under the consideration of complexity.

Learning is the processing to learn the main idea of notions. Abstraction of concepts is essential and the construction of models is one way. Although modelling would lose some information, it can help us understand the general nature of the discipline, and not be disturbed by some details that not matter much. **Automata** may sound unacquainted but we have actually got in touch with it in digital design. Mealy machine and Moore machine are two useful finite state automata in digital design. They describe how the hardware behaves in a determined state, and how it transfers to next state. It is a good way for us to understand the HDL (hardware description language) instead of focusing on the concrete circuits. Automata can execute the algorithms. Another one of the most commonly used automata is Turing machine, which is powerful in describing algorithms. A simple criterion involving the number of steps a Turing machine needs to solve a problem allows us to distinguish between problems that can be solved in a reasonable time and those that cannot. However, not all problems are solvable with finite automata or Turing machine (also named "undecidable" problems), we can look for another more powerful type of automata.

There are thousands of kinds of programming languages and these languages can be categorized into several categories, based on their functions, design ideas, targets and so on. But which language should we choose to formalize and describe the problems? For generality, we need to design a kind of language that is understandable and can be utilized easily to construct the problems. These languages are called **formal languages**. It is an abstraction of the general characteristics of programming languages. It consists of a set of symbols and some rules of formation by which these symbols can be combined into entities called sentences. A formal language is the set of all sentences permitted by the rules of formation. Although it may be simpler than the famous programming languages, they have many of the same essential features. And the construction of automata is usually based on the formal languages, which will make it easier. Besides, context-free grammars and push-down automata are used in software form in compiler design and other eminently practical areas.

In a nutshell, the theory of computation provides science in computer world with concepts, models, and formalisms for reasoning about both the resources needed to carry out computations and the efficiency of the computations that use these resources. There is no notion or concept in computer world can escape from the accompany of the theory of computation.