International Journal of Advanced Research Methodology in Engineering & Technology,
Volume 1, Issue 2, March 2017, ISBN 978-1-63535-889-6

A survey: Theory of Computation

Lokesh Kumar Singh
Department of Computer Applications
Dayananda Sagar College of Arts Science and Commerce,
Bangalore, India
lokeshsingh91192@gmail.com

Lawrence Borah
Department of Computer Applications
Dayananda Sagar College of Arts Science and Commerce,
Bangalore, India
borahlawrence@gmail.com

Abstract— An artistic representation of a Turing machine. Turing machines are frequently used as theoretical models for computing. In theoretical computer science and mathematics, the theory of computation is the branch that deals with how efficiently problems can be solved on a model of computation, using an algorithm. The theory of computation has much application functioning in "real time" such as a working computer machine, the compliers in computers, scanners for scanning etc and all. In this research paper our main focus was on three topics, primitive recursive functions, halting problem and primitive recursive.

I. Introduction

In theoretical mathematics and computer science, the theory of computation is the branch that deals with how competently problems can be solved on a model of computation, using an algorithm. The field is divided into three major branches: automata theory and language, computability theory, and computational complexity theory. In order to perform a rigorous study of computation, computer scientist's work with a mathematical abstraction of computers called a model of computation. There are several models in use, but the most commonly examined is the Turing machine Computer scientists study the Turing machine because it is simple to formulate, can be analyzed and used to prove results, and because it represents what many consider the most powerful possible "reasonable" model of computation. It might seem that the potentially infinite memory capacity is an unrealizable attribute, but any decidable problem solved by a Turing machine will always require only a finite amount of memory. So in principle, any problem that can be solved by a Turing machine can be solved by a computer that has a finite amount of memory.

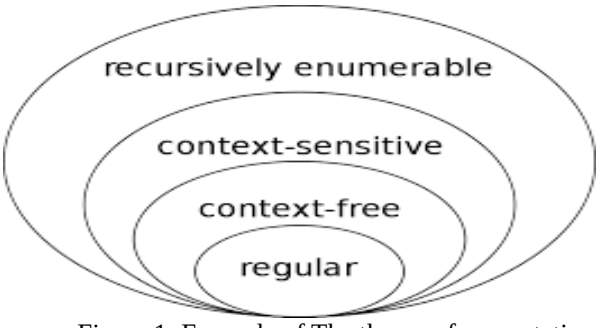

Figure 1: Example of The theory of computation

II. Theory of Computation Advantages

Theory of computation lays a strong foundation for a lot of abstract areas of computer science. If you look at it from a distance, theory of computation is a very close cousin of Artificial Intelligence than say Probability or Computer vision. Theory of computation teaches you about the elementary ways in which a computer can be made to think. There is a great deal of work that was made possible in the area of Natural Language Processing that involved building Finite State machines also known as Finite State Automata. State machines are also used in certain areas of mathematics like Number theory. Regular expressions can be beautifully represented using Non-deterministic Finite Automata. Any algorithm can be expressed in the form of a finite state machine and can serve as a really helpful visual representation of the same. Sometimes, the finite state machines are easier to understand thus helping the cause furthermore. For example, consider topological sort. Representing each stage of the topological sort as a state of the automata can end up as a brilliant explanation that couples ideas from so many fields of theoretical computer science put together. What I have talked about are only a few day to day commonplace applications of theory of computation. There are a lot of interesting areas where concepts from theory of computation are used.

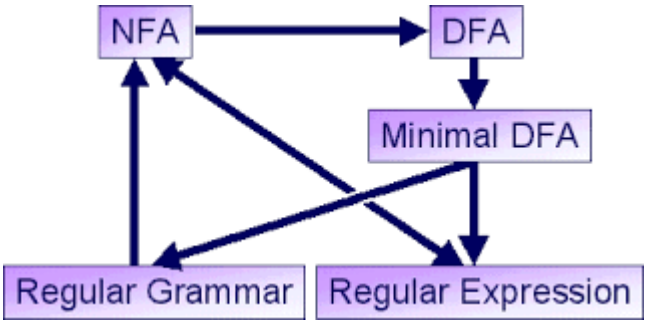

Figure 2: Basic expression of The theory of computation

III. Theory of Computation real life applications

The two important practical applications are:

1. Let's start with one very simple application of Theory of computation: an application functioning in "real time." It is called a computer, computers run on a real-time algorithm. If the hardware can't process data in a known time allotment, nothing can be processed in "hard" real time. There are other general uses of Theory of computation, such as cloud computing. The concept of a VM was invented in Theory of computation, and then developed in the real world.
2. Computer: It is nothing but Turing Machine with limited memory.
3. Programming Languages/ Compiler, Finite Automata / Regular Language (for scanning), Push Down Automata/ Context Free Language (for parsing)

Apart from this, Theory of Computation is the base for Complexity Theory which discusses what computer can do, how much efficient etc. A computer science engineer generally comes across lot many NP Hard problems. So knowing that whether a problem is efficiently solvable or not is also a practical application derived from Theory of Computation.

IV. Primitive Recursive Functions

.

In computability theory, primitive recursive functions are a class of functions that are defined using primitive recursion and composition as central operations and are a strict subset of the total μ-recursive functions. Primitive recursive functions form an important building block on the way to a full formalization of computability. These functions are also important in proof theory. Most of the functions normally studied in number theory are primitive recursive. For example: addition, division, factorial, exponential and the nth prime are all primitive recursive. So are many approximations to real-valued functions. In fact, it is difficult to devise a total recursive function that is not primitive recursive, although some are known. The set of primitive recursive functions is known as PR in computational complexity theory. Primitive recursion is a way of mathematically encoding the idea of a certain type of algorithm. Rather than giving definitions, I'll illustrate the distinction with examples which should be clear enough. In particular, I'll give two algorithms to divide an even integer by 2.

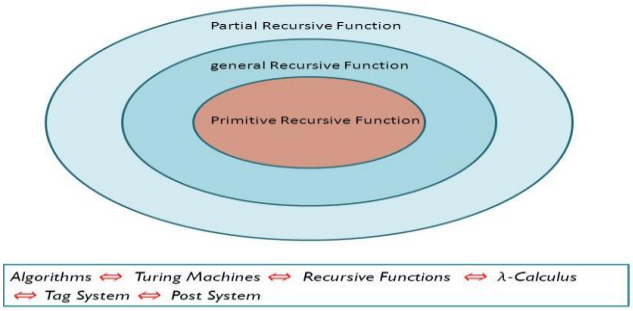

Figure 3: Block diagram of primitive Recursive function

V. Primitive Recursive Functions Advantages

Avoidance of unnecessary calling of functions, a substitute for iteration where the iterative solution is very complex. For example to reduce the code size for Tower of Honai application, a recursive function is best suited. Extremely useful when applying the same solution.

VI. Primitive Recursive Functions Disadvantages

Primitive recursive functions tend to correspond very closely with our intuition of what a computable function must be. Certainly the initial functions are intuitively computable (in their very simplicity), and the two operations by which one can create new primitive recursive functions are also very straightforward. However the set of primitive recursive functions does not include every possible total computable function — this can be seen with a variant of Cantor's diagonal argument. This argument provides a total computable function that is not primitive recursive.

VII. Halting Problem

The halting problem is a cornerstone problem in computer science. It is used mainly as a way to prove a given task is impossible, by showing that solving that task will allow one to solve the halting problem. Randall, however, is providing a simpler solution. He implements his own code for the question "Does it halt?" which always returns "true", and directs us to think about the bigger picture. From a physical perspective, according to our current understanding of physics, this is right. Given enough time, any program will halt. This is due to factors external to the actual program. Sooner or later, electricity will give out, or the memory containing the program will get corrupted by cosmic rays, or corrosion will eat away the silico1n in the CPU, or the second law of thermodynamics will lead to the Heat death of the universe. Nothing lasts forever, and this includes a running program.

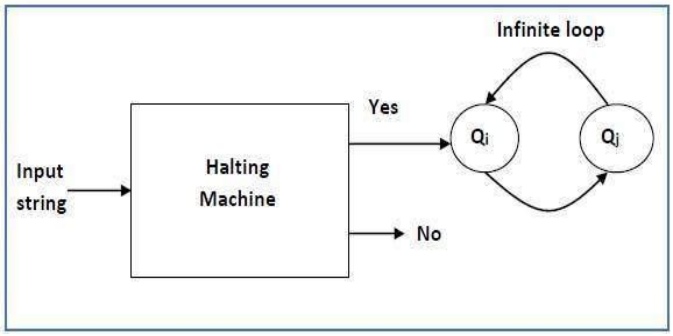

Figure 4: Block diagram of Halting Program

HALTING PROBLEM ADVANTAGES

An advantage of compile-time type checking is that it catches errors earlier than run-time checking. This follows from the undecidability of the halting problem.

VIII. HALTING PROBLEM APPLICATIONS

Alan Turing proved in 1936 that a general algorithm to solve the halting problem for all possible program-input pairs cannot exist. A key part of the proof was a mathematical definition of a computer and program, which became known as a Turing machine; the halting problem is undecidable over Turing machines.

X. RECURSIVE PREDICATES

Case discrimination function is primitive **recursive**. The function is primitive **recursive** by the following explicit **definition** which uses monadic discrimination on the first argument: D(0,y,z) = z D(x + 1,y,z) = y. 1.3.2 Equality **predicate** is primitive **recursive**.

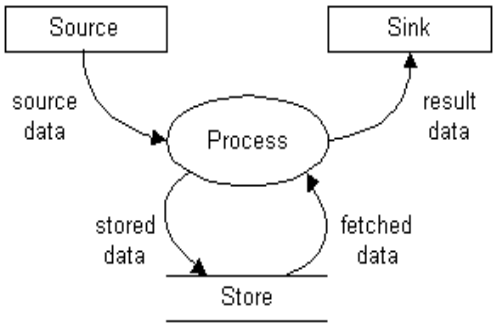

Figure 5: Flow of Work

CONCLUSION

There are lots of advantages of the Thery of Computation in the field of computer science. Therefore in this research paper our main focus was on the importance of primitive recursive functions, halting problem and primitive recursive.

REFERENCES

1. ACM/IEEE Joint Task Force on Computing Curricula. (2012). Computer Science Curricula 2013 Strawman Draft (Feb. 2012).

2. ACM/IEEE Joint Task Force on Computing Curricula. (2005). The overview report. ACM, AIS, IEEE-CS, September 30, 2005.

3. ACM/IEEE Joint Task Force on Computing Curricula. (2004). Software Engineering 2004. ACM, IEEE-CS, August 23, 2004.

4. Denning, P.J., Comer, D.E., Gries, D., Mulder, M.C., Tucker, A., Turner, A.J. & Young, P.R. (1989) Computing as a discipline. Communications of the ACM, 32(1), 9-23. Dewar, R.B.K. & Schonberg, E. (2008).

5. Computer Science Education: where are the software engineers of tomorrow? Crosstalk: The Journal of Defense Software Engineering, January 2008, 28-30. Dijkstra, E.W. (1988).

6. On the cruelty of really teaching computing science. Unpublished manuscript EWD 1036. Hopcroft, J.E., Motwani, R. & Ullman, J.D. (2007).

7. Introduction to Automata Theory, Languages, and Computation, 3rd ed., Pearson Education. IDEA League (2001). Report on comparison of curricula in Computer Science. ICL, TUD, ETH & RWTH. Oct, 12, 2001.