Machines in Automata Theory

Tom Fuller

Automata theory is the area of computer science concerned with the abstract machines used to solve various problems. Since these problems can vary in terms of difficulty, it is only natural that not all of these abstract machines are suitable to solve the desired problems. The various manifestations of automaton build upon each other, each allowing for more complicated problems to be solved under the additional overhead added between the different machines.

The most basic automaton is the deterministic finite automaton, or DFA. This automaton is the foundation for every other machine. It is the most basic because of the deterministic nature, in that only one path from the current state can be taken. Additionally, the automata is finite in nature, meaning that it can be stored within memory. Due to the mathematical notation for these machines, a DFA's validity can be proved by induction. This is what makes the DFA the simplest automaton, and it is therefore the foundation for all other automaton that look to solve more difficult sets of problems.

The second class of automata is the Nondeterministic Finite Automaton, or NFA. This allows for some simplification in terms of setup and notation, but due to its finite nature, this automaton can also be converted to a DFA, at the expense of more states. The class of problems that both the DFA and NFA can solve fall under the category of regular languages. Regular languages are any language that can be described using regular expressions for their notation.

In order to expand upon the amount of languages that can be interpreted by the automata, the Pushdown automaton, or PDA was created. The additional overhead added from the NFA is the stack structure, where the top of the stack is used and manipulated in order to take transitions in the state diagram. The stack allows for acceptance of languages that would not be possible using an NFA or DFA. The languages that are accepted by the PDA are said to be context-free because they do not depend on the adjacent symbols like a regular language. These languages can be given in the form of production rules, which are only concerned with the current state and the next possible states. The stack structure is needed to achieve this because the most recent state is always pushed to the top of the stack, and the actual position in the state transition diagram becomes less important.

The next step is to allow for all formal grammars to be accepted. This is accomplished by the Turing machine, which allows for a transmutable tape to be given to the machine. By moving back and forth on the tape, which is different than the stack which only allows access to the top-most element, the machine is said to accept all recursively enumerable languages. These languages are said to be recursively enumerable because there exists a set of words in the language that are defined by all recursive combinations over the alphabet for that language. The recursive nature ensures that all possible words are covered, making this the most complicated set of problems that can be solved. Although these machines are capable of solving every language in the Chomsky hierarchy, it is important to note that at this point it is no longer possible to prove that a specific Turing machine will actually halt due to the inherent recursive nature.

The halting problem leads into the concept of computational complexity, since certain programs can be analyzed using a turing machine to determine the amount of computations required. The two sets of languages that arise are formed by the set of all languages that are a subset of $\{0, 1\}^*$. The sets are called P, for all languages that have a turing machine with polynomial time complexity. However, the other set, NP, is composed of turing machine solutions that may exist, but have been mathematically proven to have a nondeterministic polynomial solution. The deterministic nature of the polynomials is what creates the difference in complexity across the two recursively enumerable languages or problems. From the foundations in computational theory, these core theoretical concepts for modern day algorithms are formed.