The Turing Machine

Guy Lebanon

August 22, 2006

Computers are different from each other in their software and hardware. To construct a stable theory of computation that will apply to all standard computers we need to consider a theoretical model for a computer that will be equivalent to all other standard computers. The theory can then be phrased in terms of that theoretical model and be valid by extension to all other standard computers.

The standard theoretical model in computer science for a computer is the Turing machine which was proposed by Alan Turing around 1936. It is considered event today, 70 years after its formulation, equivalent[1] to standard modern computers[2]. A Turing machine is a machine that interacts with the outside world by reading an infinite tape (the tape is divided to cells on which symbols or blanks are written), one symbol at a time, and writing its output on the same tape as the reading continues. The machine has a head that executes the reading and writing. The head is positioned on a specific cell of the tape and may move left (denoted L) or right (denoted R) one cell at a time. The computing is done via a transition function that tells the machine how to react to the symbols on the tape.

Formally, a Turing machine is a 7-tuple $(Q, q_0, \Gamma, b, \Sigma, F, \delta)$ where

- Q is a finite set of states with one of them $q_0 \in Q$ being a designated starting state (this is the state the machine starts its operation in).
- Γ is a finite set of symbols with one of them $b \in \Gamma$ being a designated blank symbol (writing b on the tape does not change the tape's content).
- $\Sigma \subset \Gamma$ is a subset of input symbols
- $F \subset Q$ is a subset of accepting states which finalizes the computation (when the machine reaches F, the computing has finished.
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{\mathtt{L}, \mathtt{R}\}$ is a partial[3] transition function (if the machine reaches a state and input that are not defined for δ, the machine hangs).

The machine, at state $q \in Q$, reads the current symbol $\gamma \in \Gamma$ on the tape leading to $\delta(q, \gamma) = (q', \gamma', d)$ where $q' \in Q$ is the next state, $\gamma' \in \Gamma$ is the output symbol being written by the head on the tape, and $d \in \{\mathtt{L}, \mathtt{R}\}$ indicates the movement of the head (left or right) on the infinite tape.

Consider the following example of a Turing machine that is supposed to process a binary tape to the right of the starting position and halt execution when the number of ones reaches 10. In this case we have $Q = \{q_0, q_1, \ldots, q_{10}\}$, q_0 is the starting state, $F = \{q_{10}\}$, $\Gamma = \{0, 1, b\}$, $\Sigma = \{0, 1\}$ and δ defined as follows $\delta(q_i, 0) = (q_i, b, \mathtt{R})$ and $\delta(q_i, 1) = (q_{i+1}, b, \mathtt{R})$. The machine reads the input left to right, does not modify the tape (since it writes blanks) and halts computation after reading 10 1-symbols (it transitions at this point to the finishing state q_{10}).

As indicated above, the tape serves the purpose of input, output, and memory while the state of the machine $q \in Q$ is a different form of memory - called control. The control, or state, is hard-wired into the machine in a similar way to hardware. The tape, on the other hand, may change from one operation of the machine to another. In other words, the control defines what the machine does like the program

[1]The notion of equivalency can be formalized to mean that a Turing machine can do whatever a standard computer can do and vice verse

[2]Turing machine have infinite memory and therefore they are actually equivalent to computers with infinite memory

[3]A partial function is not necessarily defined on all the domain

code of a routine. The tape serves as input, output and memory (or scratch pad to keep track of the current computation). Note that while the tape is infinite, Q is finite. This means that the Turing machine corresponds to a finite program - but a program that has at its disposal an infinite ream of scratch paper.

The example shows how difficult it is to design a Turing machine that will do a specific task. This is why programming languages, which are a much more convenient way to design a computation, are used instead of the Turing machine formalism in practice. On the other hand, if one wants to analyze the theoretical properties of computers, the Turing machine is a convenient formalism. It is rather simple and does not change as computers do when you change the software or hardware.

If a Turing machine can accomplish the task, the task is called computable. If a Turing machine cannot accomplish the task, the task is called non-computable. The study of these classes of problems is the topic of the theory of computability. The time it takes a Turing machine to finish its computation is called its time complexity. The amount of tape it uses is called the space complexity. These study of space and time complexities of different problems is the topic of the theory of complexity.