Turing Machines

A function $f : \mathbf{Z} \to \mathbf{Z}$ is "effectively computable" if there is an "algorithm" takes as input any integer x and halts in a finite number of steps, returning $f(x)$. It is easy to see that the "cardinality" of the set of all functions is the same as that of the reals; but if one assumes that an "algorithm" means a finite piece of code (over a finite alphabet), then the cardinality of the set of algorithms is the same as that of the integers (even forgetting the requirement of terminating on all inputs). So by Cantor's argument (which we recall), there are functions which are not computable (even if we hypothetically allow non-terminating algorithms.)

Definition of Turing Machines A TM has one read/write tape. It is in a current state and is reading a symbol on the tape. Based only on these two pieces of information, it goes to a next state, changes the current symbol to another (possibly the same) one and moves its head to the right or left.

A single operation is very simple. Also step-counting, (measuring running time) is simple.

TM 's are robust : TM's with multiple tapes can be simulated by single tape machines. Many other modifications like multiple heads on the same tape, different write-alphabet, ability to stay at the same cell (in addition to possibly moving right / left) etc. all can be simulated by the original defintion of TM.

A language is called **recursively enumerable (or r.e.)** if there is a TM accepting it. Are there languages which are not r.e. ?

Definition of non-deterministic Turing Machines. The class of sets accepted by non-det TM's are precisely the r.e. sets.