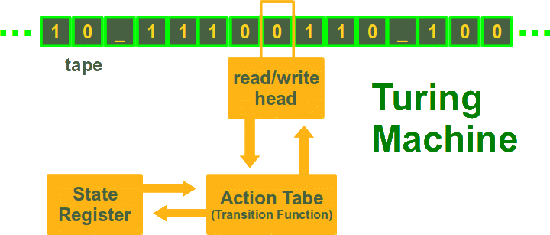
**Turing Machine**



Just to let you know straight-away: The Turing machine is not a machine. It is a mathematical model, which was formulated by the English mathematician Alan Turing in 1936. It's a very simple model of a computer, yet it has the complete computing capability of a general purpose computer. The Turing machine (TM) serves two needs in theoretical computer science:

1. The class of languages defined by a TM, i.e. structured or recursively enumerable languages
2. The class of functions a TM is capable to compute, i.e. the partial recursive functions

A Turing machine consists only of a few components: A tape on which data can be sequentially stored. The tape consists of fields, which are sequentially arranged. Each field can contain a character of a finite alphabet. This tape has no limits, it goes on infinitely in both directions. In a real machine, the tape would have to be large enough to contain all the data for the algorithm. A TM also contains a head moving in both directions over the tape. This head can read and write one character fro

m the field over which the head resides. The Turing machine is at every moment in a certain state, one of a finite number of states. A Turing program is a list of transitions, which determine for a given state and character ("under" the head) a new state, a character which has to be written into the field under the head and a movement direction for the head, i.e. either left, right or static (motionless).

**Formal Definition of a Turing machine**

A deterministic Turing machine can be defined as a 7-tuple  
  
M = (Q, Σ, Γ, δ, b, q0, qf)  
  
with

* Q is a finite, non-empty set of states
* Γ is a finite, non-empty set of the tape alphabet
* Σ is the set of input symbols with Σ ⊂ Γ
* δ is a partially defined function, the transition function:  
  δ : (Q \ {qf}) x Γ → Q x Γ x {L,N,R}
* b ∈ &Gamma \ Σ is the blank symbol
* q0 ∈ Q is the initial state
* qf ∈ Q is the set of accepting or final states

### Example: Binary Complement function

Let's define a Turing machine, which complements a binary input on the tape, i.e. an input "1100111" e.g. will be turned into "0011000".  
Σ = {0, 1}  
Q = {init, final}  
q0 = init  
qf = final

|  |  |
| --- | --- |
| **Function Definition** | **Description** |
| δ(init,0) = (init, 1, R) | If the machine is in state "init" and a 0 is read by the head, a 1 will be written, the state will change to "init" (so actually, it will not change) and the head will be moved one field to the right. |
| δ(init,1) = (init, 0, R) | If the machine is in state "init" and a 1 is read by the head, a 0 will be written, the state will change to "init" (so actually, it will not change) and the head will be moved one field to the right. |
| δ(init,b) = (final, b, N) | If a blank ("b"), defining the end of the input string, is read, the TM reaches the final state "final" and halts. |

## Implementation of a Turing machine in Python

## We implement a Turing Machine in Python as a class. We define another class for the read/write tape of the Turing Machine. The core of the tape inside the class Tape is a dictionary, which contains the entries of the tape. This way, we can have negative indices. A Python list is not a convenient data structure, because Python lists are bounded on one side, i.e. bounded by 0. We define the method \_\_str\_\_(self) for the class Tape. \_\_str\_\_(self) is called by the built-in str() function. The print function uses also the str function to calculate the "informal" string representation of an object, in our case the tape of the TM. The method get\_tape() of our class TuringMachine makes use of the str representation returned by \_\_str\_\_. With the aid of the method \_\_getitem\_\_(), we provide a reading access to the tape via indices. The definition of the method \_\_setitem\_\_() allows a writing access as well, as we can see e.g. in the statement self.\_\_tape[self.\_\_head\_position] = y[1] of our class TuringMachine implementation. The class TuringMachine: We define the method \_\_str\_\_(self), which is called by the str() built-in function and by the print statement to compute the "informal" string representation of an object, in our case the string representation of a tape.

## BB0100100110011000110001BB