# Turing Machine

Artemii Yanushevskyi

**Abstract**

The second half of the twentieth century is characterized by a burgeon of technology. It was influenced primarily by mathematicians who tried to relate non-mathematical objects, such as mechanisms, with purely mathematical, like mathematical logic. Allan Turing was asked, what is a 'mechanical' process, he replied, it is the 'process' which can be done by a machine. It is a reasonable answer, yet how would we define a 'machine' then? Turing decided to elaborate such general notion of a machine. This question contributed to the appearance of machine that we call in his name: Turing machine.

## 1 Intuitive definition

**L**et's assume that we have a creature that can observe a single place on infinite tape. Every place has either 1 or 0 written on itself. This creature can move the tape to left or right and rewrite a digit in front of it. Additionally, the creature has some states, which include terminal state 'q0'. This creature acts upon what state it currently holds and what it is seeing on a tape at the moment.

We will call that entire system *Turing machine*.

The input is a tape. Although the tape is infinite, it should have just a finite number of 1. We can represent each vector $x \in \mathbb{N}^k$ as a string of $k$ blocks with $(x_i + 1)$ 1. Those blocks are separated with 0.

EXAMPLE. The vector $x = (3,\ 2,\ 1,\ 0) \in \mathbb{N}^4$ is represented on a tape

$$\ldots 0001111011101101000 \ldots$$

The machine performs basic operations: check a number on a tape, move left or right, and change its state. The rules which guide the machine are presented as a list of commands, which is called a Turing code. What the creature will do next depends on two factors: what it sees on a tape at the moment and what state it has.

EXAMPLE. Now we will make a single command of the machine. After, we will write a command that corresponds to it. In case creature

1. has the state q3 and
2. sees 0 on a place,

the creature should:

1. change a number on a place from 0 to 1
2. move a strip left in order to face with a place to the right side
3. change a state to q3.

The command that defines this process is the following:

$$(\mathrm{q2}, 1) -> (\mathrm{q2}, 1, \mathrm{R})$$

Usually the starting position is a first 1. The starting state is q1, we will call it terminal state. Below is an example of code, that adds $a$ and $b$. The input vector $(a,\ b) \in \mathbb{N}^2$.

```
( q1 ,  1) -> ( q1 ,  1 , R)
( q1 ,  0) -> ( q2 ,  1 , R)
( q2 ,  1) -> ( q2 ,  1 , R)
( q2 ,  0) -> ( q3 ,  0 , L)
( q3 ,  0) -> ( q0 ,  0 , L)
( q3 ,  1) -> ( q4 ,  0 , L)
( q4 ,  1) -> ( q5 ,  0 , L)
( q5 ,  1) -> ( q5 ,  1 , L)
( q5 ,  0) -> ( q0 ,  0 , S)
```

The result line will consist of $(a+b+1)$ 1 in a single row, which represents $a+b$.

**Remark.** *When machine acquires state* q0*, it stops. Afterward, we can read the output row.*

## 2 Mathematical definition

The intuitive definition given above boils down to a particular set of functions.

**Definition 2.1.** *Turing machine is a map*

$$M : \{\text{q0},\ \ldots\ ,\ \text{q}n\} \times \{1,\ 0\} \to \{\text{q0},\ \ldots\ ,\ \text{q}n\} \times \{1,\ 0\} \times \{L,\ S,\ R\}$$

*where* $\{\text{q0},\ \ldots\ ,\ \text{q}n\}$ *– is a set of states, 0 and 1 is an alphabet of the machine, and each element is a command.*

We say that function can be computed by a Turing machine if there is a Turing machine which has the same input-output set as the function does. It has been demonstrated that any algorithm (in the intuitive sense) can be presented as some Turing machine.

## 3 A model of Turing machine

It is unbelievable that all algorithms and function, regardless of their complexity, can be represented in such easy and intuitive 'mechanical' model. Turing machine is regarded as an ultimate computing machine.

This mechanical process of computing a function attracted my interest and I built a real model of an abstract Turing machine, the only difference is that a computer RAM memory restricts the size of a tape. 8 GB of RAM can contain $8e + 9$ long tape.

The application consists of a few different modules. The most important part is `TuringCore.py` where all actions by machine are being performed. Other modules deal with design, input data processing, visualization, etc. Turing created his machine hypothetically, but now we can operate with its real analog.

## 4 How to work with application

The program requires Python 3.5. To run the app open `TuringMachie.py` in IDLE and click 'Tun Module' in the top menu. The navigation window wit pop up. The top element is a navigation bar; it has three indicators. This bar is made to assist a user through the process. A first indicator lights up when a user selects a Turing code. Second, indicates that Turing machine has a compiled version of the code. Later, it will be executed on input numbers. The last indicator turns on after we choose entering values.

The next section is a where we gather all information that is needed to run the machine: Code and Values. After the program has all the data, each indicator should turn green. It means that we are ready to start the machine. Now we want to click 'Run' button.

After a Turing program have executed, we can navigate on a tape with Left and Right button to read the result.

# 5 Further research on Turing Machines

## 5.1 Turing set

**E**ach Turing machine is defined by its code. By selecting different input values, we obtain some output. There are two cases either this result is a tape with a finite number of 1 and 0; or machine could never stop. Either way, we have an infinite set $\{(input,\ output), \forall\ input\}$. Although this set equally defines a Turing machine, this set is not finite. We can't store this set in computer memory.

**Definition 5.1.** *The set* $\boldsymbol{T} \subset \bigcup_{k=1}^{\infty} \mathbb{N}^k$ *is called a* ***Turing set*** *if there exists a Turing program whose output set equals to* $\boldsymbol{T}$*.*

We can represent an infinite set of vectors with some finite *Turing code*. This set has to be a *Turing set*.

## 5.2 Acquire a Turing code

**G**iven some Turing machine $M$. Its code is unknown to us. But we can still use the machine.

Let $\boldsymbol{I} \subset \bigcup_{k=1}^{\infty} \mathbb{N}^k$ be an *input* set.

Based on some set of results $\{(input,\ output) :\ \forall\ input \in \boldsymbol{I}\}$ we might draw some assumptions on what the machine actually does. We can definitely build a machine $\hat{M}$. Can we guarantee that machine $M$ will be identical to its emulation $\hat{M}$? Another question is: will $M$ and $\hat{M}$ have the same result on $\bigcup_{k=1}^{\infty} \mathbb{N}^k$?

Before formulating a hypothesis, we need a definition.

**Definition 5.2.** *Turing machine is called* ***irreducible*** *if we can not obtain another Turing machine which has the same input-output set but with shorter Turing code.*

HYPOTHESIS: If $M$ and $\hat{M}$ are *irreducible*, their Turing code have the same length $n$, and they produce the same result on $\{1, \ldots, 2^n\} \subset \mathbb{N}$. Then $M$ is equivalent to $\hat{M}$.

## 5.3 Possible Application in Artificial Intelligence

**I** remember playing in the game. We had four teams and 2-4 members of each team. The rules were: each team has to write a coherent story, using words that were given one per minute (those words were completely unrelated). The winner is a team with the most interesting story. It also should make sense.

My team had two players. It was just a girl and me. We were trying our best to relate those words in our story. Somewhere at the end of the game, the word 'temple' came up. It didn't fit in our fairy-tale at all. I began to build the most feasible version of further developing of our story. And I proposed it.

Obviously, the way I made the word fit in a story sounded quirky. My teammate didn't like it, so she started to think about continuation aloud. She precisely repeated my chain of thoughts and *arrived at the same result* as I did. We ended up having the same continuation of our story, despite we thought independently

It may be just a coincidence, or we think in a similarly. To the extent when we tend to align our story to the new word in the same way. It may demonstrate that we had the same comprehension of the game rules and it may be a sign that we have a similar mindset. Now, what does that have to do with AI?

If only we can make some robot to comprehend rules and enrich its mindset with ideas and experience, we can expect him to make identical conclusions that a human would do.

With that being said, let's consider some Turing machine with unknown code. We can draw an analogy between *an original code* of a Turing machine and *mindset* of a human. Our task is to obtain a code. Once it is done, we are able to run this code to predict what Turing machine would do.

As it was mentioned in Acquire a Turing code section, we may build at least some approximation of a Turing machine based on its input-output data.

> FACEBOOK LIKES ANALOGY. There is a person that sometimes hits on a like button under some posts. Wouldn't that be nice to supply posts that the person is more likely to 'like'?
>
> In order to do that we need to create a somewhat accurate virtual representation of that particular person. And then test all posts on their 'likeability' on the model, before supplying it to a real person. If

an accurate model likes a post, chances are a person would like it.

We can build a model of a real person by analyzing input-output data. In that case, the data is presented in this way:

$$\{(post,\ like):\ post \in Posts,\ like \in \{Yes,\ No\}\}.$$

Describing a post in some terms is necessary. We want to distinguish them from each other, likewise to point out similarities between them.

After that is done, we would like to convert this data set in Turing input-output set. Based on that set, we will create a Turing code, in the way it was explained in Turing set Acquire a Turing code section.

This will be a Turing model of our Facebook user.

**Remark.** *Even though an emulation of our behavior with Turing machines doesn't appear like an accurate representation of humans with all our enormous complexity, I still believe it is a good start.*