

Turing Machines

Neither finite nor pushdown automata are powerful enough to serve as general models of computation. *Turing machines* (TMs) provide such a general model.

They are formal models of computation that capture the capabilities of *any* algorithm.

We will use them to show that if a TM cannot solve a particular problem (in a finite amount of algorithmic steps), then they are *unsolvable* no matter what type of computer we have available to us.

Turing Machines

The *Church-Turing thesis* says that *Every computable algorithm can be implemented as a TM*, i.e. TMs represent *all conceivable algorithms*.

That is, we can *enumerate* them. Using a diagonalisation argument, we can show that the set of all functions cannot be enumerated. Therefore, some functions cannot be computed (in finite time).

Note also that if TMs can compute *anything*, they must be able to operate on their own codes. Again, using diagonalisation, we can construct a TM that halts and does not halt on its own code! In so doing, we will expose a large class of non-algorithmic computational problems:

Algorithms are not capable of recognizing properties of functions computable by algorithms!

Turing Machines

TMs consist of three components:

1. an infinite length of tape;
2. a tape head;
3. a control unit containing a finite amount of states.

TMs extend the idea of linear bounded automata (LBA) by having an infinite length of tape. Let's use the symbol \triangleright to mark the leftmost bound of the tape beyond which the tape head cannot move and which cannot be overwritten. There is no rightmost bound on the tape.

As for LBA, TMs will perform one of the actions $A \in \{Y, N, L, R\}$ at each step, where:

- Y denotes "Yes", accept the input string
- N denotes "No", do not accept the input string
- L denotes "Left", move the read-write head one space to the left
- R denotes "Right", move the read-write head one space to the right

Formal Definition of Turing Machines

A Turing machine is a 5-tuple $M = (Q, \Sigma, \Gamma, q_0, \delta)$, where:

- Q is a finite set of states.
- Σ is an alphabet (*input symbols*).
- Γ is an alphabet (*store symbols*).
- $q_0 \in Q$ is the *initial state*.
- δ , the *transition function*, is from $Q \times (\Gamma \cup \{\triangleright\})$ to $Q \times (\Gamma \cup \{\triangleright\}) \times \mathcal{A}$.

If $((q, a), (q', b, action)) \in \delta$, then when in state q with a at the current read position on the tape, M may replace a with b on the tape, perform the specified *action*, and enter state q' .

Let's use the symbol " $\#$ " to denote a blank square on the tape.

M accepts $w \in \Sigma^*$ iff it starts with configuration $(q_0, w\#)$ and the action Y is taken.

An Example

00 → **00R**
01 → **131L**
10 → **651R**
11 → **10R**
20 → **01R.STOP**
21 → **661L**
30 → **370R**
 ...
 ...
2100 → **31L**
 ...
 ...
2581 → **00R.STOP**
2590 → **971R**
2591 → **00R.STOP**

01 → **131L** means if the device is in internal state **0** and reads **1** on the tape, then it *must* change to internal state **13**, leave the **1** as a **1** on the tape, and move one square along the tape to the left.

An Example

00 → **00R**
01 → **131L**
10 → **651R**
11 → **10R**
20 → **01R.STOP**
21 → **661L**
30 → **370R**
 ...
 ...
2100 → **31L**
 ...
 ...
2581 → **00R.STOP**
2590 → **971R**
2591 → **00R.STOP**

2591 → **00R.STOP** means if the device is in internal state **259** and reads **1** on the tape, then it *must* change to internal state **0**, erase the **1** to produce a **1** on the tape, move one square along the tape to the right, and terminate the calculation.

In such cases, we can rewrite **R.STOP** as **STOP** ...

Binarising the Example

00 → 00R
 01 → 11011L
 10 → 10000011R
 11 → 10R
 100 → 01STOP
 101 → 10000101L
 110 → 1001010R
 ...
 ...
 110100100 → 111L
 ...
 ...
 1000000101 → 00STOP
 1000000110 → 11000011R
 1000000111 → 00STOP
 ▷ ... ##1111010011100100101101## ...

11010010	0
----------	---

▷ ... ##1111010011100101101101## ...

11	0
----	---

Functions on Natural Numbers

Represent numbers in unary notation using only the symbol 1 (zero is represented by the empty string).

i.e. $1 \rightarrow 1, 2 \rightarrow 11, 3 \rightarrow 111$ etc.

Let's describe the TM **UN+1** which adds one to a unary number:

0# → 0#R
 01 → 11R
 1# → 01STOP
 11 → 11R

Example: $f(n) = n + 1$ for each $n \in \mathbb{N}$.

<i>State</i>	<i>Symbol</i>	$\delta(\textit{State}, \textit{Symbol})$
q_0	1	$(q_0, 1, R)$
q_0	#	$(q_0, 1, Y)$

$(q_0, \underline{11}\#) \vdash_M (q_0, \underline{11}\#) \vdash_M (q_0, \underline{11}\#) \vdash_M (q_0, \underline{11}\#) \vdash_M (q_0, \underline{11}\#)$

▷ ... ##1111## ...

Functions on Natural Numbers

Here's the TM UNx2:

0# → 0#R
 01 → 1#R
 1# → 101L
 11 → 11R
 10# → 11#R
 101 → 100#R
 11# → 01STOP
 111 → 111R
 100# → 1011L
 1001 → 1001R
 101# → 101L
 1011 → 1011L

which *doubles* a unary number.

Question : rewrite this TM as the 5-tuple

$$M = (Q, \Sigma, \Gamma, q_0, \delta).$$

Functions on Natural Numbers

Here's a TM that adds two unary numbers:

It is assumed that the initial configuration of the machine is $(a, \underline{m}\#n\#)$, where m and n are the two numbers to be added.

<i>State</i>	<i>Symbol</i>	$\delta(\textit{State}, \textit{Symbol})$
a	1	$(b, \#, R)$
a	#	$(a, \#, Y)$
b	1	$(b, 1, R)$
b	#	$(c, \#, R)$
c	1	$(c, 1, R)$
c	#	$(d, 1, L)$
d	1	$(d, 1, L)$
d	#	$(e, \#, L)$
e	1	$(e, 1, L)$
e	#	$(a, \#, R)$

Extensions

The following extensions do not increase the power of Turing Machines.

- 2-way infinite tape
- Multiple tapes
- Multiple heads on one tape
- Two-dimensional “tape”
- Non-determinism

Universal Turing Machines

In order to be able to study the limits of Turing machines, we need a general method of specifying them, so that their specification can be used as input to other Turing machines (if we regard a Turing machine as a program, then we are treating programs as data).

We can create a *Universal Turing machine*:

- input is a specification for a Turing machine M and a string w .
- output is the output from M on w .

Thus, for a universal Turing machine U , we must have:

$$U(M, w) = M(w)$$

The Halting Problem

Problem: Given a Turing Machine M and a string w , will M halt when run on w ?

If there is a Turing machine M_0 that solves this problem, then an algorithm to convert a Turing-accepting machine M_1 to a Turing-deciding machine is:

- Feed M_1 and its input w to M_0 .
- If the answer is “halts” then output Y .
- Otherwise, output N .

This crucial question is called the *Halting Problem*.

The Halting Problem

There is no program to solve the halting problem: problems where halting cannot be guaranteed are *undecidable*.

A TM that halts for all inputs is *polynomially bounded* iff there is a polynomial $P(n)$ such that:

$$f_{TM}(n) \leq p(n).$$

We use the notation \mathbf{P} for all problems that can be solved by a polynomially bounded TM.

A problem is *intractable* if it is not in \mathbf{P} .