POWER OF TURING MACHINE

American University of Beirut

DECEMBER 31, 2019

DR. JAD MATTA
aub

There is an effective procedure for solving a problem if and only if there is a turing machine that halts for all input and solves the problem.

It has never been proved that there exist a more powerful machine than a Turning Machine

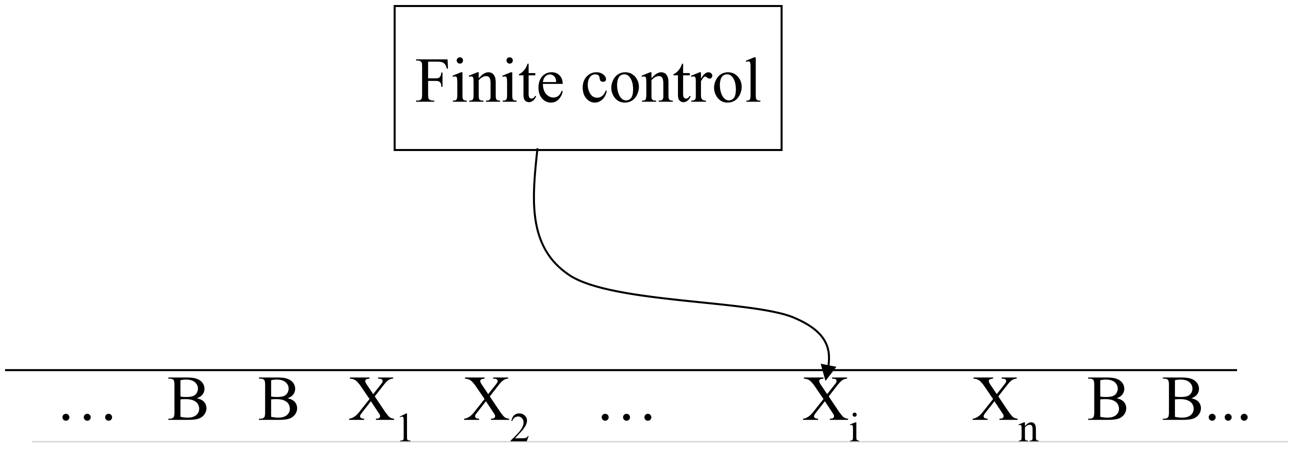
Format definition of a Deterministic Turing Machine

A DTM is a seven-tuple:

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

Q	A finite set of states
Σ	A finite input alphabet, which is a subset of $\Gamma – \{B\}$
Γ	A finite tape alphabet, which is a strict superset of Σ
B	A distinguished blank symbol, which is in Γ
q_0	The initial/starting state, q_0 is in Q
F	A set of final/accepting states, which is a subset of Q
δ	A next-move function, which is a *mapping* (i.e., may be undefined) from $Q \times \Gamma \rightarrow Q \times \Gamma \times \{L,R\}$

Intuitively, δ(q,s) specifies the next state, symbol to be written, and the direction of tape head movement by M after reading symbol s while in state q.



Definition: A language, L, is a set of strings, not necessarily finite in cardinality, which contains only finitely many characters. For instances, the encodings of graphs that are 3 colorable, or the encoding of binary integers.

Definition: (Decision Problem) a decision problem is the computational problem of deciding whether or not a string is within some language.

The fundamental ideas behind the Turing Machine are as follows:

1. Turing Machines (try to) answer decision problems

2. Turing Machines might or might not halt at all, but if it does halt, it gives an answer (Accept/yes or reject/no)

3. Turing Machines follow a fixed length algorithm

4. Turing Machines have an infinite block of memory, which initially only contains the input

5. Turing Machines can read and write to a single space in memory in a single operation although this is not precisely the same as a Turing Machine, it is equivalent in what it can compute, and how fast it can compute it. Interestingly enough, almost all reasonable models of computation are equivalent in this sense.

One limitation is that not every decision problem can be solved. It is not necessarily the case that a Turing Machine will always stop/halt. A Turing Machine decides a language if it accepts every string from the language, and rejects every string not in the language. A language is decidable, if some Turing Machine decides it.

A language is undecidable if no Turing Machine exists that decides it. Many undecidable languages exist, for example, the language HALT is undecidable, where halt is the language of Turing Machine, input pairs, such that the Turing Machine halts on the given input.

Nondeterministic Turing Machine

Deterministic Turing Machine is equivalent to accepting the same language as nondeterministic Turing. But, the deterministic Turing Machine might take exponentially more time than the nondeterministic Turing Machine.

Theorem: Since any deterministic Turing Machine is also nondeterministic, there exists a nondeterministic Turing Machine for every deterministic Turing Machine.

Equivalence of Turing Machines and Computers

In one sense, a real computer has a finite amount of memory, and thus is weaker than a Turing machine. But, we can postulate an infinite supply of tapes, disks, or some peripheral storage device to simulate an infinite Turing Machine tape.

Additionally, we can assume there is a human operator to mount disks, keep them stacked neatly on the sides of the computer.

Problem Statement: We need to show both directions, A Turing Machine can simulate a computer and that a computer can simulate a Turing Machine.

1. Computer Simulate a Turing Machine

Given a computer with a modern programming language such as C, C++, java or Python, certainly, we can write a computer program that emulates the finite control of the Turing Machine.

The only issue remaining is the infinite tape. Our program must map cells in the tape to storage locations in a disk. When the disk become full, we must be able to map to a different disk in the stack of disks mounted by the human operator.

2. Turing Machine Simulate a Computer

This simulation is performed at the level of stored instructions and accessing words of main memory.

- Turing machine has one tape that holds all the used memory locations and their contents
- Other Turing machine tapes hold the instruction counter, memory address, computer input file, and scratch data.
- The computer's instruction cycle is simulated by:

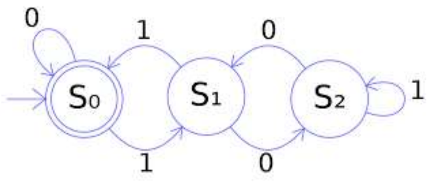
- Find the word indicated by the instruction counter on the memory tape.
- Examine the instruction code, and get the contents of any memory words mentioned in the instruction, using the scratch tape
- Perform the instruction, changing any words values as needed, and adding new address-value pairs to the memory tape, if needed.

Anything a computer can do, a Turing Machine can do, and vice versa. Moreover, a Turing machine is much slower than the computer but the difference is polynomial. Each step done in the computer can be completed in O (n^2) using a Turing Machine.

Whenever we define an algorithms to solve problems, we can equivalently talk about how to construct a Turing Machine to solve the problem. If a Turing Machine cannot be built to solve a particular problem, then it means our modern computer cannot solve the problem either.

Devices of Increasing Computation Power

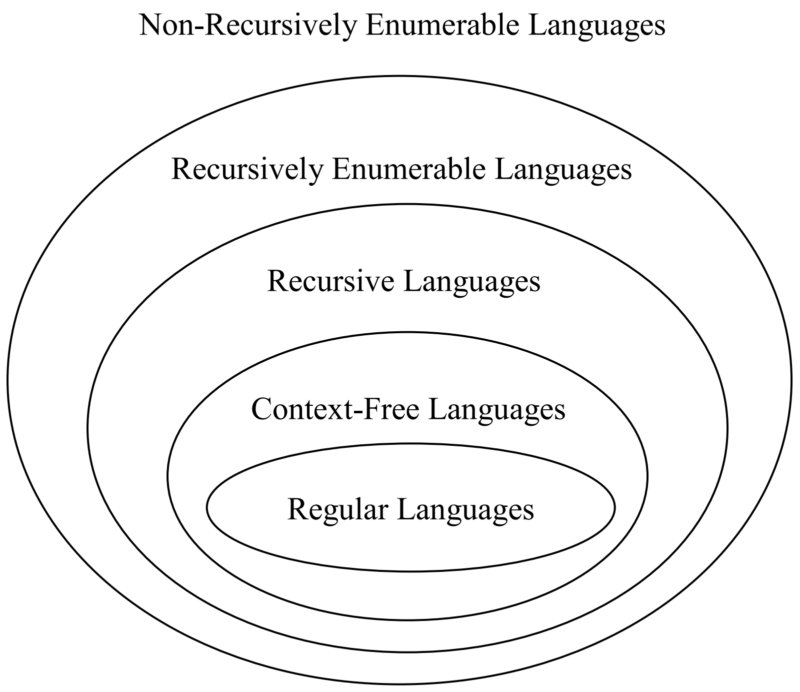
-- Finite Automata – good for devices with small amounts of memory, relatively simple control



– Pushdown Automata – stack-based automata

- But both have limitations for even simple tasks, too restrictive as general purpose computers
- Enter the Turing Machine
  - More powerful than either of the above
  - Essentially a finite automaton but with unlimited memory
  - Although theoretical, can do everything a general purpose computer of today can do
    - If a TM can't solve it, neither can a computer

Classes of languages determined by a Turing Machine


Non-Recursively Enumerable Languages
Recursively Enumerable Languages
Recursive Languages
Context-Free Languages
Regular Languages

Some Important Definitions

- **Definition**: Let M = (Q, Σ, Γ, δ, q_0, B, F) be a TM, and let w be a string in Σ*. Then w is *accepted* by M if and only if

$$q_0w \vdash^* \alpha_1 p \alpha_2$$

where p is in F and α_1 and α_2 are in Γ^*

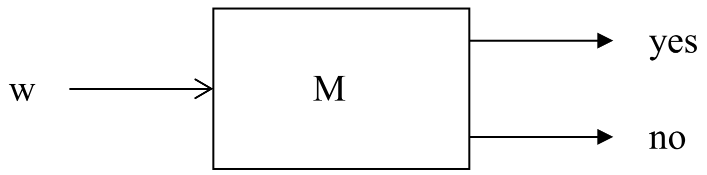
- **Definition**: Let M = (Q, Σ, Γ, δ, q_0, B, F) be a TM. The *language accepted by M*, denoted L(M), is the set

  {w | w is in Σ* and w is accepted by M}

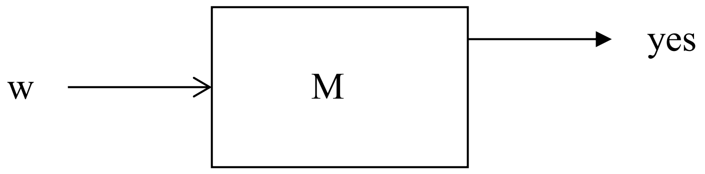
- **Definition:** Let *L* be a language. Then *L* is *recursively enumerable* if there exists a TM *M* such that L = L(M).
  - If *L* is r.e. then L = L(M) for some TM *M*, and
    - If *x* is in *L* then *M* halts in a final (accepting) state.
    - If *x* is not in *L* then *M* may halt in a non-final (non-accepting) state or no transition is available, or loop forever.
- **Definition:** Let *L* be a language. Then *L* is *recursive* if there exists a TM *M* such that L = L(M) and M halts on all inputs.
  - If *L* is recursive then L = L(M) for some TM *M*, and
    - If *x* is in *L* then *M* halts in a final (accepting) state.
    - If *x* is not in *L* then *M* halts in a non-final (non-accepting) state or no transition is available (does not go to infinite loop).

Turing Machine Block Diagrams:

- If *L* is a recursive language, then a TM *M* that accepts *L* and always halts can be pictorially represented by a "chip" or "box" that has one input and two outputs.



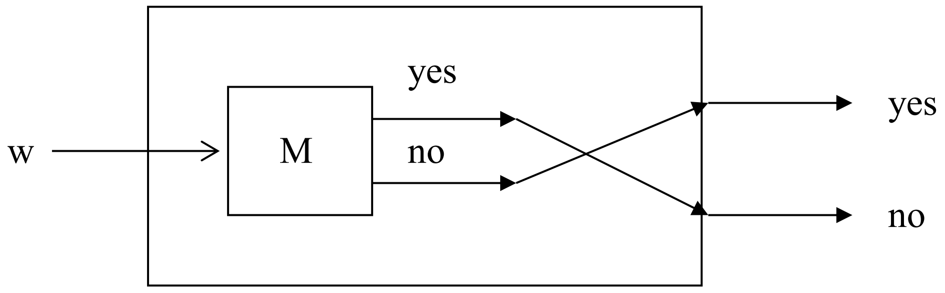
- If *L* is a recursively enumerable language, then a TM *M* that accepts *L* can be pictorially represented by a "box" that has one output.



- Conceivably, *M* could be provided with an output for "no," but this output cannot be counted on. Consequently, we simply ignore it.

- **<u>Theorem 1</u>:** The recursive languages are closed with respect to complementation, i.e., if *L* is a recursive language, then so is $\overline{L} = \Sigma^* - L$

- **Proof:** Let *M* be a TM such that L = L(M) and *M* always halts. Construct TM *M'* as follows:



- **Note That:**
    - *M'* accepts iff *M* does not
    - *M'* always halts since *M* always halts

  From this it follows that the complement of *L* is recursive.

- **<u>Theorem 2</u>:** The recursive languages are closed with respect to union, i.e., if L_1 and L_2 are recursive languages, then so is $L_3 = L_1 \cup L_2$

- **Proof:** Let M_1 and M_2 be TMs such that $L_1 = L(M_1)$ and $L_2 = L(M_2)$ and M_1 and M_2 always halts. Construct TM *M'* as follows:

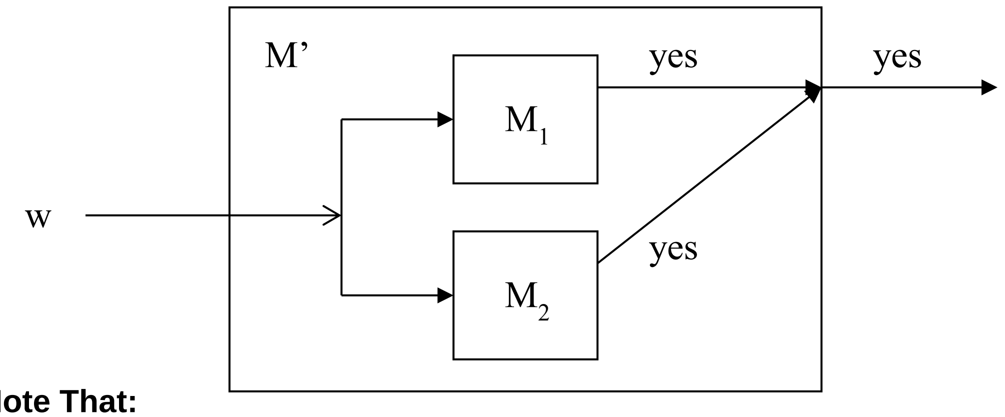
- **Note That:**
  - L(M') = L(M_1)   L(M_2)
    - L(M') is a subset of L(M_1) U L(M_2)
    - L(M_1) U L(M_2) is a subset of L(M')
  - *M'* always halts since *M_1* and *M_2* always halt

  It follows from this that is recursive

- **<u>Theorem 3</u>:** The *recursive enumerable languages* are closed with respect to union, i.e., if *L_1* and *L_2* are recursively enumerable languages, then so is

$$L_3 = L_1 \cup L_2$$

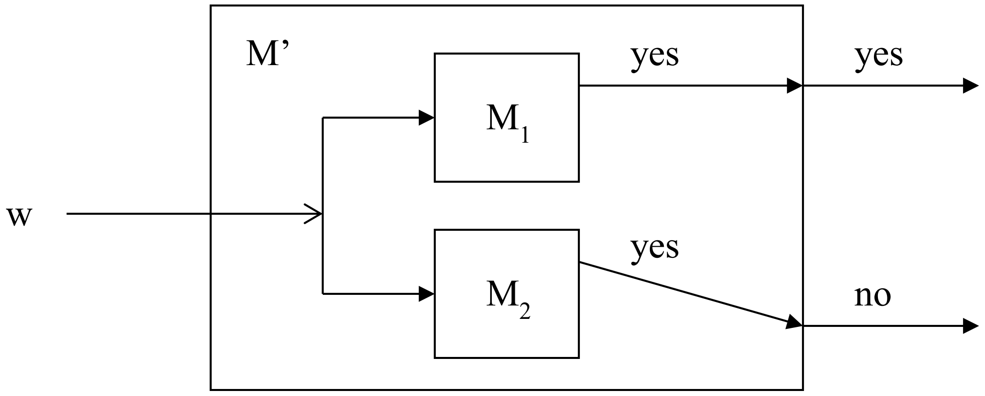
- **Proof:** Let *M_1* and *M_2* be TMs such that L_1 = L(M_1) and L_2 = L(M_2). Construct *M'* as follows



- **Note That:**
  - L(M') = L(M_1) U L(M_2)
    - L(M') is a subset of L(M_1) U L(M_2)
    - L(M_1) U L(M_2) is a subset of L(M')
  - *M'* halts and accepts if and only if *M_1* or *M_2* halts and accepts

    It follows from this that is recursively enumerable

- **<u>Theorem 4</u>:** If *L* and $\overline{L}$ are both recursively enumerable then *L* (and therefore $\overline{L}$) is recursive.
- **Proof:** Let M_1 and M_2 be TMs such that L = L(M_1) and $\overline{L}$ = L(M_2). Construct *M'* as follows:



- **Note That:**
  - L(M') = L
    - L(M') is a subset of *L*
    - *L* is a subset of L(M')
  - M' is TM for L
  - *M'* always halts since <u>either</u> M_1 <u>or</u> M_2 halts for any given string
  - M' shows that L is recursive

It follows from this that *L* (and therefore its' complement) is recursive.

So, $\overline{L}$ is also recursive.