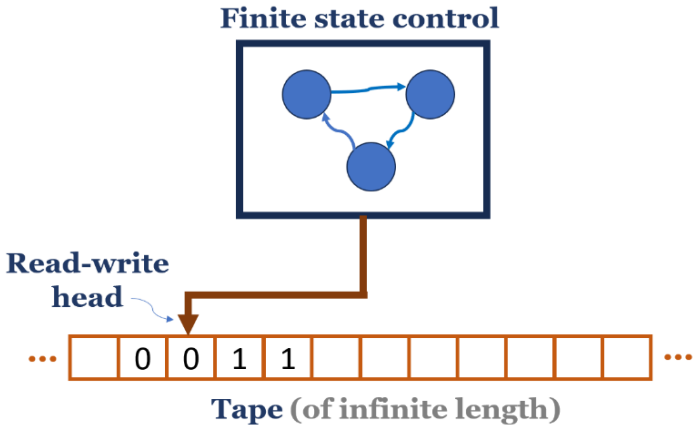
Department of Computer Science and Engineering
PES University, Bangalore, India

UE23CS242A: Automata Formal Language and Logic

Prakash C O, Associate Professor, Department of CSE

Turing Machine: The Foundation of Modern Computing

- **A Turing Machine is a theoretical computational model equivalent to a digital computer**.
- It was invented by the mathematician Alan Turing in 1936, and has been since then the most widely used model of computation in computability and complexity theory.
- Despite its simplicity, **the Turing machine can simulate ANY computer algorithm**, no matter how complicated it is!



Note: By convention, the input for computation is preloaded on the tape and if required special markers can be used to indicate the start and end of the input string.

The Turing Machine comprises of three main components:

1. **The Tape** (of infinite length),
2. **The Read-Write head**, and
3. **Finite state control (i.e., FSM)**.

The Tape:

- Turing machine consists of an infinitely-long tape which acts like the memory in a typical computer, or any other form of data storage.
- The tape is divided into squares(discrete cells), and each of which can hold a single symbol drawn from a finite set of symbols called the alphabet of the machine.
- A Turing machine can read, write, and erase symbols on an infinitely long tape.

The read-write head:

- The head is a device that can read, write and erase symbols on the cell individually and move left or right along the tape one cell at a time.
- Turing machine head can only read/write one symbol at a time, and the choice of which direction to move the head, and whether to halt is based on a finite table that specifies what to do for each combination of the current state and the symbol that is read.

Note:

- Like a real computer program, it is possible for a Turing machine to go into an infinite loop which will never halt.
- The problem with Turing Machines is that a different one must be constructed for every new computation to be performed, for every input output relation.
- In the context of automata theory and the theory of computation, **Turing machines are used to study the properties of algorithms and to determine what problems can and cannot be solved by computers**. They provide a way to model the behavior of algorithms and to analyze their computational complexity, which is the amount of time and memory they require to solve a problem.
- Alan Turing's groundbreaking work on the theoretical foundations of computation paved the way for the development of modern computing technology.

Formal Definition of Turing Machines

Formally, a deterministic one-tape Turing machine is a 7-tuple

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

where:

- **Q** is a finite non-empty set of states
- **Γ** is a finite non-empty set of tape symbols (which can be written on Tape)
- □ is blank symbol (every cell is filled with blank except input alphabet initially), and $\square \in \Gamma$
- **Σ** is the input alphabet (symbols which are part of input alphabet), and $\Sigma \subseteq \Gamma$
- **δ** is a transition function which maps $Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$.

  Depending on its **present state** and **present tape alphabet** (pointed by read-write head), it will move to **new state**, **change the tape symbol** (may or may not) and **move head to either left or right.**
- q_0 is the initial (or start) state
- **F** is the set of final (or accept) states. If any state of F is reached, input string is accepted.

Variants of a Turing Machine

- Multi-tape TMs.
- Non-deterministic TMs
- Multi-head TMs
- Single sided vs double sided infinite tape TMs
- Turing Machines with a Stay-Option (δ: $Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, S\}$)
- Turing machine with multiple tracks
- ...

Non-deterministic TMs

At any point in the computation, the TM may proceed according to several possibilities. Thus, the transition function has the form $\delta: Q \times \Gamma \rightarrow 2^{Q \times \Gamma \times \{L, R\}}$

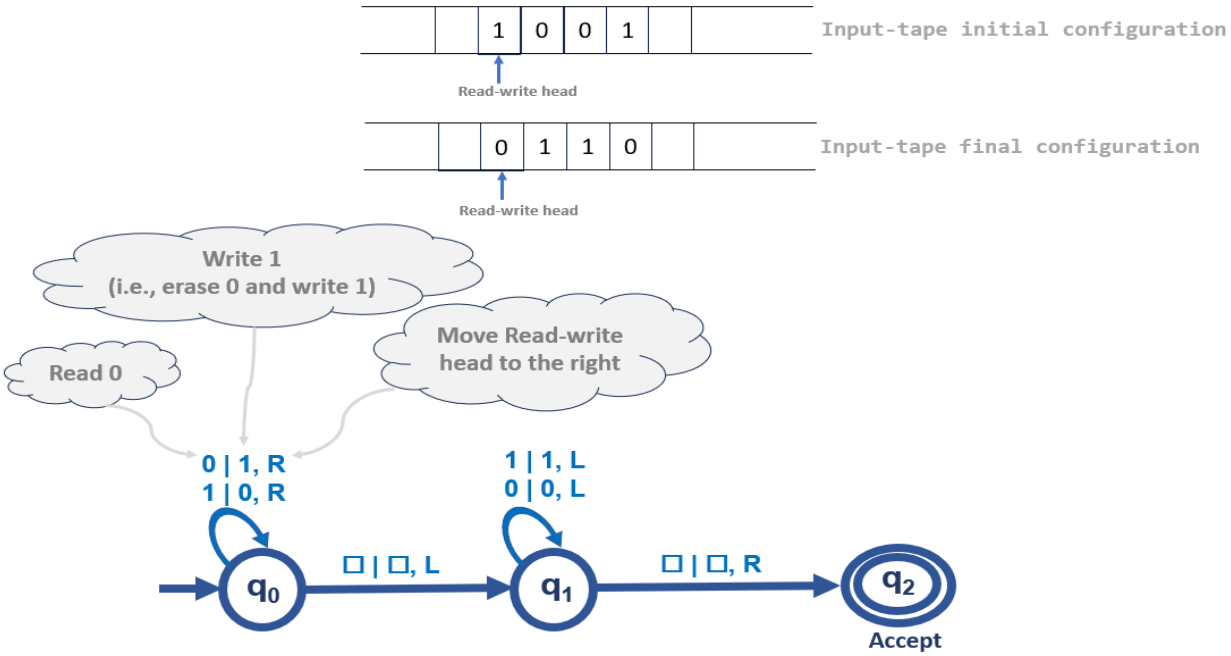
Deterministic TMs Examples:

1) Construct a Turing Machine as Transducer to convert a given binary number to its 1's complement and move the head back to the first symbol after complement.

Solution: High-Level Idea (or Logic)

1. Sweep from left to right, Turn zero to one and one to zero.
2. Move the head back to the first symbol after complement.

Transition Diagram:



Transition function δ is given in Table as:

δ	0	1	□
q_0	(q_0, 1, R)	(q_0, 0, R)	(q_0, □, L)
q_1	(q_1, 0, L)	(q_1, 1, L)	(q_2, □, R)
q_2*	-	-	-

Note:

- **Accepting states in Turing machine have no outgoing transitions.**
- **Other accepted transition notations: 0→1, L or 0, 1, L or 0|1|L or 0,1|L**

2) Construct a Turing Machine as decider for L = { 0^{2^n}, n>=0 }

Solution:

We want to accept, if and only if:

- the input string consists entirely of zeros, and
- the number of zeros is a 2 power (i.e., 1, 2, 4, 8, 16, 32, …)

High-Level Idea (or Logic).

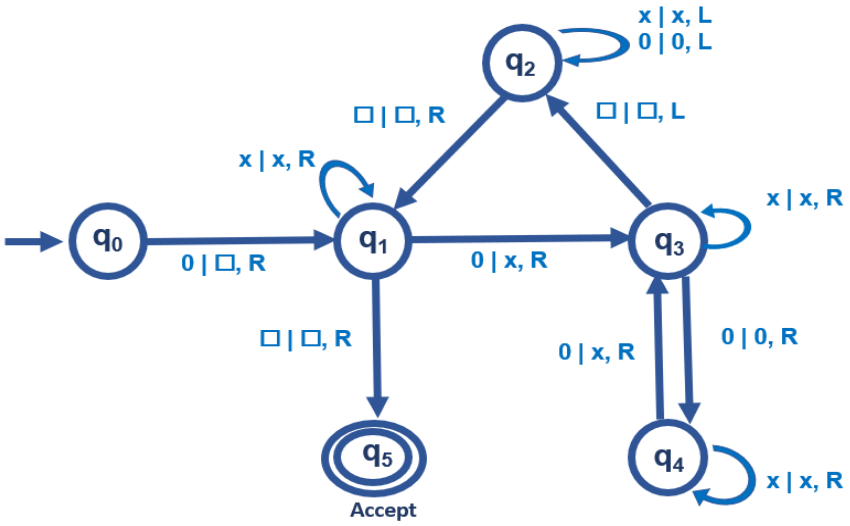
- Repeatedly divide the number of zeros in half (2 equal parts of zeros) until it becomes an odd number.
- If we are left with a single zero, then accept, Otherwise, reject.

Pseudocode:

1. Turn first zero to blank.

2. Sweep from left to right, cross out every other zero (i.e., alternate zero). (This divides number of zeros into half, including the first zero i.e., turned into blank)
    a. If the tape had only one 0 (including the first zero i.e., turned into blank), ***accept.***
    b. Else **if further equal division is not possible** and **the tape had an odd number** of **0**'s (including the first zero i.e., turned into blank), ***reject***.
3. Move the head back to the first input symbol. Repeat step-2 process

Transition Diagram:



Note: The Turing machine halts in a state if there is no transition to follow.

This is a state diagram corresponding to the pseudocode.

- Let's think about how the two correspond.
- We have an extra symbol "x" that is used to cross out symbols.

We start by writing the first 0 over with a blank.

- If there was just a single 0, we accept.
- In states q3 and q4, we cross out every other 0, until we see a blank.
- If we see a blank in state q4, then the number of 0's left was odd, so we reject.
- If we see a blank in state q3, then the number was even, so now in q2 we move left until we hit blank. We move back to q1

3) Construct a Turing Machine as decider for L = {$a^n b^m c^{n*m}$ | n, m ≥ 1}

Solution:

High-Level Idea (or Logic).

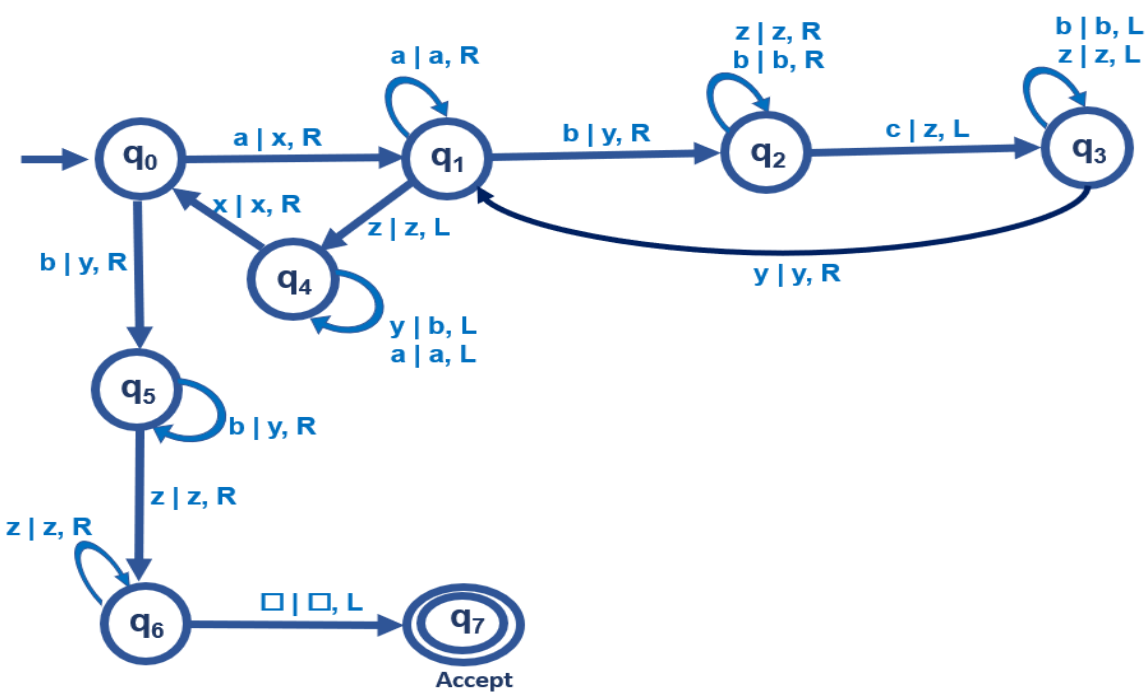
```
For each occurrence of a: {
        For each occurrence of b: {
                Replace an occurrence of c by z.
        }
}
```

Pseudocode:

1. Replace a by x, scan to the right until b.
2. Replace b by y, scan to the right until c.
        replace c by z, scan to the left until y. goto step 2
        If too few c's, reject.
3. Replace all y's by b.

4. If there's another 'a' left, then repeat step 1.
   If all a's are replaced by x,
      Check if all c's are replaced by z.
      If yes, then *accept*, else *reject.*

Transition Diagram:



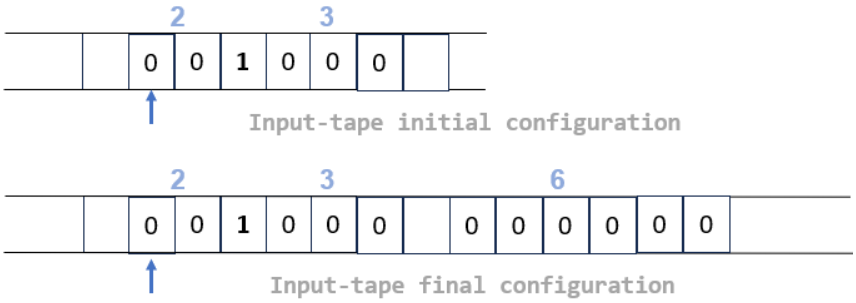
4) Construct a Turing Machine as Transducer for the following. Given an input: 0^m10^n, write 0^{m*n} as output on the tape (Language of Unary number multiplication).
Input and output must be separated by a blank cell (i.e., □).

Solution:

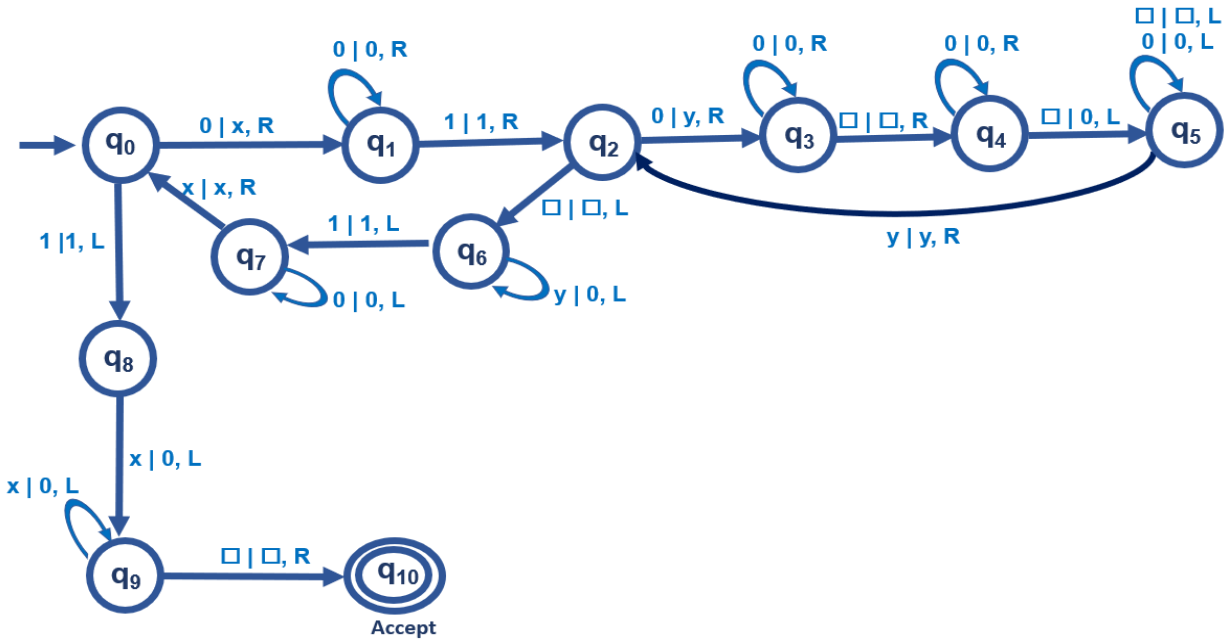
High-Level Idea (or Logic).

For each occurrence of 0 before 1: {
    For each occurrence of 0 after 1: {
        Write(after input leaving one blank) a digit 0 on tape i.e., replace blank by 0.
    }
}

Input-tape initial and final configuration.



Turing machine diagram (4 marks)



Input Acceptance in TM:

- Machine M accepts an input string, if M halts in an accept state.

Input Rejection in TM:

- If machine M halts in a non-accepting state, then it rejects the input string.
- If machine M enters an infinite-loop (runs forever), then also it rejects the input string.

Types of Halting in TM:

- **Halting in an accept state**, and then accepting the input string.
- **Halting in a non-accept state**, and then rejecting the input string.

Exercises:

Construct a Turing Machine as Transducer for the following

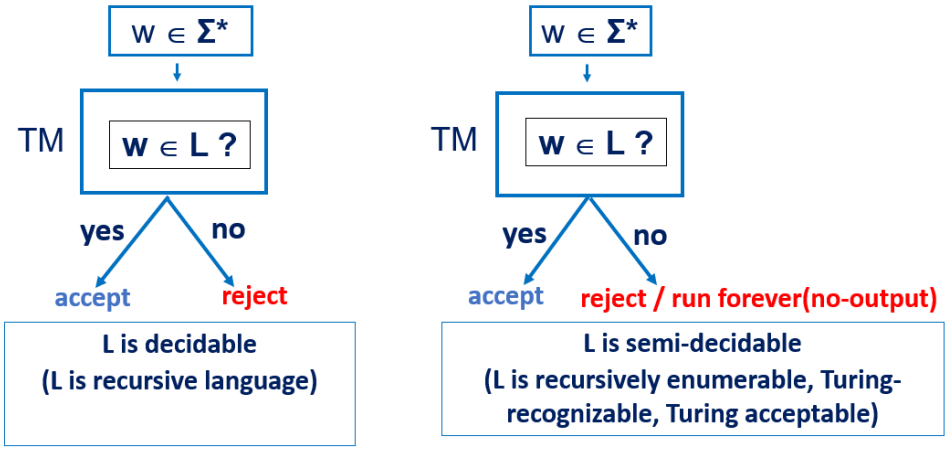
1. Given input: $\mathbf{0^m10^n}$, write $\mathbf{0^{m+n}}$ as output on the tape (**Language of addition**), m, n>=1
2. Given a string w ∈ {a,b}+, write **w** as output on the tape separated by a blank. Final contents on the tape must look like: **w□w** (**Copy operation**)
3. Given an input: $\mathbf{a^nb^m}$, Write the difference **n-m** (in terms of no. of c's) on the tape after input. If the difference is negative append a negative sign.
   For example: If k=3 write ccc on the tape.
   If k=-2, write -cc on the tape. (Language of subtraction).
   Input and output must be separated by a □.
4. Given a string w∈ {a,b}+, **sort the symbols in the input string**. For example if the input is babaa, output should be: aaabb. Final contents on the tape must look like: **babaa□aaabb**
5. Given a string w∈ {a,b}+, write $\mathbf{w^R}$ as the output on the tape (Reversing a string). Final contents on the tape must look like : $\mathbf{w□w^R}$

Construct a Turing Machine as decider for the following

1. L = { a^nb^n, n>=1 }
2. L = { $a^nb^nc^n$, n>=1 }

3. L = { $0^n1^{n\hat{}2}$, n>=1 }
4. L = { ww, w ∈ $\{a,b\}^+$ }
5. L = { w#w, w ∈ $\{0,1\}^*$ }
6. L = { $a^nb^mc^nd^m$, n>=1 }
7. L is a given regular language, say L = { (a + b)*b(a + b)* }.
   Tip: Simulate an DFA using a TM?
8. L is a given context-free language, say L = {w: w is a palindrome}
   Tip: Simulate an PDA using a TM?
9. L = {a^p | p is a prime number}

Turing-Decidable and Turing-Recognizable



Theorem: L is decidable, if both L and L^c are recursively enumerable

Turing-Decidable: **Turing Machines that halt on all inputs. The language recognized by a Turing Machine Decider is called Recursive Language.**

A language is "Turing-Recognizable" iff there exists a Turing Machine such that

- when encountering a string in that language, the machine terminates and accepts that string;
- when encountering a string not in that language, the machine either terminates and rejects that string or doesn't terminate at all.

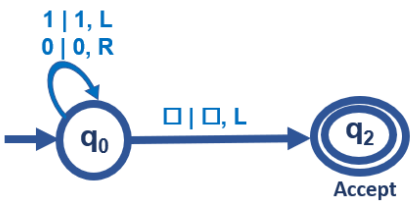
Note: there is no requirement that the Turing Machine should halt for strings not in the language.

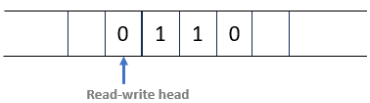
A language is "Turing-Decidable" iff there exists a Turing Machine such that

- when encountering a string in that language, the machine terminates and accepts that string;
- when encountering a string not in that language, the machine terminates and rejects that string.

Note that "Turing-Decidable" is a stronger condition than "Turing-Recognizable", because, if a language is Turing-Decidable then its corresponding Turing Machine never runs forever.

Note: Infinite Loop Example:





Encoding Turing machines as strings

- Every Turing machine can be encoded as a string in {0, 1}*, i.e., Just encode the description of the machine (in binary)!
- Every string in {0, 1}* is a TM
- Of course, some strings don't represent a valid encoding, then we can map them to a TM that does nothing.

Notation

< M >, a string representation of Machine M Description.

Universal Turing Machine

- In his 1936 paper "On Computable Numbers", Turing proved that we can build a Turing machine U_{TM} that acts as an interpreter for other Turing machines.
  In other words, U_{TM}'s input tape can contain a description of another Turing machine, which is then simulated step by step. Such a machine U_{TM} is called a Universal Turing machine.
- If a universal machine didn't exist, then in general we would need to build new hardware every time we wanted to solve a new problem: there wouldn't even be the concept of software.

U_{TM} – Universal Turing machine (has multiple tapes: Input, output, working, …)
(<M>, w) – Description of Turing Machine-M encoded in binary and an input string w.

U_{TM} simulate the behavior of Turing Machine M and runs on input w,

- U_{TM} accepts (<M>, w), if M accepts w.
- U_{TM} rejects (<M>, w), if M rejects w.
- U_{TM} runs for ever on (<M>, w), if M runs for ever on w.

In simple terms, a **Universal Turing Machine is like a "computer" that can simulate the behavior of any other Turing Machine**.

- It achieves this by reading the description of another Turing Machine from its input tape. This description (encoded in binary) specifies the states and transitions of the other Turing Machine and enables the Universal Turing Machine to simulate the behaviour of the other machine on its own tape. (i.e., U_{TM} simulates the behaviour of the read Turing machine)
- The Universal Turing Machine then reads the input tape for input w, and uses the simulated Turing Machine to compute the desired output.
- The output is then written onto the output tape, and the Universal Turing Machine stops.

A universal turing machine can thus simulate any other machine. Any problem that is not computable by the universal turing machine is considered to be non-computable.

Note:

- The Universal Turing Machine (U_{TM}) is considered one of the fundamental concepts in computer science and is the theoretical foundation of modern computing.

- U_{TM} demonstrates the power of a single, general-purpose computing device and is a testament to the concept of algorithmic computation.
- A Universal Turing Machine is similar to a regular Turing Machine but it has solutions to all problems that are computable.

Note: Turing completeness

- A computational system that can compute every Turing-computable function is called Turing-complete (or Turing-powerful). Alternatively, such a system is one that can simulate a universal Turing machine.

Questions on Universal Turing Machine

1. What can a Turing machine compute?

Solution: A Turing machine can theoretically compute anything that is computable. It can simulate the behavior of any algorithm or **solve any problem that has a well-defined algorithmic solution**. **This property is known as Turing completeness.**

2. What is the Halting Problem, and why is it important in Turing machine theory?

Solution: The Halting Problem is a fundamental problem in computer science and mathematics. It asks whether, **given a description of a Turing machine and its input**, can we determine whether the machine will eventually halt (terminate) or run forever on that input.

Alan Turing proved that the Halting Problem is undecidable.

Turing's efforts were instrumental in proving that **a general algorithm that solves the Halting problem for all possible program-input pairs cannot exist.** As such, he demonstrated the restrictions of computers, thus establishing a limitation on the power of mechanical computation.

3. Can a Turing machine simulate any modern computer?

Solution: Yes, in theory, a Turing machine can simulate any modern computer, as long as there is enough tape and time available. However, practical considerations, such as the enormous amount of tape needed for even simple computations, make this infeasible for complex computations.

The Church-Turing Thesis (aka computability thesis)

Related to the idea of universal machines is the so-called Church-Turing thesis, which claims that anything we would naturally regard as "computable" is actually computable by a Turing machine.

There are various equivalent formulations of the Church-Turing thesis.

1. A common one is that every effective computation can be carried out by a Turing machine.
2. Anything that can be computed by an algorithm can be computed by a Turing Machine.
3. What can be computed using Turing machine is known as computable.
4. For any computable problem, there is a Turing machine which computes it.

The Turing Test

The Turing test, one of the most discussed methods for assessing artificial intelligence (AI).

The Turing test, originally called the imitation game by Alan Turing in 1950, is a test of a machine's ability to exhibit intelligent behaviour equivalent to, or indistinguishable from, that of a human.

The Turing Test is a widely used **measure of a machine's ability to demonstrate human-like intelligence**.

References:

- https://www.cs.cmu.edu/~emc/flac09/reading.html
- https://web.stanford.edu/class/archive/cs/cs103/cs103.1142/lectures/18/Small18.pdf
- https://plato.stanford.edu/entries/turing-machine/#:~:text=Turing%20machines%2C%20first%20described%20by,the%20computing%20of%20real%20numbers.
- https://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/turing-machine/four.html
- https://web.mit.edu/manoli/turing/www/turing.html
- https://www.britannica.com/technology/Turing-test
- https://introcs.cs.princeton.edu/java/52turing/

Videos:

- A Turing Machine
  - https://www.youtube.com/watch?v=eZ13KTaQLts
- A Turing Machine - Overview
  - https://www.youtube.com/watch?v=E3keLeMwfHY
- A Turing Machine - Counting Program
  - https://www.youtube.com/watch?v=WJ-ODmFjmrU
- A Turing Machine - Busy Beaver 4-state
  - https://www.youtube.com/watch?v=2PjU6DJyBpw
- A Turing Machine - Subtraction
  - https://www.youtube.com/watch?v=aBToqFJLrl4
- LEGO Turing Machine
  - https://www.youtube.com/watch?v=FTSAiF9AHN4

END

Additional Info:

Formulations of Turing's thesis in terms of numbers

Computable numbers are (real) numbers whose decimal representation can be generated progressively, digit by digit, by a Turing machine.

Examples are:

- any number whose decimal representation consists of a finite number of digits (e.g., 109, 1.142)
- all rational numbers, such as one-third, two-sevenths, etc.
- some irrational real numbers, such as π and e.

Some real numbers, though, are uncomputable, as Turing proved. Turing's proof pointed to specific examples of uncomputable real numbers, but it is easy to see in a general way that there *must* be real numbers that cannot be computed by any Turing machine, since there are *more* real numbers than there are Turing-machine programs. There can be no more Turing-machine programs than there are whole numbers, since the programs can be counted: 1st program, 2nd program, and so on; but, as Georg Cantor proved in 1874, there are vastly more real numbers than whole numbers (Cantor 1874).

A (real) number is Turing computable if there exists a Turing machine which computes an arbitrarily precise approximation to that number. All of the algebraic numbers (roots of polynomials with algebraic coefficients) and many transcendental mathematical constants, such as *e* and π are Turing-computable. Turing gave several examples of classes of numbers computable by Turing machines

Note:
A rational number is a number that is expressed as the ratio of two integers, where the denominator should not be equal to zero, whereas an irrational number cannot be expressed in the form of fractions. **Rational numbers are terminating decimals but irrational numbers are non-terminating (endless) and non-recurring digits after the decimal point.** Value of $\sqrt{5}$ = 2.2360679775…….. It is a non-terminating value and hence cannot be written as a fraction. It is an irrational number.

https://plato.stanford.edu/entries/church-turing/
https://plato.stanford.edu/entries/turing-machine/#CompNumbProb

How Does the Turing Test Work?

The Turing Test is performed by placing a human in one room and a machine in another. Then a judge, or panel of judges, addresses each room with questions regarding any topic to which a human should be able to respond. If the machine passed Turing's test, it shows the machine's ability to process human syntax and semantics, which is thought to be a step toward creating artificial general intelligence.

Regardless of a computer's ability to pass the Turing Test, there is no real way for us to tell whether or not a machine truly understands human semantics. The test simply judges machines on their ability to converse with human-like eloquence, not human-like understanding. This limitation has led some AI researchers to argue the Turing Test is less relevant than it used to be.

Turing Test Example Questions

While there is no official list of Turing Test questions, a judge would likely ask questions that relate to human experiences like emotions and maturation, or linguistic riddles that could be difficult for a machine to parse. Here are some questions to ask if you find yourself judging a Turing Test:

- What is your most memorable childhood event and how has that impacted you today?
- Describe yourself using only colors and shapes.
- Describe why time flies like an arrow but fruit flies like a banana?
- How do you feel when you think about your upbringing and what makes you feel that way?
- What historical event changed you the most and where were you when it happened?
- Which of the previous questions was the most difficult to answer and why?

References:

https://builtin.com/artificial-intelligence/turing-test

What is a CAPTCHA?

A CAPTCHA test is designed to determine if an online user is really a human and not a bot. CAPTCHA is an acronym that stands for "Completely Automated Public Turing test to tell Computers and Humans Apart." Users often encounter CAPTCHA and reCAPTCHA tests on the Internet. Such tests are one way of managing bot activity, although the approach has its drawbacks.

Although CAPTCHAs are designed to block automated bots, CAPTCHAs are themselves automated. They're programmed to pop up in certain places on a website, and they automatically pass or fail users.

How does a CAPTCHA work?

Classic CAPTCHAs, which are still in use on some web properties today, involve asking users to identify letters. The letters are distorted so that bots are not likely to be able to identify them. To pass the test, users have to interpret the distorted text, type the correct letters into a form field, and submit the form. If the letters don't match, users are prompted to try again. Such tests are common in login forms, account signup forms, online polls, and e-commerce checkout pages.

At any one time, the machine has a head which is positioned over one of the squares on the tape and a "state" selected from a finite set of states. With this head, the machine can perform three very basic operations:

- Read the symbol on the square under the head.
- Edit the symbol by writing a new symbol or erasing it
- Move the tape left or right by one square so that the machine can read and edit the symbol on a neighbouring square.

At each step of its operation, the head reads the symbol in its cell. Then, based on the symbol and the machine's own present state, the machine writes a symbol into the same cell or the machine makes no changes on to the cell, and moves the head one step to the left or the right, or halts the computation.

Turing machine can only read one symbol at a time, and the choice of which replacement symbol to write, which direction to move the head, and whether to halt is based on a finite table that specifies what to do for each combination of the current state and the symbol that is read.

Note:

- Like a real computer program, it is possible for a Turing machine to go into an infinite loop which will never halt.
- The problem with Turing Machines is that a different one must be constructed for every new computation to be performed, for every input output relation.
- Turing Machine forms the foundation of theoretical computer science. Because of its simple description and behavior, it is amenable to mathematical analysis. This analysis has led to a deeper understanding of digital computers and computation, including the revelation that there are some computational problems that cannot be solved on computers at all, no matter how fast the processor, or how much memory is available.

- In the context of automata theory and the theory of computation, **Turing machines are used to study the properties of algorithms and to determine what problems can and cannot be solved by computers**. They provide a way to model the behavior of algorithms and to analyze their computational complexity, which is the amount of time and memory they require to solve a problem.
- Today, Turing Machines are considered to be one of the foundational models of computability and (theoretical) computer science.
- A Turing machine (automaton) is capable of enumerating some arbitrary subset of valid strings of an alphabet. A set of strings which can be enumerated in this manner is called a Recursively Enumerable language (generated by Type-0 Grammar). The Turing machine can equivalently be defined as a model that recognises valid input strings, rather than enumerating output strings.
- Given a Turing machine *M* and an arbitrary string *s*, it is generally not possible to decide whether *M* will eventually produce *s*. This is due to the fact that the halting problem is unsolvable, which has major implications for the theoretical limits of computing.
- The Turing machine is capable of processing an unrestricted grammar, which further implies that it is capable of robustly evaluating first-order logic in an infinite number of ways. This is famously demonstrated through lambda calculus.
- Turing machines are widely accepted as a synonym for algorithmic computability (Church-Turing thesis). Using these conceptual machines Turing showed that first-order logic validity problem is non-computable. That is there exists some problems for which you can never write a program no matter how hard you try!
- Alan Turing's groundbreaking work on the theoretical foundations of computation paved the way for the development of modern computing technology.