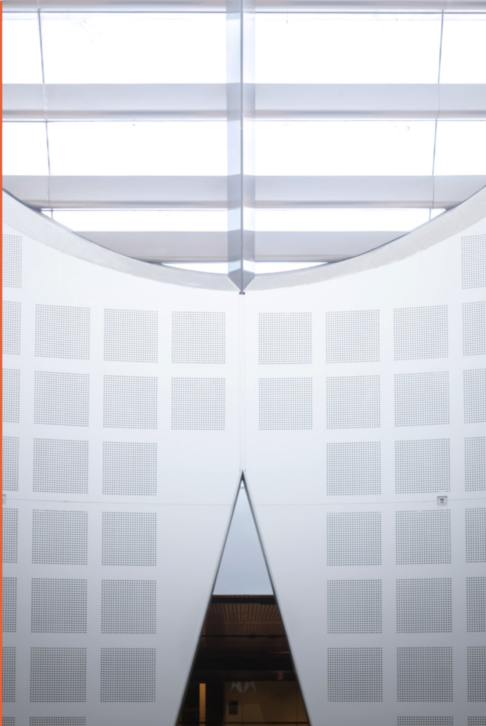
COMP2022
Models of Computation

Turing Machines

Sri AravindaKrishnan Thyagarajan (Aravind)

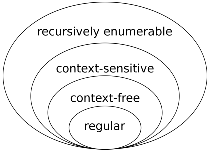
September 5, 2024

THE UNIVERSITY OF SYDNEY



Why study Turing Machines?

1. It is a simple and universal programming language
2. It is a formal model of computation that captures what we think an algorithm is, and can simulate any known computer or programming language.
3. It helps answer the questions such as:
   - "Can every computational problem be solved by an algorithm?"
   - "Is there such a thing as a program that can simulate every other program?"
4. It is part of computing culture.
   - appeared in Alan Turing's 1936 paper *On computable numbers, with an application to the Entscheidungsproblem*
   - earlier, an algorithm was described roughly as "a process with a finite number of operations"
5. It can describe the largest class in the Chomsky Hierarchy:



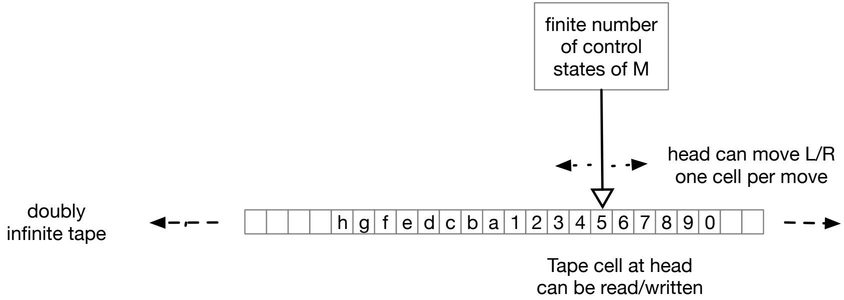
Turing (1936) describing his model

- Computing is normally done by writing certain symbols on paper. We may suppose this paper is divided into squares like a child's arithmetic book.
- I shall also suppose that the number of symbols which may be printed is finite. . . . The effect of this restriction of the number of symbols is not very serious. It is always possible to use sequences of symbols in the place of single symbols.
- The behaviour of the computer at any moment is determined by the symbols which he is observing, and his "state of mind" at that moment.
- We may suppose that in a simple operation not more than one symbol is altered. Any other changes can be split up into simple changes of this kind.
- The most general single operation is ... a possible change of symbol together with a possible change of state of mind.
- We may suppose that there is a bound B to the number of symbols or squares which the computer can observe at one moment. If he wishes to observe more, he must use successive observations.
- We will also suppose that the number of states of mind which need be taken into account is finite. ... Again, the restriction is not one which seriously affects computation, since the use of more compli- cated states of mind can be avoided by writing more symbols on the tape.

Turing Machines in a nutshell

A Turing Machine (TM) M is like a program that:

- Can only use variables with finite domains
  - e.g., variable called `state` taking finitely many values
- Has an infinite tape, made of cells
  - there is a single pointer (aka head) on the tape
  - the pointer can move left, right, or stay
  - the pointer can read and write symbols (from a bigger tape alphabet alphabet Γ)


finite number of control states of M
head can move L/R one cell per move
doubly infinite tape
h g f e d c b a 1 2 3 4 5 6 7 8 9 0
Tape cell at head can be read/written

- Initially the input string over input alphabet Σ is written on the tape
- The machine can, depending on the state, decide to Halt (aka stop running) and "Accept" or "Reject".
- It is deterministic.
- A TM M accepts a string $u \in \Sigma^*$ if M starting with u on the input tape, with the pointer on the left-most symbol, from the initial state, runs and reaches an "Accept" state.
- The language recognised by M (or language of M) is the set of strings it accepts.

TM is written as a list of "instructions"

- Each instruction is of the form

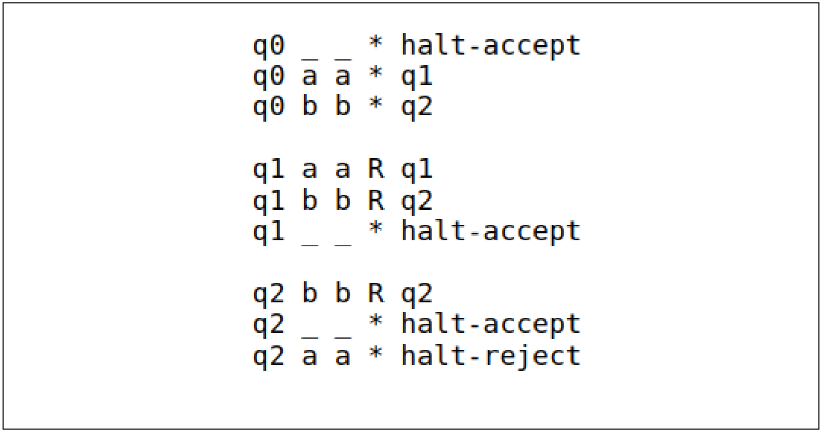
```
<current state> <current symbol> <new symbol> <direction> <new state>
```

  The instruction "q1 a b R q2" tells the machine that if it is in state $q1$ and sees the symbol a under the head, it should write the symbol b under the head, move one cell to the right, and change to state $q2$.
  - Symbols are from the tape alphabet Γ, which includes the input alphabet Σ and blank '_'.
  - Directions:
    - Move one cell to the left 'L'
    - Move one cell to the right 'R'
    - Do not move '*' (aka S)
- The machine stops running when it reaches any state starting with 'halt', eg. halt, halt-accept, halt-reject.

TMs can be drawn

TM recognising the language of a^*b^*

- Idea: scan the tape, do not write anything, but just keep track that once you see a b you don't again see an a.
  - NB. This TM does not write on its tape
- link to TM



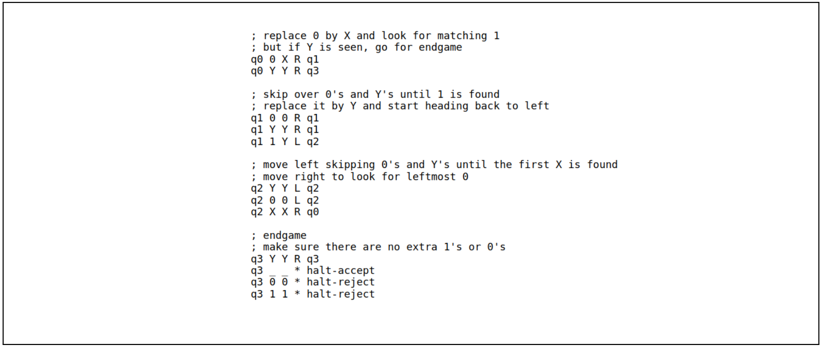
```
q0 _ _ * halt-accept
q0 a a * q1
q0 b b * q2

q1 a a R q1
q1 b b R q2
q1 _ _ * halt-accept

q2 b b R q2
q2 _ _ * halt-accept
q2 a a * halt-reject
```

TM recognising $\{0^n1^n : n \geq 1\}$

- Match leftmost 0 with leftmost 1, replace 0 by X and 1 by Y (so the TM doesn't lose its place), and repeat.
- If a 1 cannot be found in this way, reject (more 0s than 1s)
- If a 0 cannot be found in this way, go to the end and check there are no more characters
- link to TM



```
; replace 0 by X and look for matching 1
; but if Y is seen, go for endgame
q0 0 X R q1
q0 Y Y R q3

; skip over 0's and Y's until 1 is found
; replace it by Y and start heading back to left
q1 0 0 R q1
q1 Y Y R q1
q1 1 Y L q2

; move left skipping 0's and Y's until the first X is found
; move right to look for leftmost 0
q2 Y Y L q2
q2 0 0 L q2
q2 X X R q0

; endgame
; make sure there are no extra 1's or 0's
q3 Y Y R q3
q3 _ _ * halt-accept
q3 0 0 * halt-reject
q3 1 1 * halt-reject
```

Formal definition of TM

A formal definition of a TM, and what it means for a TM to accept a string is given in Sipser (pgs 167 - 170).

Some notation for TMs:

- Q = set of states.
- Σ = input alphabet.
- Γ = tape alphabet, includes Σ and the blank symbol _.
- $\delta : Q \times \Gamma \to \Gamma \times \{L, R, S\} \times Q$ is the transition function

Formal picture of Alan Turing

Relationship between TMs and DFAs?

1. You can think of a DFA as a TM that cannot write on its tape, can only move right, and must enter a halting state when it first reaches a blank symbol (i.e., it has read the entire input).
2. A TM can, instead, also write on the tape, also move left, also go past the original input, and also run forever.

Test your understanding

Let $\Sigma = \{a, b\}$. What is the language recognised by this TM?

```
q0   a   c   R   q0
q0   b   c   R   q0
```

1. All strings over alphabet $\{a, b\}$
2. All strings over alphabet $\{a, b, c\}$
3. The empty set.

Test your understanding

Let $\Sigma = \{a\}$. What does this TM do?

```
q0   a   a   R   q0
q0   _   a   R   q0
```

1. It doesn't change the contents of the tape, it just keeps moving right.
2. It keeps writing as to the end of the string on the tape, and never halts.
3. It keeps writing as to the end of the string, and then halts.

What is the right level of detail for describing TMs?

1. Formal description.
   - Lowest level of detail.
   - Specify the states and transitions in a table or diagram.
2. Implementation description.
   - English description of the way the TM moves its head and stores data on the tape.
   - Good examples of this in Sipser.
3. High-level description.
   - English description describing an algorithm.
   - No mention of the TM's tape or head.
   - This is how you do it in COMP2123

Q: Ok, so these TMs are fun. But why don't we just develop our understanding of what is computable and what is not computable using a real programming language C or Python? Why do this with Turing Machines?

A: The TM is, syntactically, much simpler than any general purpose programming language. E.g., we don't need to worry about run-time errors (dividing by zero, accessing an undefined variable, etc). This makes it easier to reason about (what we do in this course!).

Q: But it is quite cumbersome to program a TM. I guess that is the tradeoff — TMs are easier to analyse but harder to program than Python code.

A: Right. Here are some design tips...

Pro tips

1. Use blocks of code
2. States store information about what has been done or what needs to be done. Use informative names for states.
3. View the tape as having multiple tracks, and the tape alphabet as having composite symbols (for storing more information).

1. Use blocks of code

We already saw this in the examples.

2. States store information

A TM for $\{ww^R : w \in \{a,b\}^*\}$ (here w^R is the reverse of w) could match the leftmost symbol with the rightmost symbol, erasing both.

The state name can store the leftmost symbol while heading right to find the rightmost symbol:

q_a means the leftmost symbol was an a

q_b means the leftmost symbol was a b

3. View tape as multiple tracks

A TM for $\{ww : w \in \{a,b\}^+\}$ could first find and mark the midpoint of the input, and then use a matching process like before.

We view the tape as having two tracks: the first is for the input, the second is for marking the midpoint.

Formally, we change the tape alphabet. It now consists of symbols like

$$\begin{bmatrix} a \\ \ \end{bmatrix}, \begin{bmatrix} b \\ \ \end{bmatrix}, \begin{bmatrix} a \\ M \end{bmatrix}, \begin{bmatrix} b \\ M \end{bmatrix}, \begin{bmatrix} \ \\ \ \end{bmatrix}$$

and a tape might look like

$$\cdots \begin{bmatrix} \ \\ \ \end{bmatrix} \begin{bmatrix} \ \\ \ \end{bmatrix} \begin{bmatrix} a \\ \ \end{bmatrix} \begin{bmatrix} a \\ \ \end{bmatrix} \begin{bmatrix} b \\ \ \end{bmatrix} \begin{bmatrix} a \\ M \end{bmatrix} \begin{bmatrix} a \\ \ \end{bmatrix} \begin{bmatrix} b \\ \ \end{bmatrix} \begin{bmatrix} \ \\ \ \end{bmatrix} \begin{bmatrix} \ \\ \ \end{bmatrix} \cdots$$

A subtle point

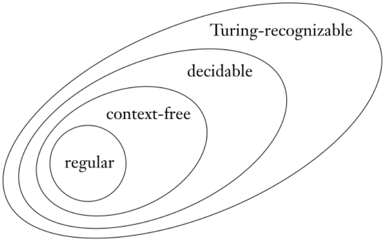
A TM can fail to accept some input for one of two reasons:

1. its computation is rejecting, *i.e.*, ends in a rejecting state.
2. its computation is diverging, *i.e.*, never reaches a rejecting or accepting state (and so is infinite).

Turing-recognisable languages

Definition

1. A language is Turing-recognisable (aka recognisable)[1] if some TM recognises it.
2. A TM that halts on all inputs is called a decider.
3. A language is Turing-decidable (aka decidable)[2] if some decider recognises it.



[1]aka *computably enumerable*, *recursively enumerable*
[2]aka *computable*, *recursive*

Good to know

Various formalisms of general computation have been proposed:

1. Turing machines (Alan Turing 1936)
2. Post systems (Emil Post)
3. μ-recursive functions (Kurt Gödel, Jacques Herbrand)
4. λ-calculus (Alonzo Church, Stephen C. Kleene)
5. Combinatory logic (Moses Schönfinkel and Haskell B. Curry)

Theorem. All these formalisms are equivalent, i.e., a problem is decidable in one formalism if and only if it is decidable by any other.

– This result strongly supports the idea that there is only one form of general computation.

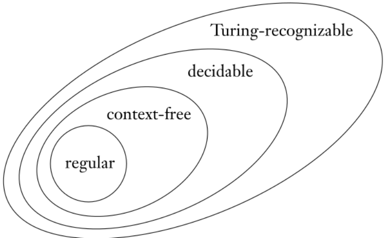
– The Church-Turing Thesis says that the intuitive notion of algorithms equals Turing machine algorithms.

Summary

- TMs are a machine model of computation, equivalent to many other models of computation.
- They are different from simpler models (e.g., DFA) because they have unrestricted access to unlimited memory.
- The Church-Turing Thesis says that the intuitive idea of an algorithm equals Turing machine algorithms.

Some questions we will answer:

- Is there a language that is not recognisable?
- Is there a recognisable language that is not decidable?



Here is some terminology, used in some tutorial questions.

Configurations

A configuration for a TM is a string $uqv \in \Gamma^* Q \Gamma^*$.

Think of it as a "snapshot in time" of an execution of the TM.

It represents the situation in which

1. q is the current state,
2. the tape content is uv (the infinite strings of blanks to the left and right of uv are not written),
3. the head is at the first symbol of v.

e.g., the configuration $XXXq_1Y11$ represents the situation where the machine is in state q_1, the tape stores $XXXY11$, and the head is over the fourth cell (that stores a Y).

Configurations

Special configurations:

- for input string w, configurations of the form q_0w are called start configurations.
  - e.g., $q_0 011$
- Configurations of the form $uq_{\text{reject}}v$ are called rejecting configurations.
  - e.g., $XYq_{\text{reject}}1$
- Configurations of the form $uq_{\text{accept}}v$ are called accepting configurations.
  - e.g., $XXYYq_{\text{accept}}$

TM computations

- For configurations C, D write $C \vdash_M D$, read C yields D, if the TM can go from C to D in one step.[3]

$$q_0 011 \vdash X q_1 11 \vdash q_2 XY1 \vdash X q_0 Y1 \vdash XY q_3 1 \vdash XY q_{\text{reject}} 1$$

- Write $C \vdash_M^* D$ if the TM M can go from C to D in any number of steps.

$$q_0 011 \vdash_M^* XY q_{\text{reject}} 1$$

- The language $L(M)$ recognised by the TM M consists of all strings of the form w for which the computation starting with the start configuration $q_0 w$ ends in an accepting configuration:

$$L(M) = \{w \in \Sigma^* : q_0 w \vdash_M^* u q_{\text{accept}} v, \text{ for some strings } u, v\}$$

[3] For a formal definition of $\vdash_M$ see Sipser pg 169.