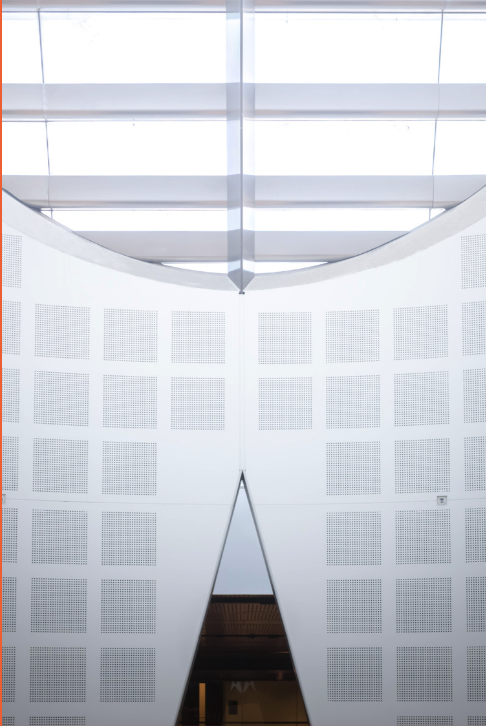
COMP2022|2922
Models of Computation

Turing Machines

Sasha Rubin

August 25, 2022

THE UNIVERSITY OF SYDNEY



How can we specify languages?

1. In English.
2. In mathematics/set-theoretic notation, and recursive definitions.
3. By regular expressions R
4. By (context-free) grammars G
5. By automata M, and more generally by machine models of computation.

Turing Machines in a nutshell

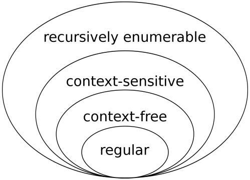
A Turing Machine M is like a symbol-processing program that:

- Can only use variables with finite domains: `state` taking finitely many values.
- Has an infinite tape on which the input string is written.
  - single pointer onto the tape
  - move the pointer left and right
  - read and write symbols onto the tape under the pointer.
- Can decide to Halt and "Accept" or "Reject" (depending on the current state).

The language recognised by M is the set of strings u such that the machine M running with u starting on the tape reaches an "Accept" state.

Why study Turing Machines?

1. It is a formal model of computation that can solve every problem a real computer can solve.
2. It helps answer the question "What problems can be solved by computation"?
3. 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 as "process with a finite number of operations".
   - Turing showed that some decision problems cannot be solved by his model of computation (we will see this later!)
4. It can describe the largest class in the Chomsky Hierarchy:



Good to know

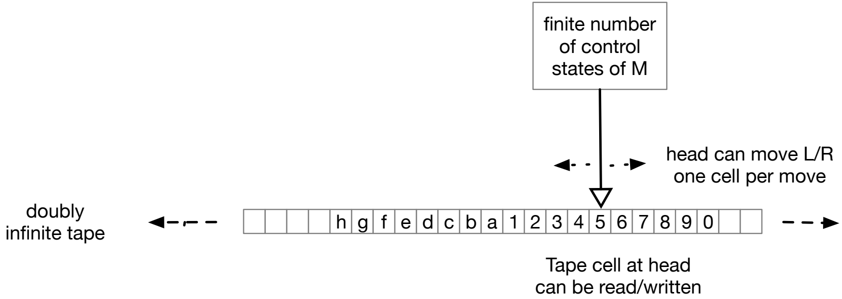
Various formalisms have been proposed:

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

Theorem. All these formalisms are equivalent, i.e., a problem is decidable by one approach 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.

Turing Machines



Syntax

- We write a TM M as a list of "instructions".
- Each instruction is of the form

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

  - You can use any string for states
  - The possible directions are move the head one cell left (L), right (R), or do not move it (S or *).
  - Symbols are from the tape alphabet or blank ($\sqcup$ or B)
- The machine halts when it reaches any state starting with 'halt', eg. halt, halt-accept.

Turing Machines: Example 1

- TM recognising the language of the regular expression a^*b^* over alphabet $\{a, b\}$.
- 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.

q_0	⊔	⊔	S	halt_accept
q_0	a	a	R	q_1
q_0	b	b	R	q_2
q_1	a	a	R	q_1
q_1	b	b	R	q_2
q_1	⊔	⊔	S	halt_accept
q_2	b	b	R	q_2
q_2	⊔	⊔	S	halt_accept
q_2	a	a	S	halt_reject

link to TM

Turing Machines: Example 2

- TM recognising the language $L = \{0^n 1^n : n \geq 1\}$
- Idea:
  1. match leftmost 0 with leftmost 1,
  2. replacing 0 by X and 1 by Y (so the TM doesn't lose its place),
  3. and repeat.
  4. Finally, once a 0 cannot be found this way, go to the end of the Y's and check that there are no more bits (to reject strings which have a proper prefix in L, like 00110 or 00111).
- link to TM

Drawing TMs

Turing Machines

Definition

A TM is a tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ with components:

1. Q is a finite set of states
2. Σ is the input alphabet, not containing the *blank symbol* $\sqcup$
3. Γ is the tape alphabet, containing $\sqcup$ and all of Σ
4. $\delta : Q \times \Gamma \to \Gamma \times \{L, R, S\} \times Q$ is the transition function

   L, R, S are sometimes called directions (S stands for "stay")
5. q_0 is the start state
6. q_{accept} is the accept state
7. q_{reject} is the reject state ($\neq q_{\text{accept}}$)

q_{accept} and q_{reject} are also the only halting states, so we sometimes call them halt-accept and halt-reject.

Language of a TM

The collection of strings that M accepts is called the language recognised by M (or the lanuage of M), and is denoted $L(M)$.

What is the formal definition of M accepting a string? See Sipser pgs 167 - 170.

Test your understanding

What is $L(M)$ for this machine TM M:

```
q_0  a  c  R  q_0
q_0  b  c  R  q_0
```

Vote now! (on mentimeter)

1. All strings over alphabet $\{a, b\}$
2. All strings over alphabet $\{a, b, c\}$
3. No strings

Turing-recognisable languages

Definition

A language is Turing-recognisable[1] if some TM M recognises it.

A TM can fail to accept some input either because its computation is:

- rejecting, *i.e.*, ends in a rejecting configuration.
- diverging, *i.e.*, never reaches a rejecting or accepting configuration (and so is infinite).

Definition

A TM M that halts on all inputs is called a decider. A language is Turing-decidable[2] if some decider M recognises it.

[1] aka *recognisable*, *computably enumerable*, *recursively enumerable*

[2] aka *decidable*, *computable*, *recursive*

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, ignoring implementation details.
   - No mention of the TM's tape or head
   - This is how you do it in COMP2123

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.

Questions we will answer:

- Can every language be recognised by a TM?
- E.g., is there a TM that can take a TM M as input(!) and decide if M has a diverging computation?

Here is some useful terminology, used in the tutorials.

Configurations

A configuration for a TM is a triple $(u, q, v) \in \Gamma^* \times Q \times \Gamma^*$ typically written as the string uqv.

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 $q_0 w$ are called start configurations.
  - e.g., $q_0 011$
- Configurations of the form $u q_{\text{reject}} v$ are called rejecting configurations.
  - e.g., $XY q_{\text{reject}} 1$
- Configurations of the form $u q_{\text{accept}} v$ are called accepting configurations.
  - e.g., $XXYY q_{\text{accept}}$

TM computations

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

$$q_0 011 \vdash Xq_1 11 \vdash q_2 XY1 \vdash Xq_0 Y1 \vdash XYq_3 1 \vdash XYq_{\text{reject}} 1$$

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

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

- 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:

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

[3] For a formal definition of $\vdash_M$ see Sipser pg 169 where it is called the 'yields' operation between configurations.