Turing Machines

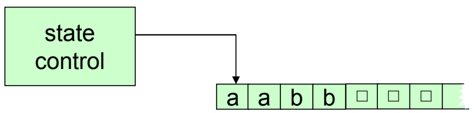
Andreas Karwath & Malte Helmert

Overview

- Turing machines
- Variants of Turing machines
  - Multi-tape
  - Non-deterministic
  - …
- The definition of algorithm
  - The Church-Turing Thesis

Turing Machine

- Infinite tape
  - Both read and write from tape
  - Move left and right
  - Special accept and reject state take immediate effect
  - Machine can accept, reject or loop



Schematic of a Turing Machine

$F = \{w\#w \mid w \in \{0,1\}^*\}$

M_1 = "On input string w:

1. Scan the input to be sure that it contains a single # symbol. If not, *reject*.
2. Zig-zag across the tape to corresponding positions on either side of the # symbol to check on whether these positions contain the same symbol. If they do not, *reject*. Cross off symbols as they are checked to keep track of which symbols correspond.
3. When all symbols to the left of the # have been crossed off, check for any remaining symbols to the right of the #. If any symbols remain, *reject*; otherwise *accept*."

$F = \{w\#w \mid w \in \{0,1\}^*\}$

0 1 1 0 0 0 # 0 1 1 0 0 0 □

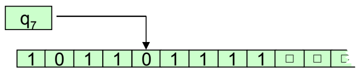
Snapshots of the Turing machine computing on input 011000#011000

Turing Machines

A *Turing machine* is a 7-tuple, $(Q,\Sigma,\Gamma,\delta,q_0,q_{accept},q_{reject})$, where Q,Σ,Γ are all finite sets and

1. Q is the set of states,
2. Σ is the input alphabet not containing the special *blank* symbol □,
3. Γ is the tape alphabet, where □ $\in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. δ: $Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L,R\}$ is the transition function,
5. $q_0 \in Q$ is the start state,
6. $q_{accept} \in Q$ is the accept state, and
7. $q_{reject} \in Q$ is the reject state, where $q_{reject} \neq q_{accept}$.

Configurations



A Turing machine with the configuration $1011q_701111$

$ua\,q_i\,bv$ yields $u\,q_j\,acv$ if $\delta(q_i,b)=(q_j,c,L)$

$ua\,q_i\,bv$ yields $uac\,q_j\,v$ if $\delta(q_i,b)=(q_j,c,R)$

cannot go beyond left border !

start configuration q_0w

accepting configuration - state is q_{accept}

rejecting configuration - state is q_{reject}

A Turing Machine accepts input w if a sequence of configurations $C_1,\ldots,C_k$ exists where

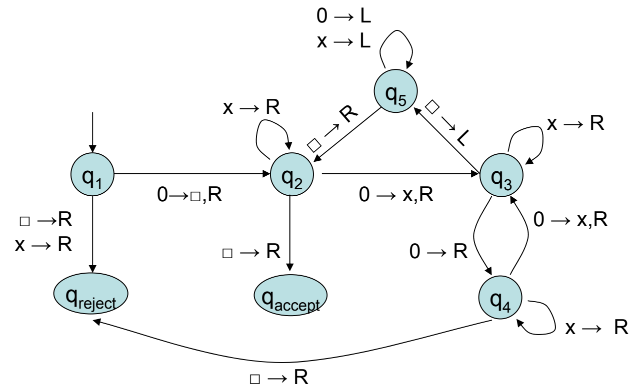
1. C_1 is start configuration
2. Each C_i yields C_{i+1}
3. C_k is an accepting state

Languages

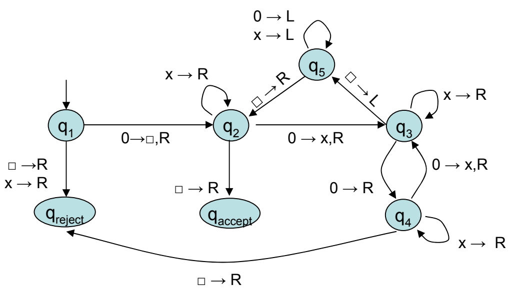
- The collection of strings that M accepts is the language of *M*, *L(M)* (or *L(M)* is language recognized by *M*)
- A language is *Turing-recognizable (recursively enumerable)* if some Turing machine accepts it
- *Deciders* halt on every input (i.e. they do not loop)
- A language is *Turing-decidable (recursive)* if some Turing machine decides it

Example 3.4

This is the description of a TM M_2 that recognizes the language consisting of all strings of $0s$ whose length is a power of 2. It decides the language $A = \{ 0^{2^n} \mid n \geq 0 \}$.



State diagram for Turing machine M_2



Example 3.5

M_1= "On input string w:

1. Scan the input to be sure that it contains a single # symbol. If not, *reject*.
2. Zig-zag across the tape to corresponding positions on either side of the # symbol to check on whether these positions contain the same symbol. If they do not, *reject*. Cross off symbols as they are checked to keep track of which symbols correspond.
3. When all symbols to the left of the # have been crossed off, check for any remaining symbols to the right of the #. If any symbols remain, *reject*; otherwise *accept*."

$$F = \{ w\#w \mid w \in \{0,1\}^* \}$$

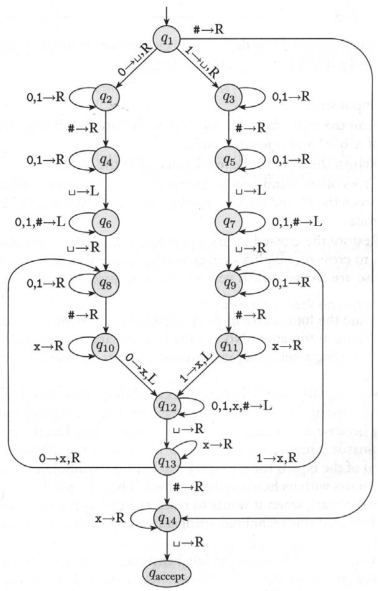


FIGURE 3.5
State diagram for Turing machine M_1

Example 3.4

The TM M_3 is doing some elementary arithmetic. It decides the language $C = \{a^i b^j c^k \mid i \times j = k \text{ and } i, j, k \geq 1\}$.

Example 3.7

The Turing machine M_4 is solving what is called the *element distinctness problem*. It is given a list of strings over $\{0,1\}$ separated by #s and its job is to accept if all the strings are different. The language is

$$E = \{\#x_1\#x_2\#\ldots\#x_l \mid \text{each } x_i \in \{0,1\}^* \text{ and } x_i \neq x_j \text{ for each } i \neq j\}$$

Machine M_4 works by comparing x_1 and x_2 through x_l, then by comparing x_2 and x_3 through x_l, and so on. An informal description of the TM M_4 deciding this language follows:

Example 3.7 (cont.)

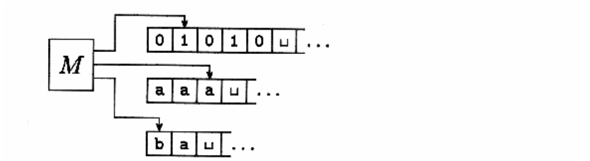
M_4 = "On input *w*:

1. Place a mark on top of the leftmost tape symbol. If that symbol was a blank, *accept*. If that symbol was a #, continue with the next stage. Otherwise, *reject*.
2. Scan right to the next # and place a second mark on top of it. If no # is encountered before a blank symbol, only x_1 was present, so *accept*.
3. By zig-zagging, compare the two strings to the right of the marked #s. If they are equal, *reject*.
4. Move the rightmost of the two marks to the next # symbol to the right. If no # symbol is encountered before a blank symbol, move the leftmost mark to the next # to its right and the rightmost mark to the # after that. This time, if no # is available for the rightmost mark, all the strings have been compared, so *accept*.
5. Go to Stage 3."

Variants of Turing Machines

- Most of them turn out to be equivalent to original model
- E.g. consider movements of head on tape {L,R,S} where S denotes "same" (for "same position" or "stay put")
- Equivalent to original model (represent S transition by first R and then L, or vice versa)

Multi-tape Turing Machines

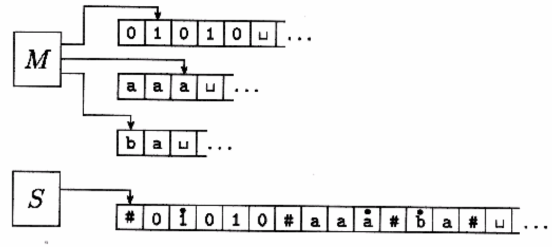


The input appears on Tape 1; others start off blank

Transition function becomes

$$\delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{L, R\}^k$$

$$\delta(q_i, a_1, \ldots, a_k) = (q_j, b_1, \ldots, b_k, L, R, \ldots, R)$$



Representing three tapes with a single one

Theorem

Every multitape Turing machine has an equivalent single tape Turing machine.

S="On input $w = w_1 \ldots w_n$:

1. First S puts its tape into the format that represents all k tapes of M. The formatted tape contains

$$\# \dot{w}_1 w_2 \ldots w_n \# \dot{\sqcup} \# \dot{\sqcup} \# \ldots \#$$

Corollary

A language is Turing recognizable if and only if some multitape TM recognizes it.

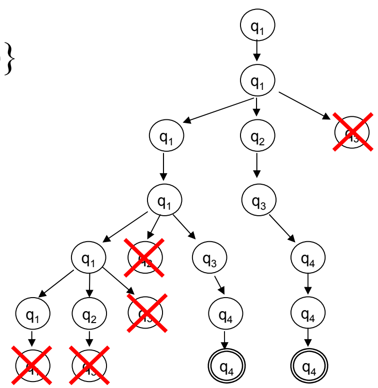
Non-deterministic TMs

Transition function becomes

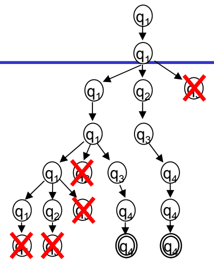
$$\delta : Q \times \Gamma \to \mathrm{P}(Q \times \Gamma \times \{L, R\})$$

$$\delta(q, a) = \{(q_1, b_1, L), \ldots, (q_k, b_k, R)\}$$

Same idea/method as for NFAs



Non-deterministic TMs



Theorem

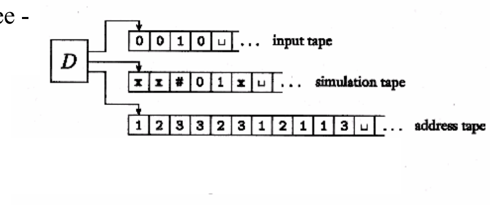
Every non-deterministc Turing machine has an equivalent deterministic Turing machine.

Proof idea

Numbering the computation.

Work with three tapes :

1. input tape (unchanged)
2. simulator tape
3. index for computation path in the tree - alphabet $\Sigma_b = \{1,\ldots,b\}$



1. Initially tape 1 contains the input *w*, and tapes 2 and 3 are empty.
2. Copy tape 1 to tape 2.
3. Use tape 2 to simulate *N* with input *w* on one branch of its non-deterministic computation. Before each step of *N* consult the next symbol on tape 3 to determine which choice to make among those allowed by *N*´s transition function. If no more symbols remain on tape 3 or if this nondeterministic chice is invalid, abort this branch by going to stage 4. Also go to stage 4 if a rejecting configuration is encountered. If an accepting configuration is encountered, *accept* the input.
4. Replace the string on tape 3 with the lexicographically next string. Simulate the next branch of *N*´s computation by going to stage 2.

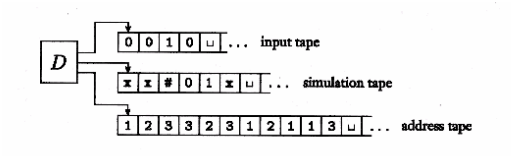
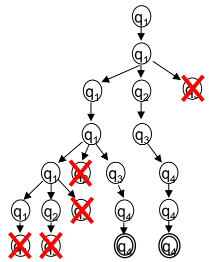


FIGURE 3.7
Deterministic TM *D* simulating nondeterministic TM *N*



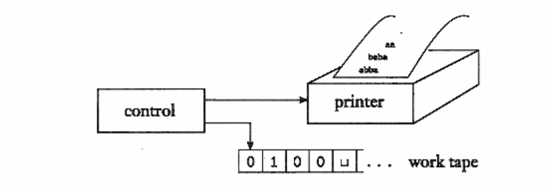
Theorem

A language is Turing-recognizable if and only if some non-deterministic TM recognizes it.

Corollary

A language is decidable if and only if some non-deterministic TM decides it.

Enumerators



Turing recognizable = Recursively enumerable

Therefore, alternative model of TM, enumerator

Works with input tape (initially empty) and output tape (printer).

The language enumerated by an Enumerator E, is the collection of all strings that it eventually prints out (in any order, with possible repetitions).

Theorem 3.13

A language is Turing-recognizable if and only if some enumerator enumerates it.

PROOF

First we show that if we have an enumerator E that enumerates a languages A, a TM M recognizes A.

Theorem 3.13 (cont.)

A language is Turing-recognizable if and only if some enumerator enumerates it.

PROOF (other direction)

If TM M recognizes a language A, we can construct the following enumerator E for A.

Say that $s_1, s_2, s_3, \ldots$ is a list of all possible strings in Σ^*.

E="Ignore the input.

1. Repeat tho following for $i = 1,2,3,\ldots$
2. Run M for i steps on each input, $s_1, s_2, \ldots, s_i$.
3. If any computations accept, print out the corresponding s_j."

If M accepts a particular string s, eventually it will appear on the list genereated by E. In fact, it will appear on the list infinitely many times because M runs from the beginning on each string for each repetition of step 1. This procedure gives the effect of running M in parallel on all possible input strings.

Equivalence with other models

- Many variants of TMs (and related constructs) exist.
- All of them turn out to be equivalent in power (under reasonable assumptions, such as finite amount of work in single step)
- Programming languages : Lisp, Haskell, Pascal, Java, C, …
- The class of algorithms described is natural and identical for all these constructs.
- For a given task, one type of construct may be more elegant.

The definition of an algorithm

- David Hilbert
  - Paris, 1900, Intern. Congress of Maths.
  - 23 mathematical problems formulated
- 10^{th} problem
  - “to devise an algorithm that tests whether a polynomial has an integral root”
  - Algorithm = “a process according to which it can be determined by a finite number of operations”

Integral roots of polynomials

$6x^3yz + 3xy^2 - x^3 - 10$

root = assignment of values to variables so that value of polynomial equals 0

integral root = all values in assignment are integers

Church – Turing Thesis

There is no
A formal not
Alonso Chu
Allen Turing

Intuitive notion of algorithm

=

Turing machine algorithms

Integral roots of polynomials

$D = \{p \mid p \text{ is a polynomial with an integral root}\}$

Turing machines

- Three levels of description
  - Formal description
  - Implementation level
  - High-level description
    - The algorithm is described
    - From now on, we use this level of description

STRINGS!!

$\langle O \rangle$: describes object O

$\langle O_1, \ldots, O_k \rangle$: describes objects $O_1, \ldots, O_k$

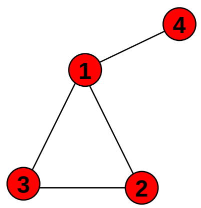
Encodings can be done in multiple manners;
often not relevant because one encoding (and therefore TM
can be transformed into another one)

Connected graphs

$A = \{\langle G \rangle \mid G \text{ is a connected undirected graph}\}$

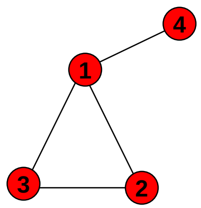
connected = every node can be reached from every other node

G =



A (connected) graph G

G =



<G> = (1,2,3,4) ((1,2),(2,3),(3,1),(1,4))

A (connected) graph G and its encoding

M ="On input $\langle G \rangle$, the encoding of a graph G:

Summary

- Turing machines
- Variants of Turing machines
  - Multi-tape
  - Non-deterministic
  - …
- The definition of algorithm
  - The Church-Turing Thesis