Turing Machines

Bernhard Nebel and Christian Becker-Asano

Overview

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

Turing Machine (TM)

- 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

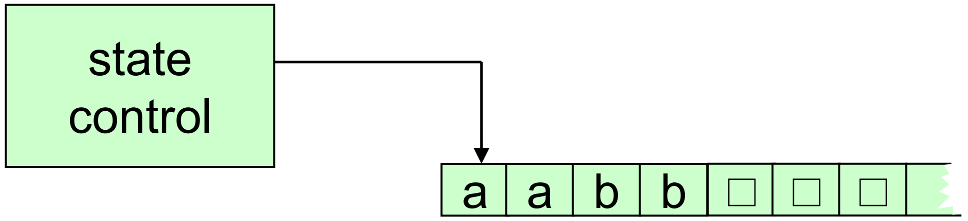


Figure 3.1: Schematic of Turing machine

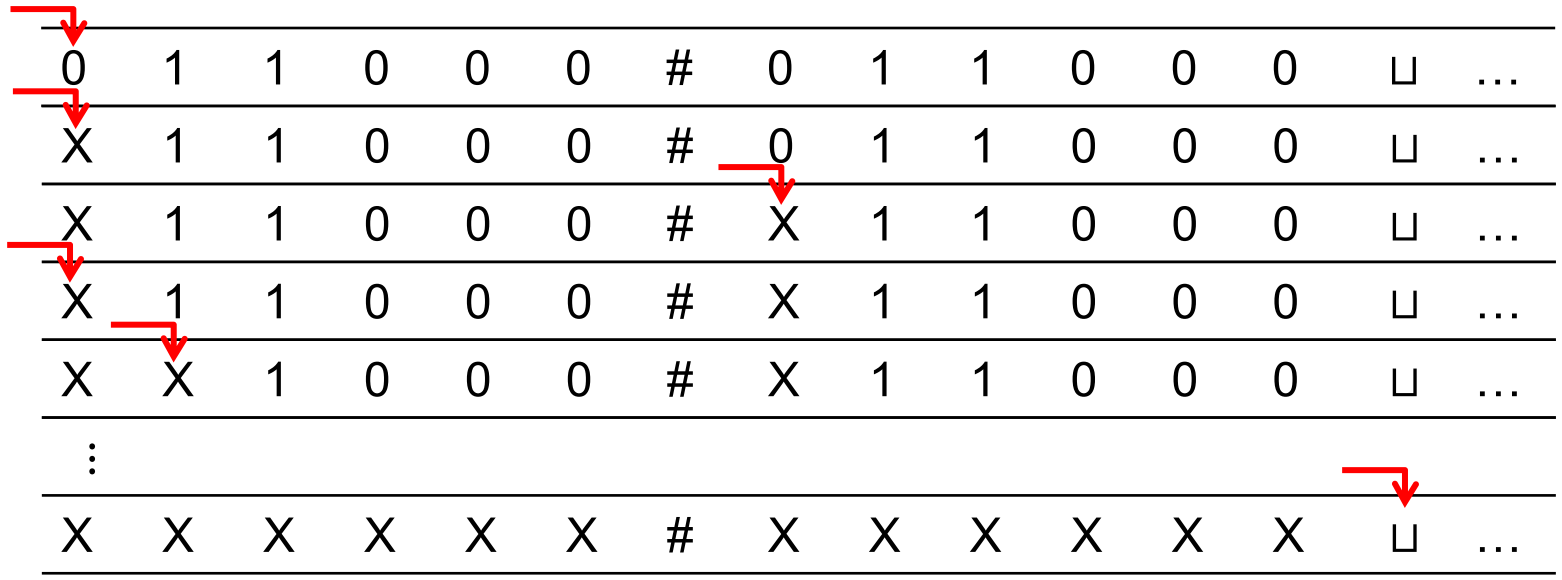
$F = \{w\#w \mid w \in \{0,1\}^*\}$

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*."

Example run of TM accepting $w_1 \in F$

$$F = \{w\#w \mid w \in \{0,1\}^*\}$$
$$w_1 \in F = \text{"011000\#011000"}$$


0 1 1 0 0 0 # 0 1 1 0 0 0 ⊔ …
X 1 1 0 0 0 # 0 1 1 0 0 0 ⊔ …
X 1 1 0 0 0 # X 1 1 0 0 0 ⊔ …
X 1 1 0 0 0 # X 1 1 0 0 0 ⊔ …
X X 1 0 0 0 # X 1 1 0 0 0 ⊔ …
⋮
X X X X X X # X X X X X X ⊔ …

Snapshots of the Turing machine

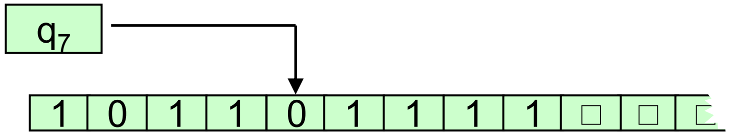
Formal definition of a Turing Machine

DEFINITION 3.3:

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 blank symbol $\sqcup$,
3. Γ is the **tape alphabet**, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ 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 of TMs



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

TMs and 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 recognizes 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 3.7: informal description

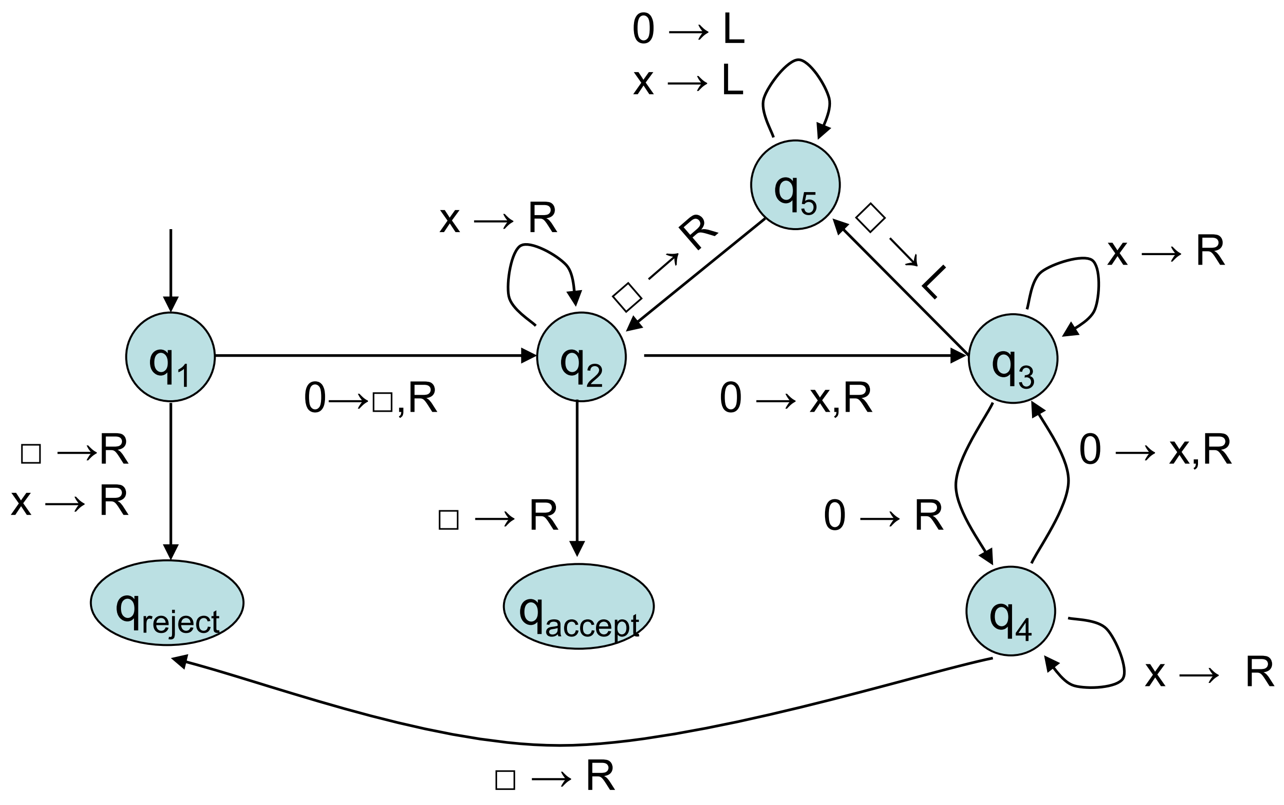
TM M_2 recognizes the language consisting of all strings of zeros with their length being a power of 2. In other words, it decides the language

$$A = \{0^{2^n} \mid n \geq 0\}.$$

M_2 = “On input string w:

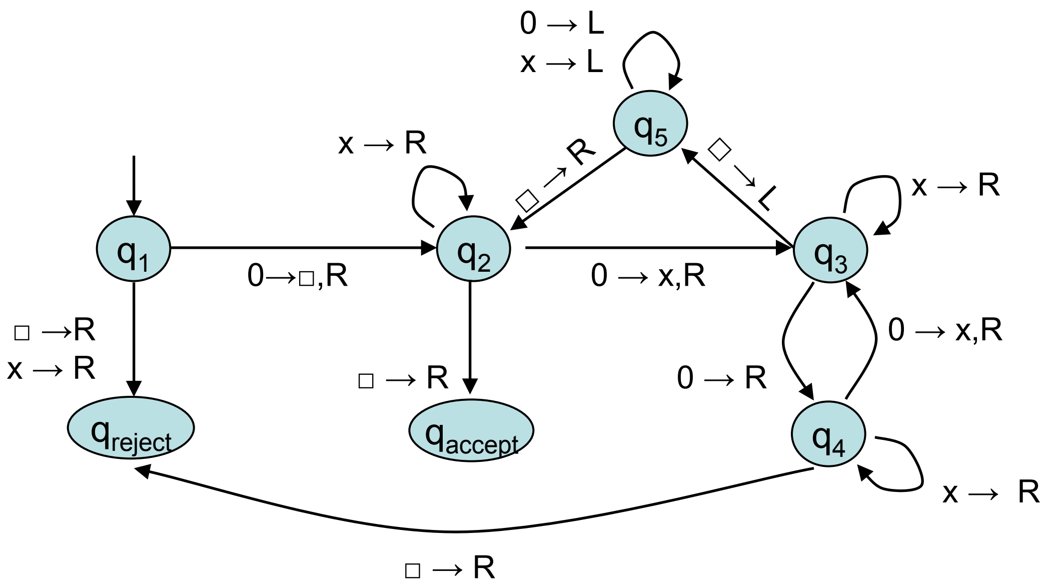
1. Sweep left to right accross the tape, crossing off every other 0.
2. If in stage 1 the tape contained a single 0, *accept*.
3. If in stage 1 the tape contained more than one 0 and the number of 0s was odd, *reject*.
4. Return the head to the left-hand end of the tape.
5. Go to stage 1.“

Example 3.7: state diagram for M_2



Example 3.7: example run of M_2

[illegible] [illegible] [illegible] [illegible] [illegible] □ q_{accept}



Example 3.9 $F = \{w\#w \mid w \in \{0,1\}^*\}$

M_1 = „On input string w:

1. Check for #, if not present *reject*.
2. Zig-zag across and cross off same symbols. If not same, *reject*.
3. When all symbols left of # are crossed off, check for additional symbols right of #. If yes, *reject*, otherwise *accept*.“

(cf. slides number 4 and 5)

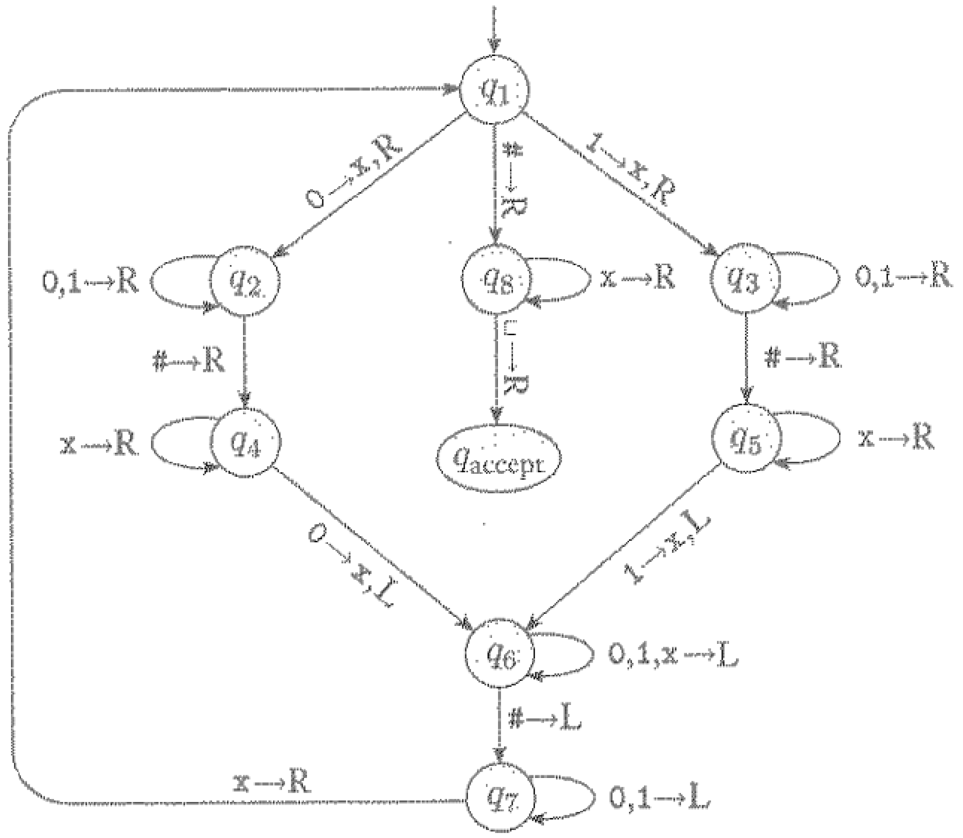


Figure 3.10: State diagram for TM M_1;
Note: for simplification transitions to *reject*-state are implicit and not shown here.

Example 3.12

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 3.12 (ctd.)

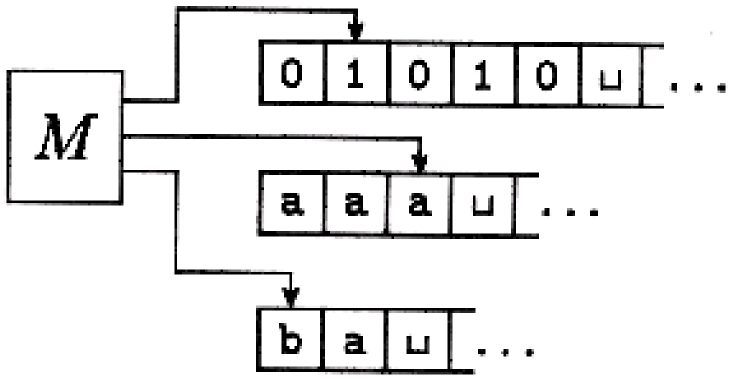
$M_4 =$ "On input w:

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

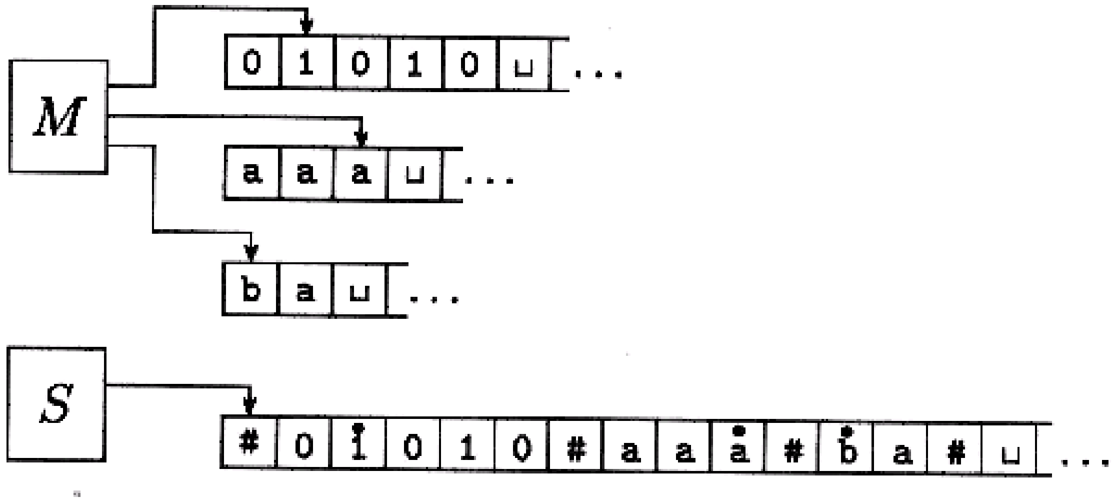
- Most of them turn out to be equivalent to original model
- E.g. consider movements of head on tape {L,R,S} where S denotes “same” (for “same position” or “stay put”)
- Equivalent to original model (represent S transition by first R and then L, or vice versa)

Multi-tape Turing Machines



- The input appears on Tape 1; the others k tapes start off blank
- Transition function is changed to:
  - $\delta: Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L, R, S\}^k$
  - E.g.: $\delta(q_1, a_1, \dots, a_k) = (q_j, b_1, \dots, b_k, L, R, \dots, L)$

Equivalence of multi- and single-tape TM



Representing three tapes with a single one

Theorem 3.13:
Every multitape Turing machine has an equivalent single-tape Turing machine.

Proof of theorem 3.13 (outline)

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

   $$\# \dot{w}_1 w_2 \ldots w_n \# \dot{\square} \quad \dot{\square}$$

2. To simulate a single move, S scans its tape from the first #, which marks the left-hand end, to the (k+1)st #, which marks the right-hand end, in order to determine the symbols under the virtual heads. Then S makes a second pass to update the tapes according to the way that M´s transition function dictates.
3. If at any point S moves one of the virtual heads to the right onto a #, this action signifies that M has moved the corresponding head onto the previously unread blank portion of that tape. So S writes a blank symbol on this tape cell and shifts the tape contents, from this cell until the rightmost #, one unit to the right. Then it continues the simulation as before."

Corollary

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

Intermezzo: TMs and programming langs

Programming language „Brainfuck“ (esoteric, 1993, Urban Müller):

- 8 language commands, each consisting of a single character

Character	Meaning
>	increment the data pointer (to point to the next cell to the right). R
<	decrement the data pointer (to point to the next cell to the left). L

```
++++++++++[>+++++++>++++++++++>+++>+<<<<-
>++.>+.+++++++..+++.>++.<<+++++++++++++++.>.+++.------.--------.>+.>.
```

.	output a character, the ASCII value of which being the byte at the data pointer.
,	accept one byte of input, storing its value in the byte at the data pointer.
[	if the byte at the data pointer is zero, then instead of moving the instruction pointer forward to the next command, jump it forward to the command after the matching] command.
]	if the byte at the data pointer is nonzero, then instead of moving the instruction pointer forward to the next command, jump it back to the command after the matching [command*.

(http://en.wikipedia.org/wiki/Brainfuck)

Intermezzo: PL BF

Note: Every non-BF-character is ignored and used for comments!

```
+++++ +++++ initialize counter (cell #0) to 10
[ use loop to set the next four cells to 70/100/30/10
       > +++++ ++ add 7 to cell #1
       > +++++ +++++ add 10 to cell #2
       > +++ add 3 to cell #3
       > + add 1 to cell #4
       <<<< - decrement counter (cell #0)
]
> ++ . print 'H'
> + . print 'e'
+++++ ++ . print 'l'
. print 'l'
+++ . print 'o'
> ++ . print ' '
```

```
<< +++++ +++++ +++++ . print 'W'
> . print 'o'
+++ . print 'r'
----- - . print 'l'
----- --- . print 'd'
> + . print '!'
> . print '\n'
```

(„Hello World!“; Try out „Visual brainfuck“ http://sites.google.com/site/visualbf/)

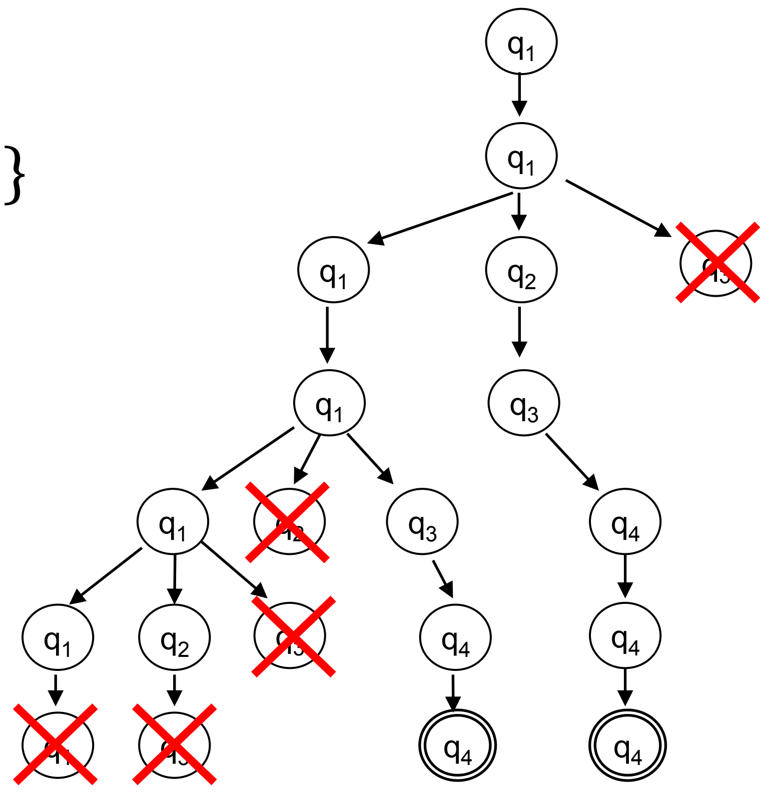
Nondeterministic TMs

Transition function is changed to:

$$\delta : Q \times \Gamma \rightarrow \wp(Q \times \Gamma \times \{L, R\})$$

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

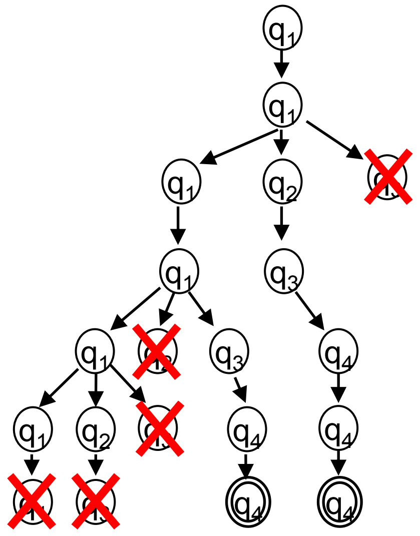
Same idea/method as for NFAs



(Non)deterministic TMs

Theorem 3.16:

Every nondeterministic Turing machine has an equivalent deterministic Turing machine.

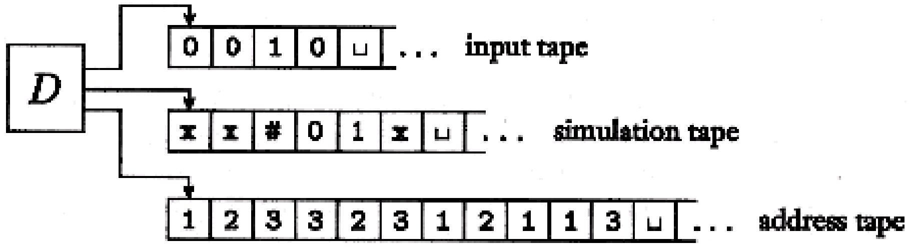


Proof idea

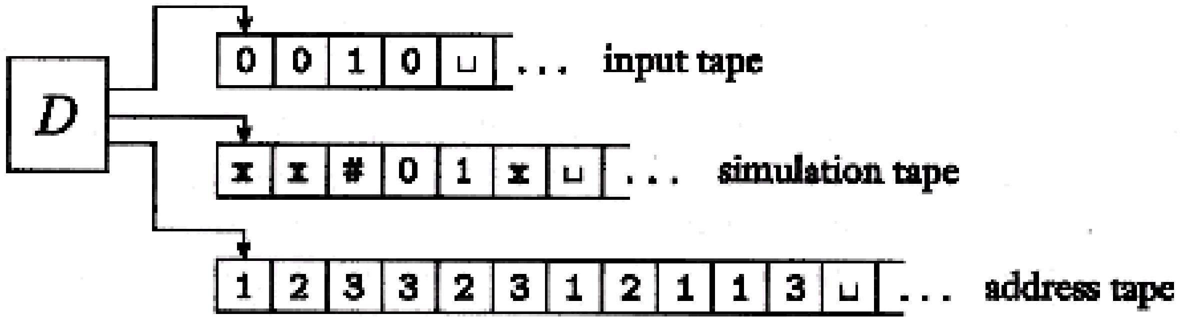
Numbering the computation.

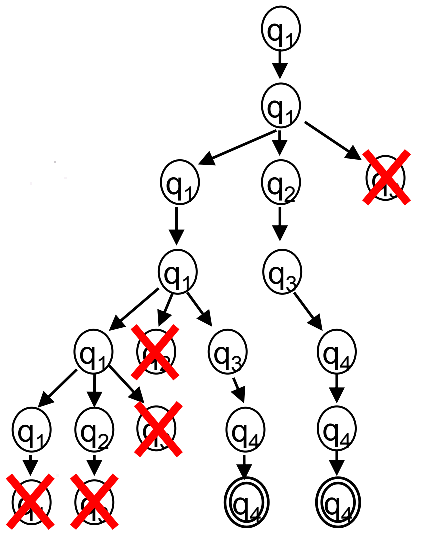
Work with three tapes :

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


D
0 0 1 0 ⊔ ... input tape
x x # 0 1 x ⊔ ... simulation tape
1 2 3 3 2 3 1 2 1 1 3 ⊔ ... address tape

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.





Nondeterministic TMs and languages

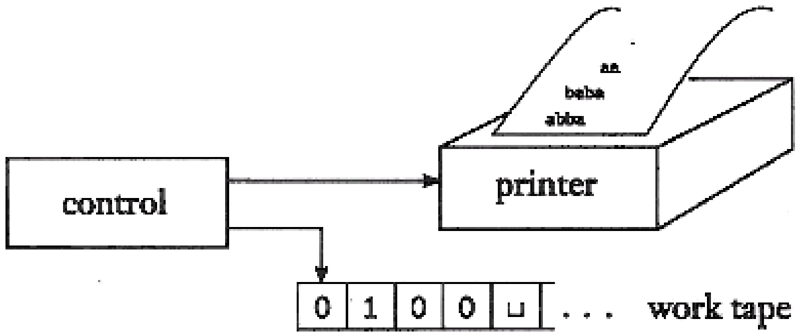
Corollary 3.18:

A language is Turing-recognizable if and only if some nondeterministic Turing machine recognizes it.

Corollary 3.19:

A language is decidable if and only if some nondeterministic Turing machine 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 3.21

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.

The TM M works in the following way.

```
$M =$ "On input $w$:
        1. Run $E$. Every time that $E$ outputs a string, compare it with $w$.
        2. If $w$ ever appears in the output of $E$, $accept$."
```

Clearly, M accepts those strings that appear on E's list.

Theorem 3.21 (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
- 10th 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

- There is no algorithm that solves this task.
- A formal notion of *algorithm* is necessary.
- Alonso Church : λ-calculus (cf. functional programming)
- Alan Turing : Turing machines

Church—Turing Thesis:

Intuitive notion of algorithm = Turing machine algorithms

Integral roots of polynomials

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

Hilbert's 10th problem : is D decidable ?

D is not decidable, but Turing recognizable

Consider $D_1 = \{p \mid p$ is a polynomial over x with an integral root$\}$

Define M_1 :

"the input is a polynomial over x

1. Evaluate p wrt x set to 0,1,-1,2,-2,3,-3,...

   If at any point p evaluates to 0, accept"

This is a recognizer for D_1 but not a decider

M_1 can be converted into a decider using the bounds $\pm k \dfrac{c_{\max}}{c_1}$ for x

k : number of terms; c_1 : coefficient highest order term; $c_{\max}$: largest absol. value coeff.

Extension of M_1 exist to D but remains a recognizer

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!!

$<O>$: describes an object

$<O_1, \dots, O_k>$: describes objects $O_1, \dots, O_k$

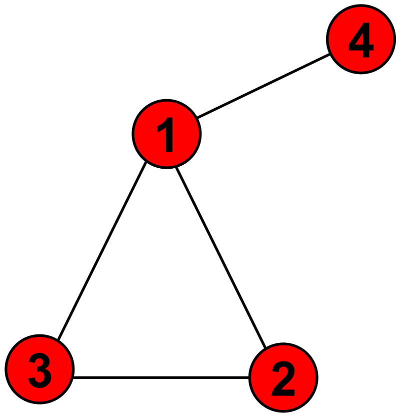
Encodings can be done in multiple manners, but this is often irrelevant because one encoding (and therefore TM) can be transformed into another one.

Connected graphs

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

connected = every node can be reached from every other node

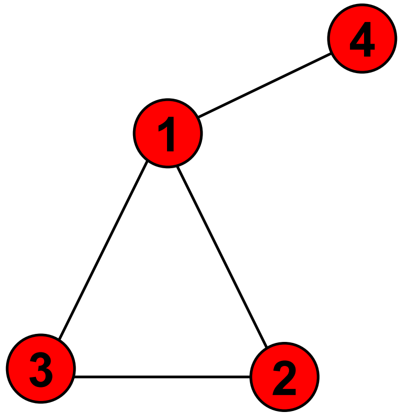
G =



A (connected) graph G

Connected graphs & TMs

G =



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

A (connected) graph G and its encoding

M = „On input <G>, the encoding of a graph G:

1. Select the first node of G and mark it.
2. Repeat the following stage until no new nodes are marked.
   - For each node in G, mark it if it is attached by an edge to a node that is already marked.
3. Scan all the nodes of G to determine whether thay all are marked.
   If yes, *accept*; otherwise *reject*.“

Summary

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