

Computational Models - Lecture 6¹

Handout Mode

Iftach Haitner and Yishay Mansour.

Tel Aviv University.

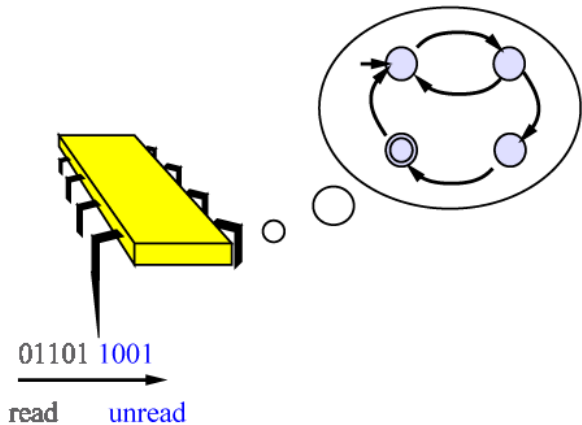
April 30/May 2, 2012

¹Based on slides by Benny Chor, Tel Aviv University, modifying slides by Maurice Herlihy, Brown University.

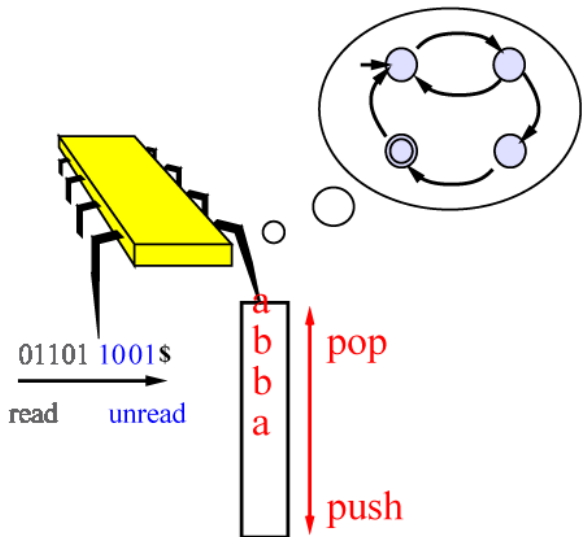
Talk Outline

- Turing Machines
- Alternative Models of Computers
- Multitape TMs, RAMs, Non Deterministic TMs
- The language classes $\mathcal{R} = \mathcal{RE} \cap \text{co}\mathcal{RE}$
- Sipser's book, 3.1, 3.2, & 3.3

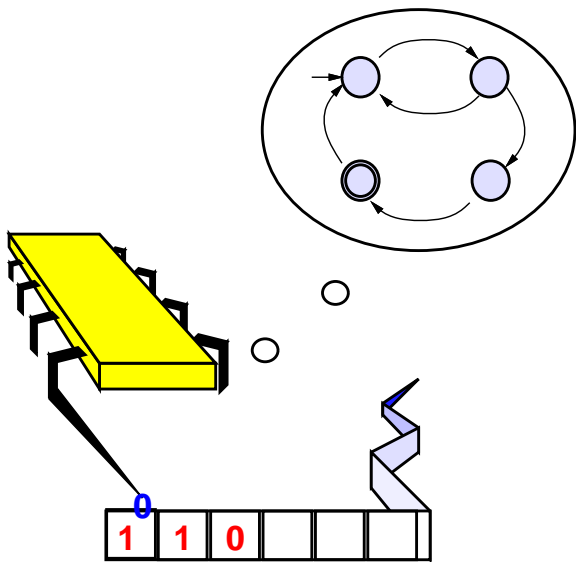
A Finite Automaton



A Pushdown Automaton



A Turing Machine



Part I

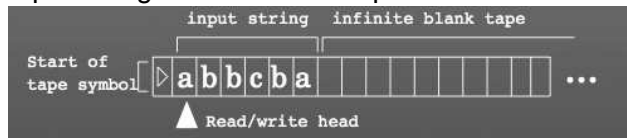
Turing Machines (TMs)

Turing Machines

- Machines so far (DFA, PDA) read input only once
- Turing Machines
 - ▶ Can back up over the input
 - ▶ Can overwrite the input
 - ▶ Can **write** information on tape and **come back** to it later

Turing Machines

- Input string is written on a tape:



- At each step, machine reads a symbol, and then
 - ▶ writes a new symbol, and
 - ▶ either moves read/write head to right,
 - ▶ or moves read/write head to left

TM vs. PDA vs. DFA

- A Turing machine can both write on the tape and read from it.
- A PDA is restricted to reading from the stack in LIFO manner.
- A DFA has no media for writing anything – it must all be in its **finite** state.
- The TM read-write head can move both to the left and to the right
- The TM read-write tape is **infinite** to the right
- The special final (accepting/rejecting) states of TM take **immediate effect** (so the head need not be at some special position)

Effects of A **Single Step**

- Changes current **state**,
- Changes **head position** and **tape content at current position**.

- Each step has very **local**, small effect.
- But many small effects can accumulate to a **meaningful** task.

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

Algorithm 1 (A TM deciding B (pseudocode))

- 1 Check for a single $\#$,
 - ▶ If false then **reject**.
- 2 Zig-zag across the tape, check identical letters, and replace them by X .
 - ▶ If not identical then **reject**.
- 3 When all the letters left of $\#$ are marked X , check for remaining letters right of $\#$
 - ▶ If there remaining letters are **reject**,
 - ▶ otherwise **accept**

Example $B = \{w\#w \mid w \in \{0, 1\}^*\}$ cont.

- Input: 0 1 1 0 0 0 # 0 1 1 0 0 0
- $\bar{0}$ 1 1 0 0 0 # 0 1 1 0 0 0
- $X \bar{1}$ 1 0 0 0 # 0 1 1 0 0 0
- $X 1$ 1 0 0 0 # $\bar{0}$ 1 1 0 0 0
- $X 1$ 1 0 0 0 # \bar{X} 1 1 0 0 0
- \bar{X} 1 1 0 0 0 # $X 1$ 1 0 0 0
- $X \bar{1}$ 1 0 0 0 # $X 1$ 1 0 0 0
- $X X \bar{1}$ 0 0 0 # $X 1$ 1 0 0 0
-
- $X X X X X X$ # $X X X X X X$

Turing Machine – Formal Definition

Definition 2 (Turing machine)

A Turing machine (TM) is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$, where

- Q is a finite set of **states**,
- Σ the **input alphabet** not containing the blank symbol \sqcup
- Γ is the **tape alphabet**, where $\sqcup \in \Gamma$ and $\Sigma \subset \Gamma$.
- $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is the **transition function**,
- $q_0 \in Q$ is the **start state**,
- $q_a \in Q$ is the **accept state**, and
- $q_r \in Q$ is the **reject state**.

Transition Function δ

$$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\} .$$

where $\delta(q, a) = (r, b, L)$ means: in state q where head reads tape symbol a , the machine:

- writes b over a ($a = b$ is possible),
- enters state r ,
- moves the head left (this is what the L stands for).
- If $\delta(q, a)$ is undefined, it means assume $\delta(q, a) = (q_r, \cdot, \cdot)$

Model of Computation

Before we start:

- an input of length n , $w = w_1 w_2 \dots w_n \in \Sigma^*$ is placed on n leftmost tape squares, one tape square per input letter.
Rest of tape contains blanks \sqcup
- since $\sqcup \notin \Sigma$, leftmost blank indicates end of input.
- read/write head is on leftmost square of tape

The “computation”:

M “computes” according to transition function δ .

Computation continues until q_a or q_r is reached. Otherwise M runs forever.

- If M tries to move head beyond left-hand-end of tape, it doesn't move (still M does not crash)

Configurations: The Yield Relation

- If $\delta(q_i, b) = (q_j, c, L)$, then configuration $uaq_i bv$ yields configuration $uq_j acv$.
- If $\delta(q_i, b) = (q_j, c, R)$, then configuration $uaq_i bv$ yields configuration $uacq_j v$.
- Special case (1): when head is at left end and tries to move left, it changes state and writes on tape but **does not move**, so if $\delta(q_i, b) = (q_j, c, L)$, configuration $q_i bv$ yields $q_j cv$.
- Special case (2): What happens when head is at right end? We let wq_i and $wq_i \sqcup$ denote the same configuration, so **moves to the right** can now be accommodated.

The new configuration is **longer** as it “annexed” one blank. This allows configurations to grow in length with computation. Yet at any given moment, they are **finitely** long.

Special Configurations

- Starting configuration: $q_0 w$
- Accepting configuration: $w_0 q_a w_1$
- Rejecting configuration: $w_0 q_r w_1$
- Halting configurations: $w_0 q_a w_1$ and $w_0 q_r w_1$

Accepting a Language

A Turing machine M **accepts** an input w , if there is a sequence of configurations C_1, C_2, \dots, C_k such that

- C_1 is the start configuration of M on w ,
- each C_i **yields** C_{i+1} ,
- C_k is an accepting configuration.

The collection of strings **accepted by** M is called the **language** of M , and is denoted $L(M)$.

Enumerable Languages

Definition 3

A language L is called **recursively enumerable** (\mathcal{RE}), if some Turing machine **accepts** L .

(In the book called **Turing-recognizable**)

On an input, w , a TM may

- **accept**
- **reject**
- **loop** (run forever)

Major concern: In general, we never know if the TM **will halt**.

Decidable Languages

Definition 4

A TM M **decides** a language L , if it accepts it, and M halts on **every** input in Σ^* .

Namely the TM either reaches state q_a (in case $w \in L(M)$) or it reaches state q_r (in case $w \notin L(M)$), but it **does not loop**.

Such TM is called a **decider**.

Definition 5

A language L is **decidable** (\mathcal{R}), if some Turing machine decides it. (In the book called **Turing-decidable**, also known as, **recursive**)

Knowing when Reaching the Left End

Question 6

To implement algorithms, we sometimes should be able to tell when a TM is at the left end of the tape.

Answer: Mark it with a special symbol. When head reads this symbol, TM “knows” it is on the leftmost tape square.

Part II

Some Examples

Example $A = \{0^{2^n} : n \geq 0\}$

Algorithm 7 (A TM deciding A (pseudocode))

On input w :

- 1 Move to the right, erasing every other 0
- 2 **Accept** if tape contains a single 0.
- 3 **Reject** if tape contains an odd number of 0 (greater than one).
- 4 Return to the start of the tape, and go to **Step ??**

Example $A = \{0^{2^n} : n \geq 0\}$ – TM formal definition

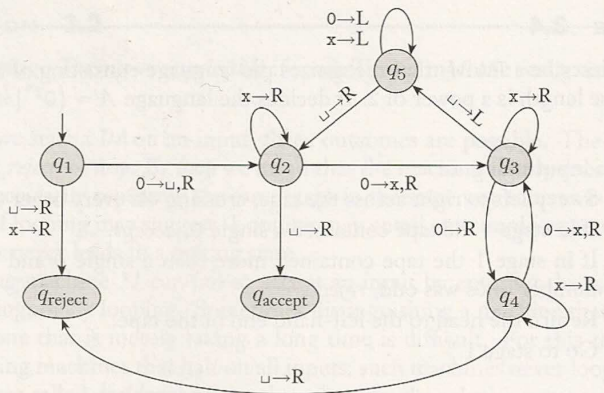
Definition 8 ($M_2 = (Q, \Sigma, \Gamma, \delta, q_1, q_a, q_r)$)

- $Q = \{q_1, q_2, q_3, q_4, q_5, q_a, q_r\}$
- $\Sigma = \{0\}$ input alphabet
- $\Gamma = \{0, x, \sqcup\}$ tape alphabet
- q_1 start state
- q_a accept state
- q_r reject state

Example $A = \{0^{2^n} : n \geq 0\}$ – Transition Function

132

CHAPTER 3 / THE CHURCH-TURING THESIS

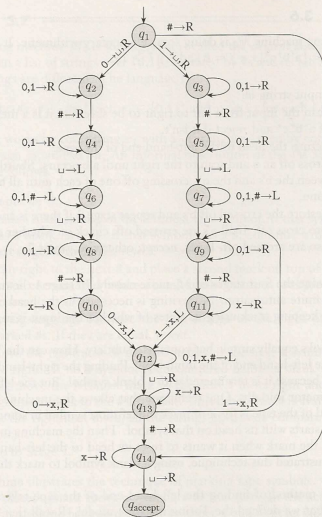


Example $B = \{w\#w \mid w \in \{0, 1\}^*\}$ – TM formal definition

Definition 9 ($M_1 = (Q, \Sigma, \Gamma, \delta, q_1, q_a, q_r)$)

- $Q = \{q_1, \dots, q_{14}, q_a, q_r\}$.
- $\Sigma = \{0, 1, \#\}$ input alphabet
- $\Gamma = \{0, 1, \#, x, \sqcup\}$ tape alphabet
- q_1 start state
- q_a accept state
- q_r reject state

Example $B = \{w\#w : w \in \{0, 1\}^*\}$ – Transition Function



Example: $C = \{a^i b^j c^k : i \cdot j = k \wedge i, j, k \geq 1\}$

Algorithm 10 (A TM deciding C (pseudocode))

- 1 Scan from left to right to check that input is $a^+ b^+ c^+$
- 2 Return to start of tape
- 3 Cross off one a and scan right until b occurs.
Shuttle between b 's and c 's, crossing off one of each, until all b 's are gone.
- 4 **Restore** the crossed-off b 's and repeat previous step if more a 's exist.
If all a 's crossed off, check if all c 's crossed off. If yes, **accept**, otherwise **reject**.

Example: Element Distinctness

Consider the **element distinctness** problem

$$E = \{\#x_1\#x_2\#\dots\#x_\ell : x_i \in \{0, 1\}^* \wedge i \neq j \implies x_i \neq x_j\}$$

Verbally,

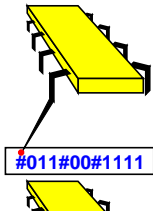
- List of strings in $\{0, 1\}^*$ separated by $\#$'s.
- List is in language (& machine **should** accept), if all strings are **different**.

Decider for $E = \{\#x_1\#x_2\#\dots\#x_\ell : x_i \in \{0,1\}^* \wedge i \neq j \implies x_i \neq x_j\}$

Algorithm 11 (A TM deciding E (pseudocode))

Input: w

- 1 Place a "mark" on leftmost tape symbol. If symbol not $\#$, **reject**.
- 2 Scan right to next $\#$ and place mark on top. If none, **reject**.
Compare the two strings to right of marked $\#$'s (**how?**). If equal, **reject**.
- 4 If possible, move rightmost mark to next $\#$ on right and go to **Step ??**.
- 5 If possible, move leftmost mark to next $\#$ on right and rightmost mark to $\#$ after that, and go to **Step ??**.
- 6 **Accept**.



Marking A symbol

Question 12

How do we “mark” a symbol?

Answer: For each tape symbol $\#$, add tape symbol $\overset{\bullet}{\#}$ to the tape alphabet Γ .

Part III

Alternative Definitions

Turing Machine Variants

For example, suppose the Turing machine head is allowed to **stay put**.

$$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, S\}$$

Question 13

Does this add any power?

Answer: No. Replace each **S** transition with two transitions: **R** then **L**.
(ahmm ... why not vice-versa?)

Important notion here: Two-way **emulation** (model **A** capable of emulating model **B**; model **B** capable of emulating model **A**).

Multitape Turing Machines

- **Constant** number of infinite tapes
- Each tape has its own head
- Initially, input string on tape **1** and rest blank

For the transition function: $\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L, R\}^k$, the expression $\delta(q_i, a_1, \dots, a_k) = (q_j, b_1, \dots, b_k, L, R, \dots, L)$ means:

Assuming the machine is in state q_i and heads **1** through k reading a_1, \dots, a_k :

- the machine goes to state q_j ,
- heads **1** through k write b_1, \dots, b_k ,
- **each** head moves **right** or **left** as specified.

Equivalence Of **Multitape** TM and **Singletape** TM

Theorem 14

A **singletape** TM can emulate a **multitape** TM.

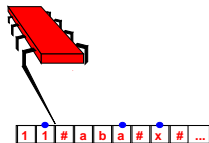
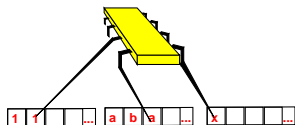
Corollary 15

A language is enumerable [resp., decidable], if and only if there is some **multitape** Turing machine that accepts [resp., decides] it.

One direction is trivial.

To prove the other direction, we will show how to convert a **multitape** TM, M , into an equivalent **singletape** TM, S .

Emulation Idea



- S emulates k tapes of M by storing them all on a **single tape** with delimiter $\#$.
- S marks the current positions of the k heads by placing • “above” the letters in current positions. It “knows” which tape the mark belongs to by counting (up to k) from the $\#$ ’s to the left.

The Emulator

Algorithm 16 (The Emulator (pseudocode))

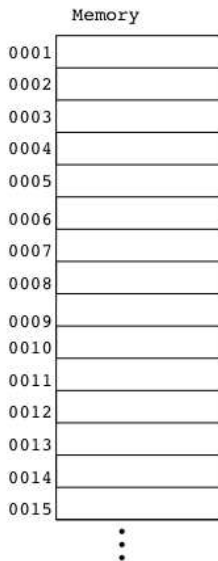
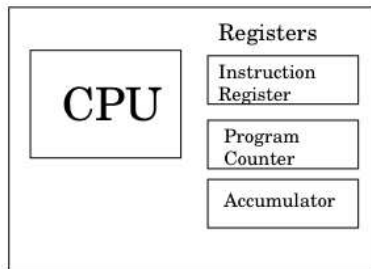
On input $w = w_1 \cdots w_n$:

- 1 Write on the tape $\# \overset{\bullet}{w}_1 w_2 \cdots w_n \# \sqcup \# \sqcup \# \cdots \#$
- 2 Scan the tape from first $\#$ to $(k + 1)$ -st $\#$ to read symbols under “virtual” heads.
- 3 Rescan the tape to write new symbols and move heads
- 4 When M moves head onto unused blank square, S will try to move virtual head onto $\#$.
 S handles it by writing blank \sqcup on tape, and shifting the rest of tape one square to the right.

Random Access Machine (RAM)

- CPU
- 3 Registers (Instruction Register (IR), Program Counter (PC), Accumulator (ACC))
- Memory
- Operation:
 - ▶ Increment PC
 - ▶ Set $IR \leftarrow \text{MEM}[PC]$
 - ▶ Execute *instruction* in IR
 - ▶ Repeat
- *Instructions* are typically compare, add/subtract, multiply/divide, shift left/right.
- We assume no limit on the registers' *size*
- All instructions are doable on a TM.

RAM: Schematic Picture



RAM: Instructions

	Instruction	Meaning
00	HALT	Stop Computation
01	LOAD x	$ACC \leftarrow MEM[x]$
02	LOADI x	$ACC \leftarrow x$
03	STORE x	$MEM[x] \leftarrow ACC$
04	ADD x	$ACC \leftarrow ACC + MEM[x]$
05	ADDI x	$ACC \leftarrow ACC + x$
06	SUB x	$ACC \leftarrow ACC - MEM[x]$
07	SUBI x	$ACC \leftarrow ACC - x$
08	JUMP x	$PC \leftarrow x$
09	JZERO x	$PC \leftarrow x$ if $ACC = 0$
10	JGT x	$PC \leftarrow x$ if $ACC > 0$

RAM: Example Program

A program that multiplies two numbers (in locations 1000 & 1001), and stores the result in 1002

Memory	Machine Code	Assembly
0001	011000	LOAD 1000
0002	031003	STORE 1003
0003	020000	LOADI 0
0004	031002	STORE 1002
0005	021003	LOAD 1003
0006	090013	JZERO 0013
0007	070001	SUBI 1
0008	031003	STORE 1003
0009	011002	LOAD 1002
0010	041001	ADD 1001
0011	080004	JUMP 4
0013	000000	HALT

TM is (at least) as strong as RAM

Theorem 17

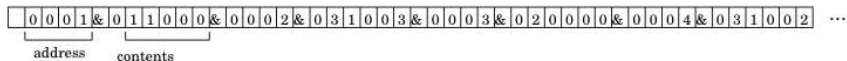
A *Turing machine* can emulate this *RAM* model.

Proof's idea: We can emulate the RAM model with a multi-tape Turing machine:

- One tape for each register (IR, PC, ACC)
- One tape for the Memory
- Memory tape will be entries of the form $\langle \text{address} \rangle \langle \text{contents} \rangle$ in increasing order of addresses.

Emulator's Tapes

Memory



Instruction Pointer



Instruction Register



Accumulator



Algorithm 18 (Emulator (high-level description))

- 1 Scan through memory until reach an address that matches the PC
- 2 Copy contents of memory at that address to the IR
- 3 Increment PC
- 4 Based on the instruction code:
 - ▶ Copy a value into PC
 - ▶ Copy a value into Memory
 - ▶ Copy a value into the ACC
 - ▶ Perform an arithmetic operation, a shift, or a comparison

Beyond RAM

- A **RAM** can be modeled (emulated) by a Turing Machine.
- Any current machine (architecture, manufacturer, operating system, power supply, etc.) can be modeled by a Turing Machine.
- If there is an algorithm for it, a Turing Machine can do it.
- Note that at this point, we don't care **how long** it might take, just that it can be done.

Turing Completeness

- A computation model is called “Turing Complete” if it can emulate a (general) Turing Machine.
- Turing Complete \Rightarrow can compute anything a TM could
 - ▶ Of course it might **not** be convenient ...
- Church-Turing Thesis: any computational model can be emulated by a Turing machine.
- More next lecture

Non-Deterministic Turing Machines

Transition function:

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

- Computation of a **deterministic** TM can be viewed as a **path** in configuration space.
- Computation of a **non-deterministic** TM (NTM) can be viewed as a **tree** in configuration space.
- NTM accepts an input if **there is** (\exists) an accepting branch in its computation tree.
- NTM rejects an input if **all** (\forall) branches in its computation tree are either rejecting or infinite (looping).

Equivalence of TM and NTM

Theorem 19

A Turing Machine can emulate a *non-deterministic* Turing machine.

Corollary 20

A language is enumerable [resp., recursive], if and only if there is some *non-deterministic* Turing machine that accepts [resp., decides] it.

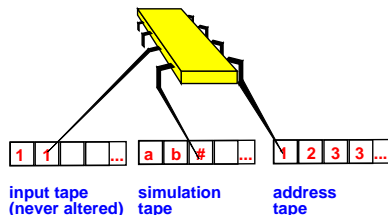
One direction is trivial.

To prove the other direction, we will show how to convert a *non-deterministic* TM, N , into an equivalent *deterministic* TM, D .

Basic idea

- D tries all possible branches
- If D finds any **accepting** branch, it **accepts**.
- If all branches **reject**, D **rejects**.
- If all branches **reject** or **loop**, D **loops**.

Emulator's Tapes



The emulating machine D has three tapes

- **Input** tape is never altered (only read from),
- **Emulation** tape serves as N 's tape,
- **Address** tape keeps track of D 's location in N 's *computation tree*.

Computation Tree

We view N 's **non-deterministic** computation as a tree in the configurations' space.

- Each tree branch is a branch of N 's non-deterministic computation
- Each tree node is a **configuration** of N
- **Root** is starting configuration
- The number of children of each node, denoted by b , is **at most** the number of N 's states, times the size of Γ , times 2 (left/right).

D traverses this tree until it finds an **accepting** configuration (if any).

Question 21

How to traverse this tree?

- depth-first search?
- breadth-first search?

Address Tape

The address tape contains a **pointer** into the configuration tree. Concretely, a string in Σ_b^* , where $\Sigma_b = \{1, \dots, b\}$.

Definition 22

By **incrementing** the value of the address tape, we mean increasing its value by 1 (in base b).

Question 23

Can a TM implement the increment function?

Question 24

Can a TM compute the value of the node indexed by the address tape (with respect to to a given input w)?

Algorithm 25 (Emulator (pseudocode))

- ① Compute the configuration of N indexed by the *address* tape :
 - ① Copy *input* tape (i.e., w) to *emulation* tape.
 - ② Use *emulation* tape to emulate the run of N on w , using the *address* tape to resolve non-deterministic choices.
Break current emulation, if
 - ★ symbols on *address* tape are exhausted (i.e., end of branch)
 - ★ non-deterministic choice is invalid
 - ★ **rejecting** configuration was reached
- ② **Accept** if accepting configuration reached.
- ③ **Increment** the value of *address* tape.
- ④ Go back to **Step ??**.