Theory of Automata
Post Machine

Hafiz Tayyeb Javed

Week 15 Lecture 02

Contents

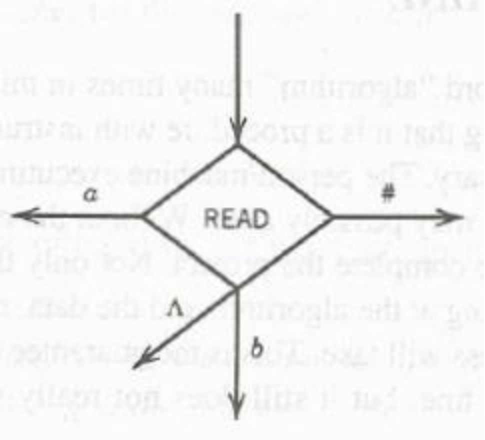
- Post Machine
- Examples
- Simulating PM on TM
- Simulating TM on PM
  - SHIFT-RIGHT CYCLICALLY.

POST Machine

- **A Post machine** denoted by **PM** is a collection of five things
  1. The alphabet Σ plus the special symbol #. We generally use Σ = { a, b}.
  2. **A** linear storage location (a place where a string of symbols is kept called the **STORE or QUEUE,** which initially contains the input string. This location can be read by which we mean the leftmost character can be removed for inspection. The STORE can also be added to, which means a new character can be concatenated onto the right of whatever is there already. We allow for the possibility that characters not in Σ can be used in the **STORE,** characters from an alphabet Γ called the store alphabet.

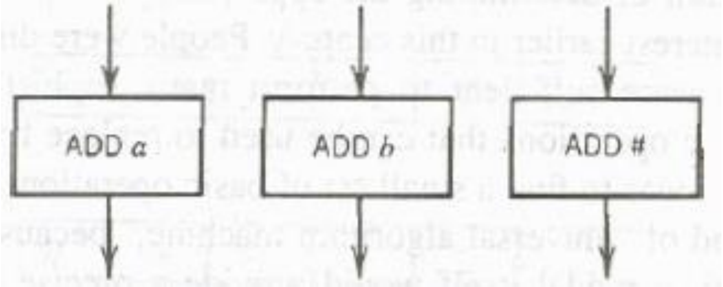
POST Machine Contd.

3. READ states, for example,



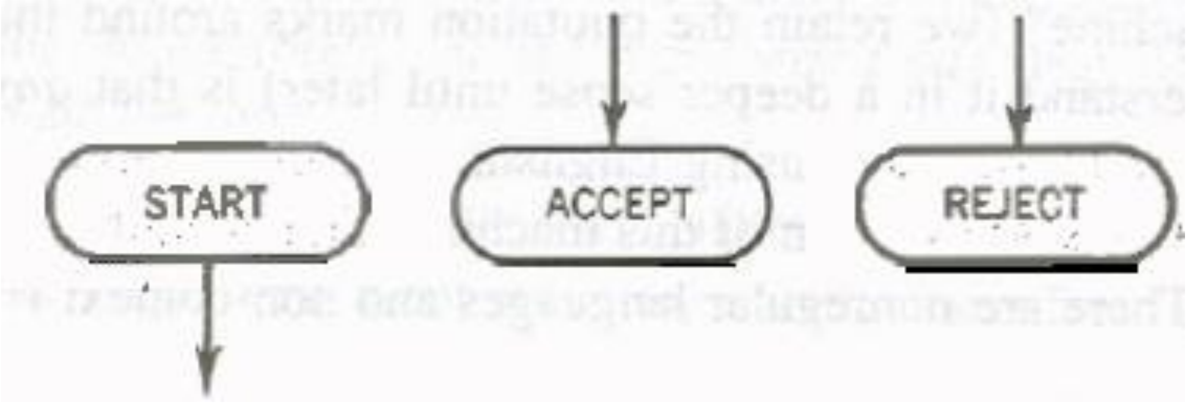
PMs **are** deterministic, so no two edges from the **READ** have the same label

4. ADD states:



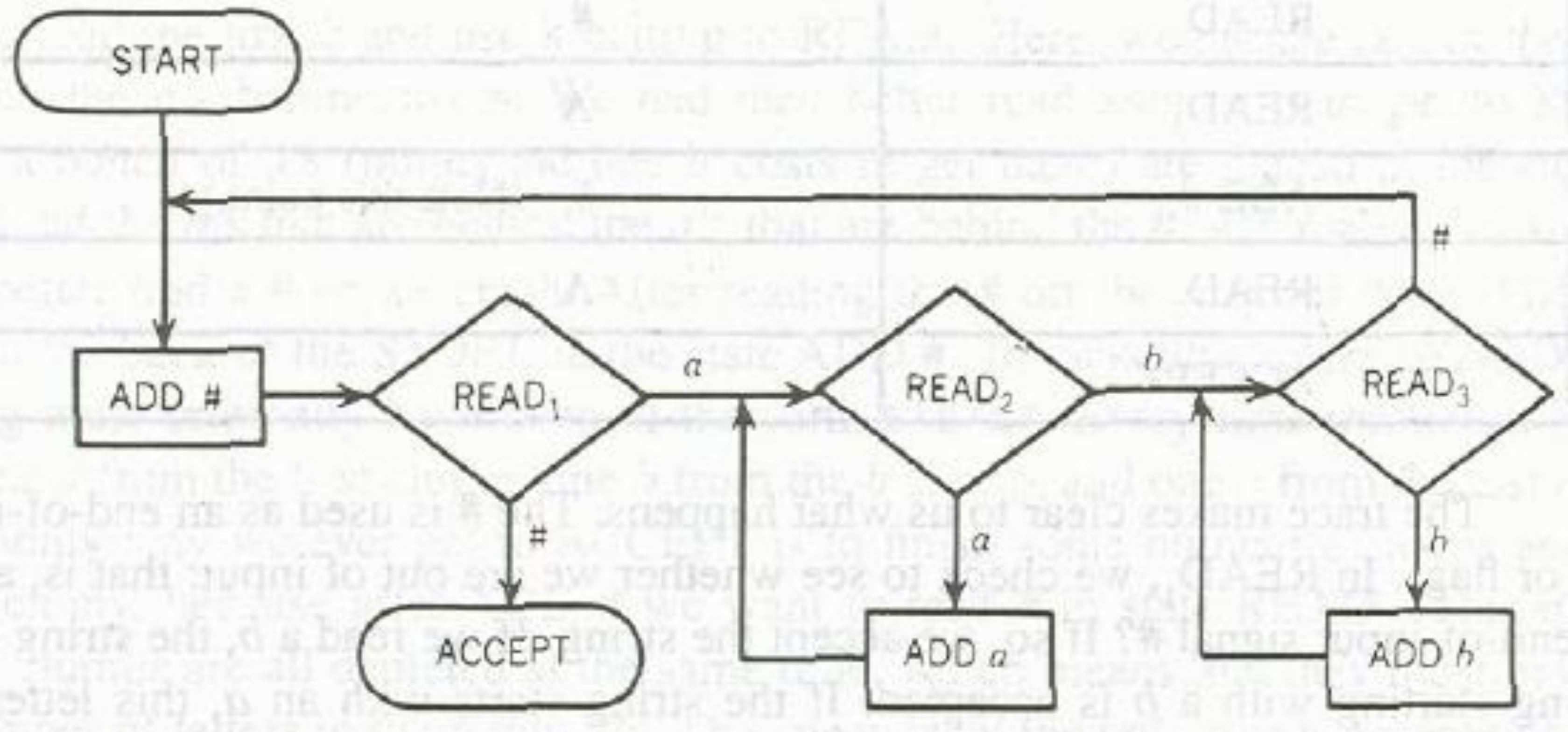
POST Machine Contd.

5. **A START** state (un-enterable) and some halt states called **ACCEPT and REJECT**



Example

- Consider the following PM and guess the language?



Simulating a PM on TM

- Theorem: 47

  A language accepted by **PM** can also be accepted by **TM**.

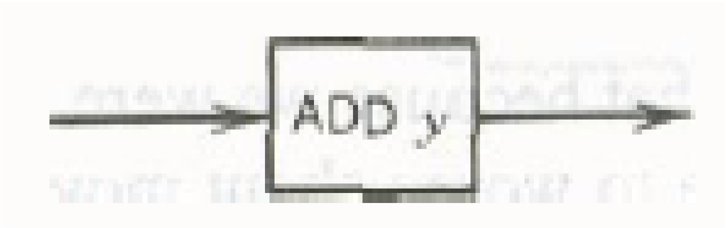
Proof:

We can simulate a PM over TM. PM consists of many components. By simulating we means, we will show how to convert each of these components into corresponding TM components that function the same way.

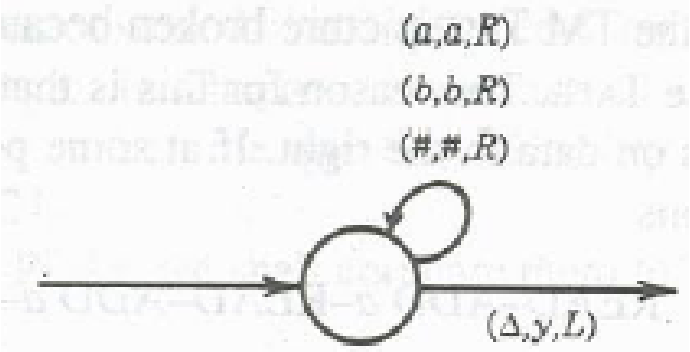
Simulating PM on TM

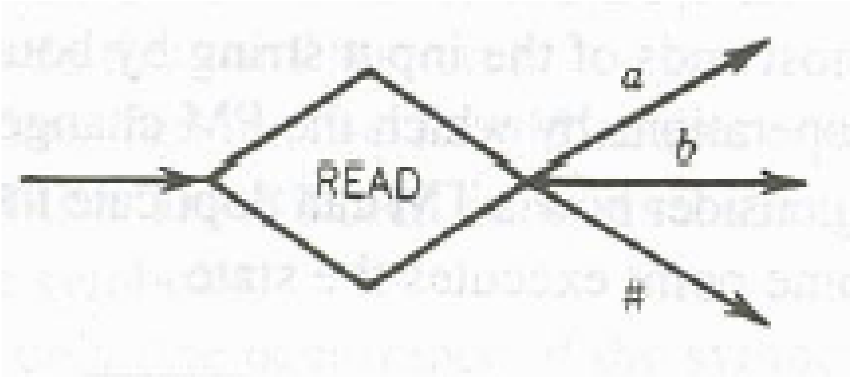
- Start state of PM is same as TM’s start state. So no conversion needed.
- Accept state of PM will be renamed as HALT state of TM.
- REJECT state of PM will be converted by deleting the REJECT states and all the edges coming into REJECT states.
- Now how to keep track of PM store contents when simulating over TM tape?

HOW to simulate STORE over TAPE?

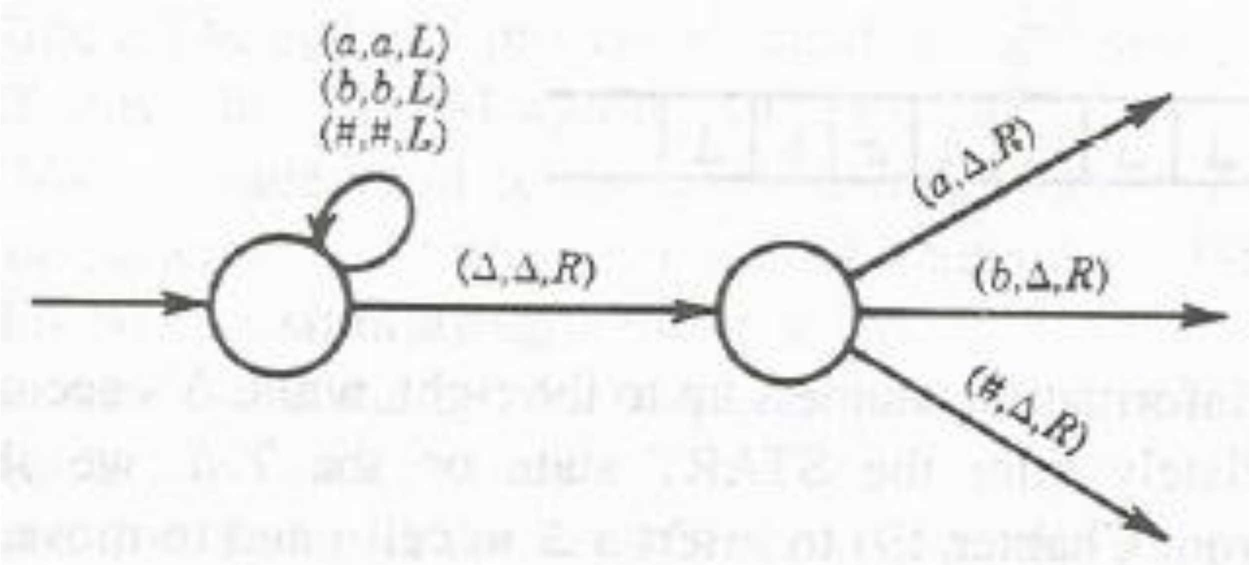


- Can be simulated by

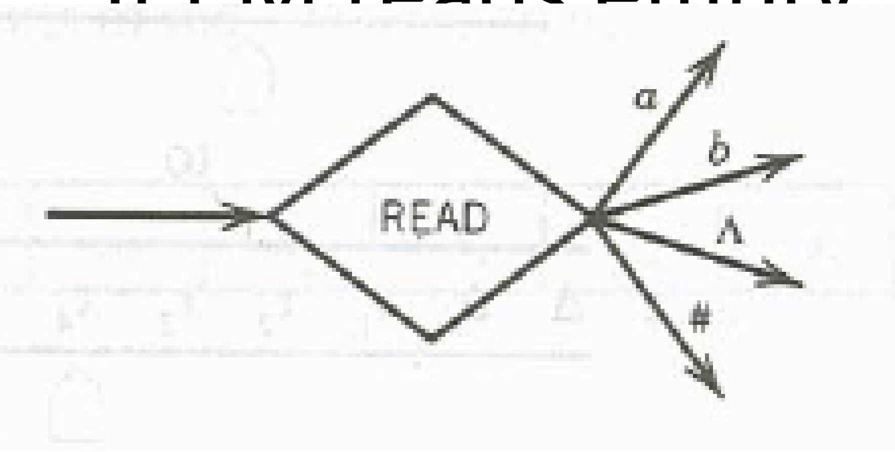




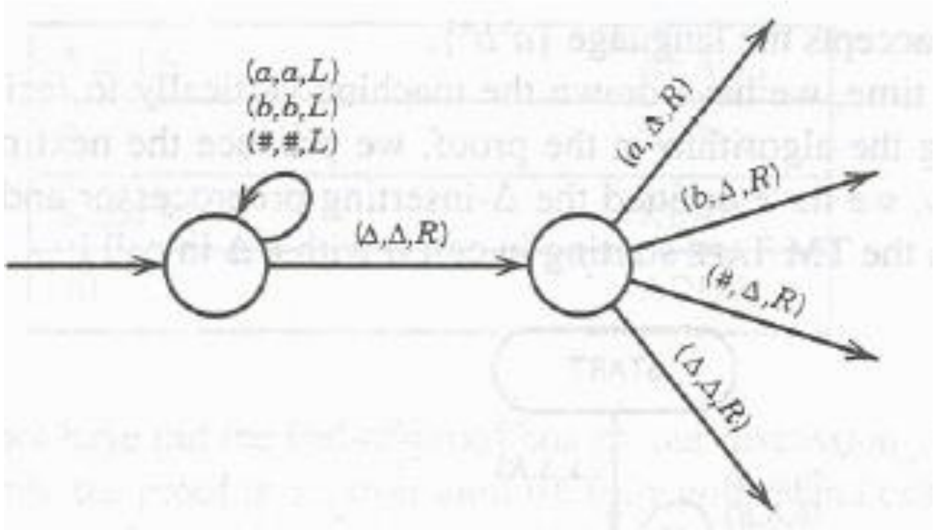
- Can be simulated by

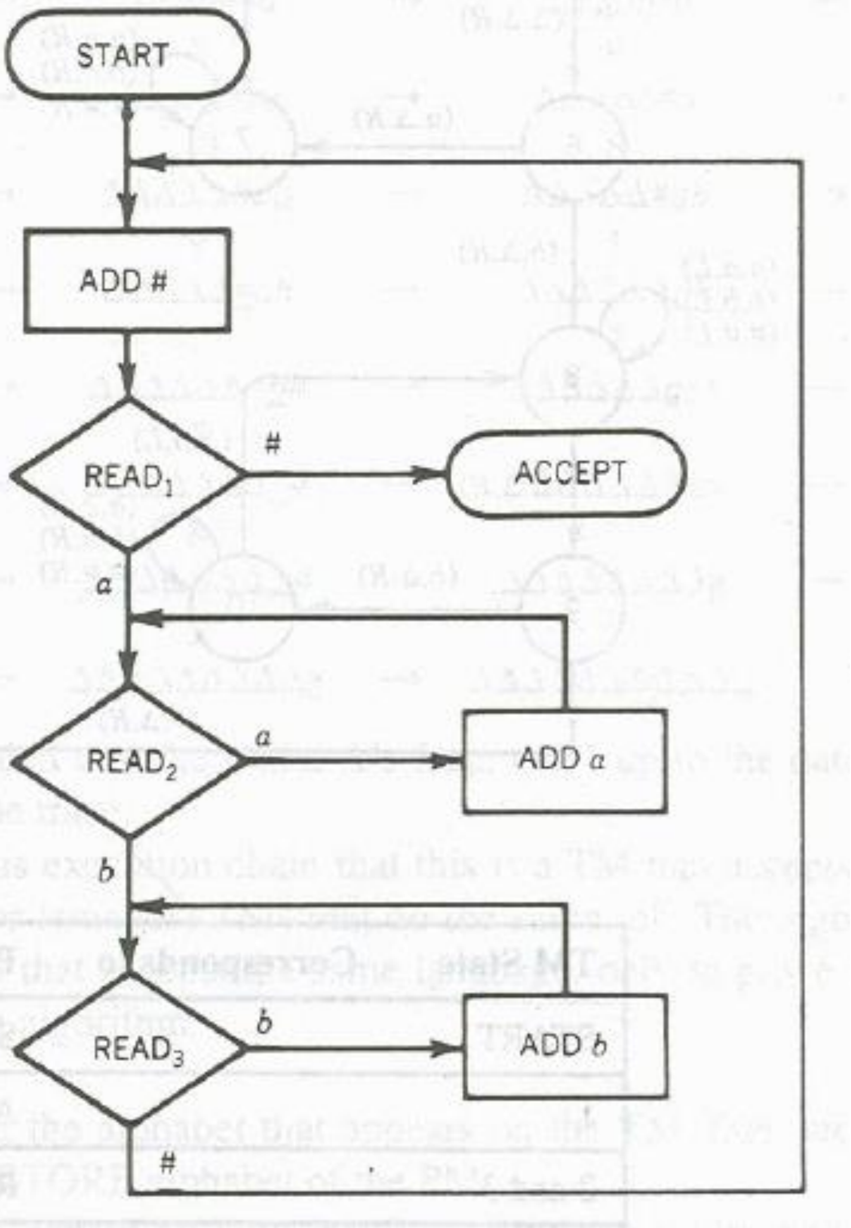


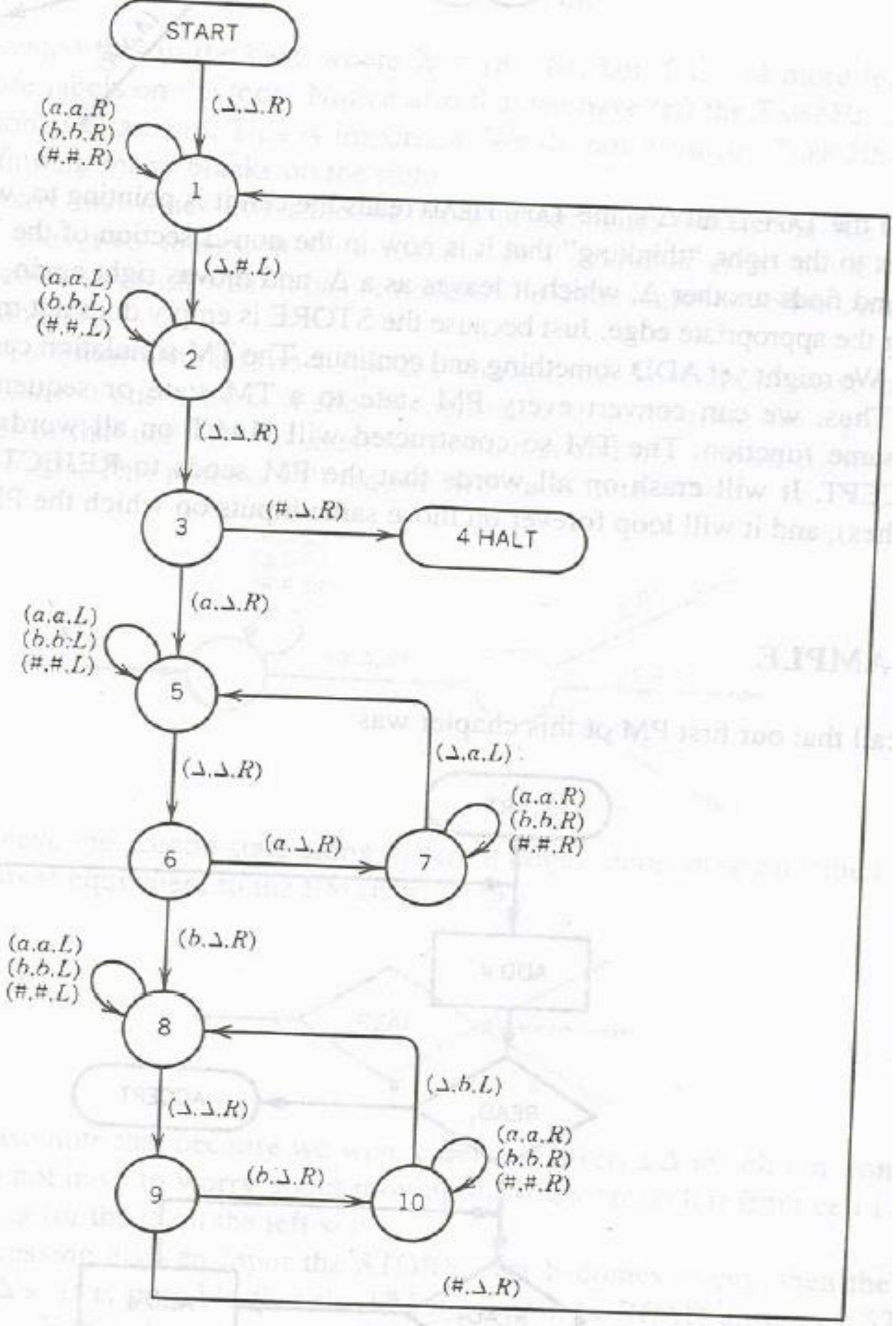
If PM reads empty STORE



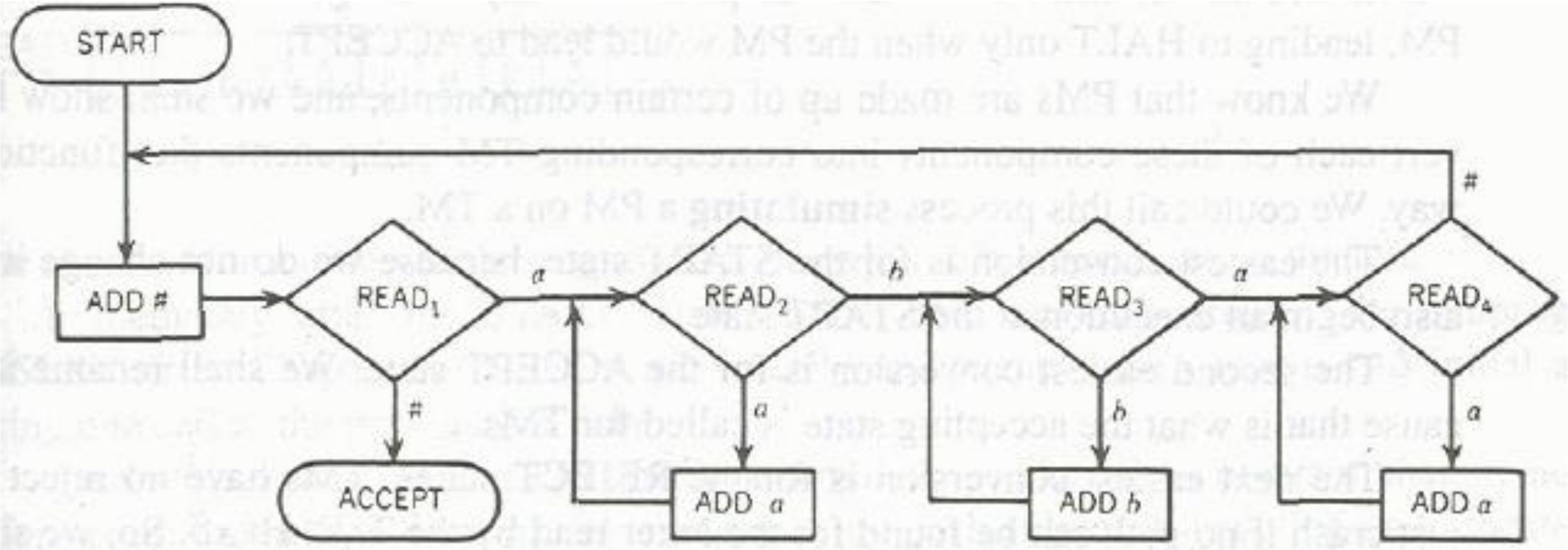
Then it becomes




START
ADD #
READ$_1$
#
ACCEPT
a
READ$_2$
a
ADD a
b
READ$_3$
b
ADD b
#

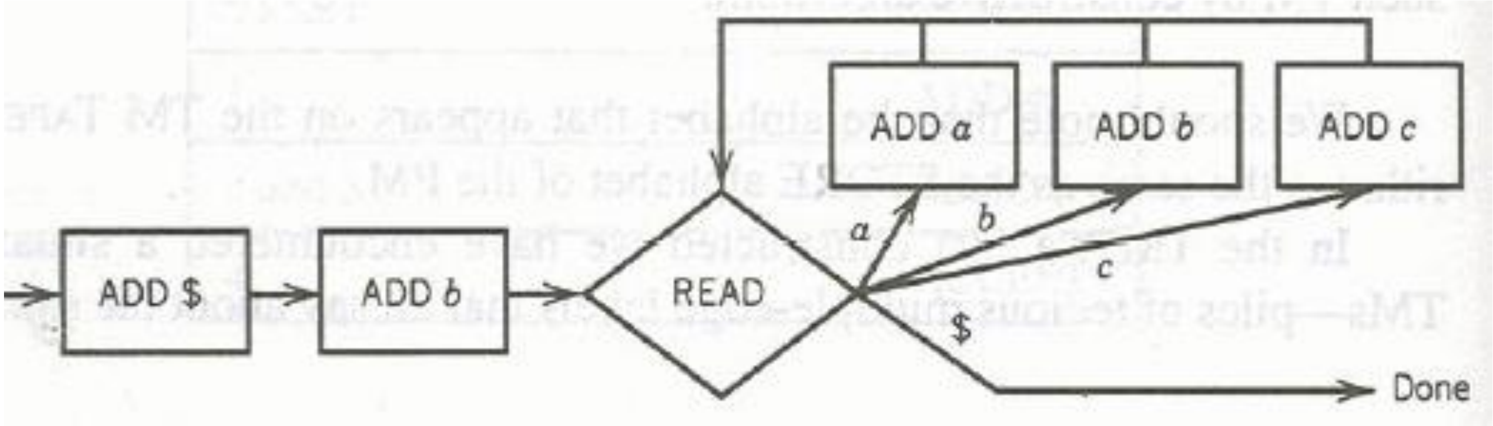

START
(Δ,Δ,R)
(a,a,R)
(b,b,R)
(#,#,R)
1
(Δ,#,L)
(a,a,L)
(b,b,L)
(#,#,L)
2
(Δ,Δ,R)
3
(#,Δ,R)
4 HALT
(a,Δ,R)
(a,a,L)
(b,b,L)
(#,#,L)
5
(Δ,a,L)
(Δ,Δ,R)
(a,a,R)
(b,b,R)
(#,#,R)
6
(a,Δ,R)
7
(b,Δ,R)
(a,a,L)
(b,b,L)
(#,#,L)
8
(Δ,b,L)
(Δ,Δ,R)
(a,a,R)
(b,b,R)
(#,#,R)
9
(b,Δ,R)
10
(#,Δ,R)

Design a Turing Machine from POST Machine



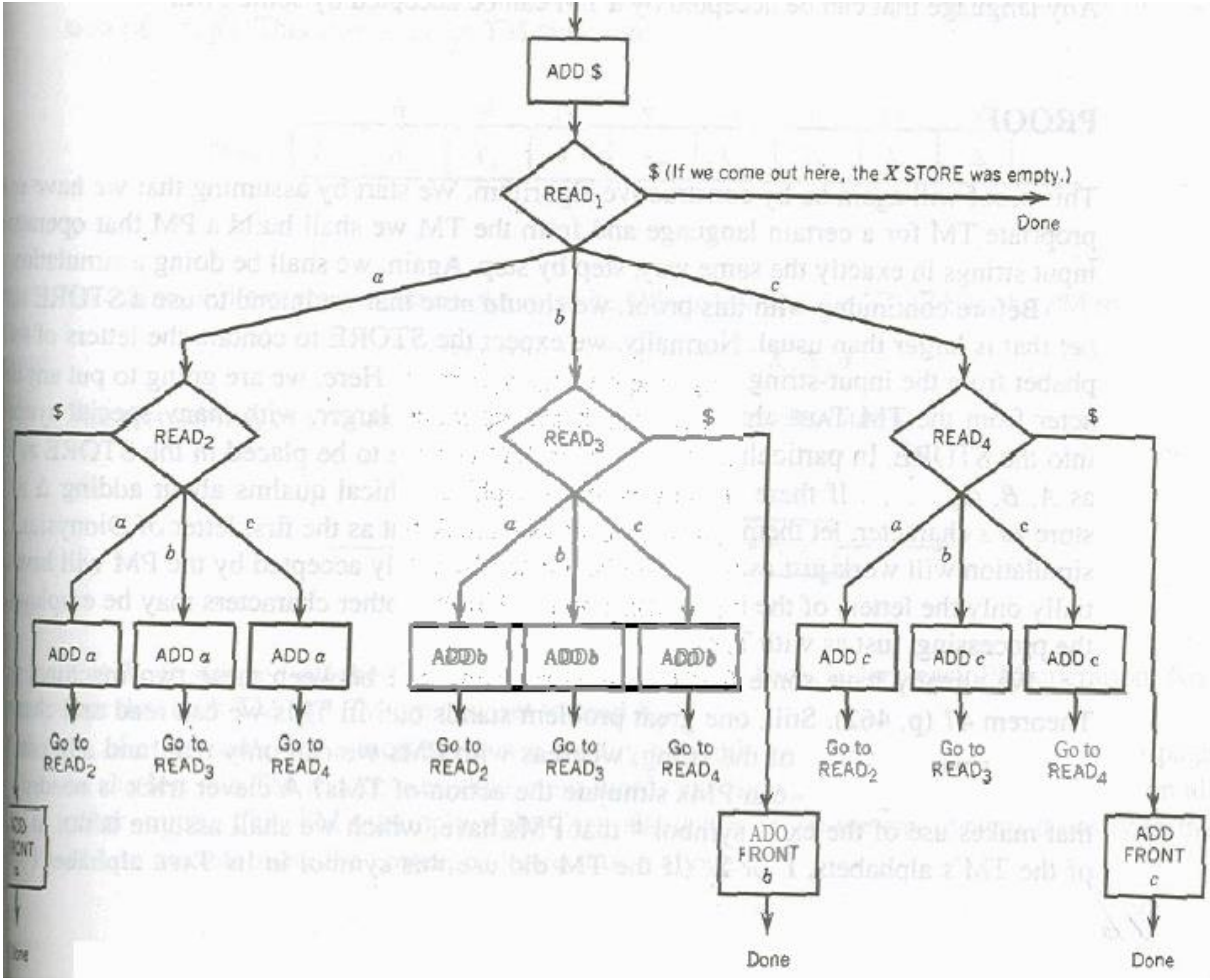
ADD FRONT

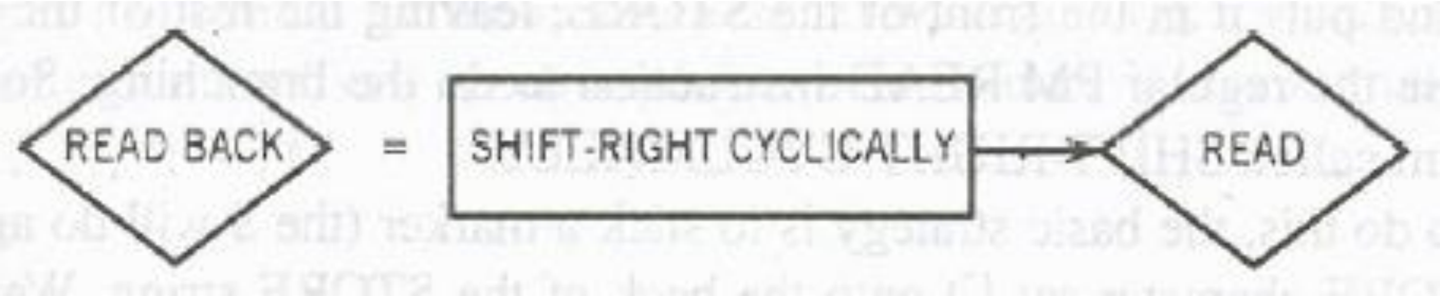
- ADD FRONT: There are subprograms that can enable PM to add character to the front (left end) of the string in the STORE



SHIFT-RIGHT CYCLICALLY.

- Read the character off the back (right end) of the string.
- The subprogram that reads the back character from the STORE and braches accordingly be known as **SHIFT-RIGHT CYCLICALLY**.


ADD $
READ$_1$
$ (If we come out here, the X STORE was empty.)
Done
a
b
c
$
READ$_2$
a
b
c
ADD a
ADD a
ADD a
Go to READ$_2$
Go to READ$_3$
Go to READ$_4$
ADD FRONT a
Done
READ$_3$
$
a
b
c
ADD b
ADD b
ADD b
Go to READ$_2$
Go to READ$_3$
Go to READ$_4$
ADD FRONT b
Done
READ$_4$
$
a
b
c
ADD c
ADD c
ADD c
Go to READ$_2$
Go to READ$_3$
Go to READ$_4$
ADD FRONT c
Done


READ BACK
=
SHIFT-RIGHT CYCLICALLY
READ

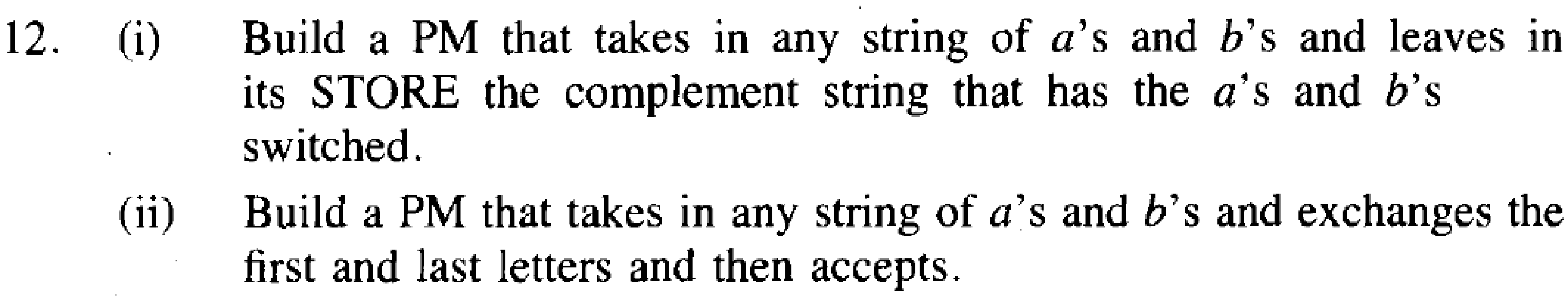
Theorem 49

A Language that can be accepted by **TM** can also be accepted by **PM**.

Proof:

Proof is left for your exercise…..

Exercise Questions



12. (i) Build a PM that takes in any string of *a*'s and *b*'s and leaves in its STORE the complement string that has the *a*'s and *b*'s switched.

    (ii) Build a PM that takes in any string of *a*'s and *b*'s and exchanges the first and last letters and then accepts.