Chapter 19

Turing Machines

19.1 Introduction

- Turing machines will be our ultimate model for computers, so they need output capabilities.
- But computers without output statements can tell us something.
- Consider the following program

  1. READ X
  2. IF X=1 THEN END
  3. IF X=2 THEN DIVIDE X BY 0
  4. IF X>2 THEN GOTO STATEMENT 4

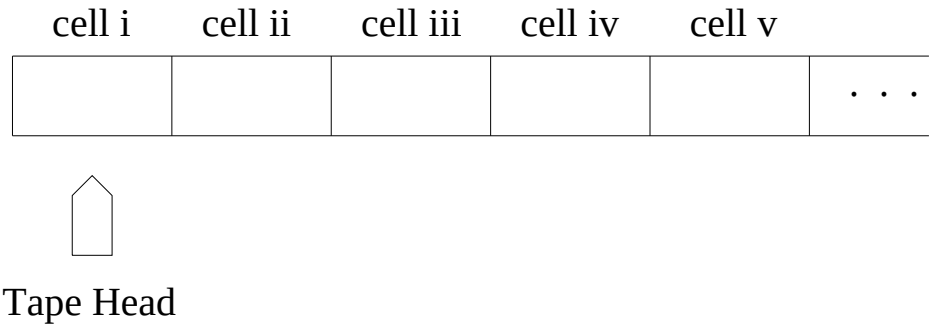
- If we assume that the input is always a positive integer, then
  - if program terminates naturally, then we know X was 1.
  - if program terminates with error message saying there is an overflow (i.e., crashes), then we know X was 2.
  - if the program does not terminate, then we know X was greater than 2.

Definition: A *Turing machine* (TM) $T = (\Sigma, \Theta, \eta, \Gamma, K, s, H, \Pi)$, where

1. An alphabet Σ of input letters, and assume that the blank $\Delta \notin \Sigma$.

2. A *Tape* Θ divided into a sequence of numbered cells, each containing one character or a blank.

   - The input word is presented to the machine on the tape with one letter per cell beginning in the leftmost cell, called cell i.
   - The rest of the Tape is initially filled with blanks Δ.
   - The Tape is infinitely long in one direction.



3. A *Tape Head* η that can in one step read the contents of a cell on the Tape, replace it with some other character, and reposition itself to the next cell to the right or to the left of the one it has just read.

   - At the start of the processing, the Tape Head always begins by reading the input in cell i.
   - The Tape Head can never move left from cell i. If it is given orders to do so, the machine crashes.
   - The location of the Tape Head is indicated as in the above picture.

4. An alphabet Γ of characters that can be printed on the Tape Θ by the Tape Head η.

   - Assume that $\Delta \notin \Gamma$, and we may have that $\Sigma \subset \Gamma$.
   - The Tape Head may erase a cell, which corresponds to writing Δ in the cell.

5. A finite set K of states including

   - Exactly one START state $s \in K$ from which we begin execution (and which we may reenter during execution).

- $H \subset K$ is a set of HALT states, which cause execution to terminate when we enter any of them. There are zero or more HALT states.
- The other states have no function, only names such as $q_1, q_2, q_3, \ldots$ or $1, 2, 3, \ldots$.

6. A *program* Π, which is a finite set of rules that, on the basis of the state we are in and the letter the Tape Head has just read, tells us

   (a) how to change states,

   (b) what to print on the Tape,

   (c) where to move the Tape Head.

   The program

   $$\Pi \subset K \times K \times (\Sigma + \Gamma + \{\Delta\}) \times (\Gamma + \{\Delta\}) \times \{L, R\},$$

   with the restriction that

   - if $(q_1, q_2, \ell, c, d) \in \Pi$ and $(q_1', q_2', \ell', c', d') \in \Pi$ with $q_1 = q_1'$ and $\ell = \ell'$, then $q_2 = q_2'$, $c = c'$ and $d = d'$;
   - i.e., for any state q_1 and any character $\ell \in \Sigma + \Gamma + \{\Delta\}$, there is only one arc leaving state q_1 corresponding to reading character ℓ from the Tape.
   - This restriction means that TMs are deterministic.

   We depict the program as a collection of directed edges connecting the states. Each edge is labeled with the triplet of information:

   (character, character, direction) $\in (\Sigma + \Gamma + \{\Delta\}) \times (\Gamma + \{\Delta\}) \times \{L, R\}$

   where

   - The first character (either Δ or from Σ or Γ) is the character the Tape Head reads from the cell to which it is pointing.
     - From any state, there can be at most one arc leaving that state corresponding to Δ or any given letter of $\Sigma + \Gamma$;
     - i.e., there cannot be two arcs leaving a state both with the same first letter (i.e., a Turing machine is deterministic).
   - The second character (either Δ or from Γ) is what the Tape Head prints in the cell before it leaves.

- The third component, the direction, tells the Tape Head whether to move one cell to the right, R, or one cell to the left, L.

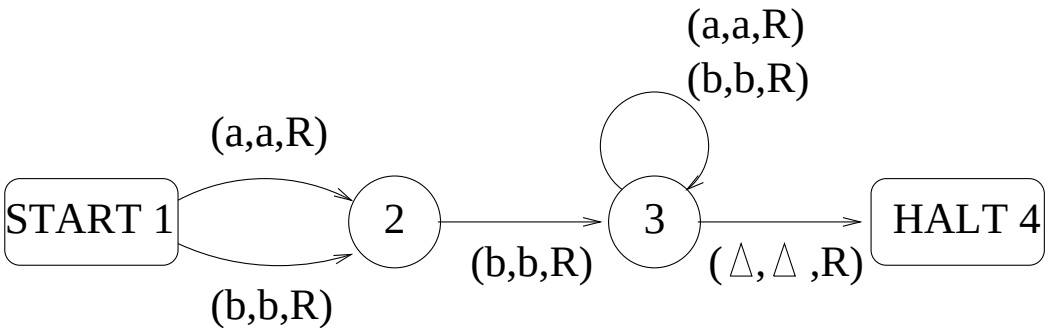
Remarks:

- The above definition does not require that every state has an edge leaving it corresponding to each letter of $\Sigma + \Gamma$.
- If we are in a state and read a letter for which there is no arc leaving that state corresponding to that letter, then the machine crashes. In this case, the machine terminates execution unsuccessfully.
- To terminate execution successfully, machine must be led to a HALT state. In this case, we say that the word on the input tape is *accepted* by the TM.
- If Tape Head is currently in cell i and the program tells the Tape Head to move left, then the machine crashes.
- Our definition of TM's requires them to be deterministic. There are also non-deterministic TM's. When we say just "TM", then we mean our above definition, which means it is deterministic.

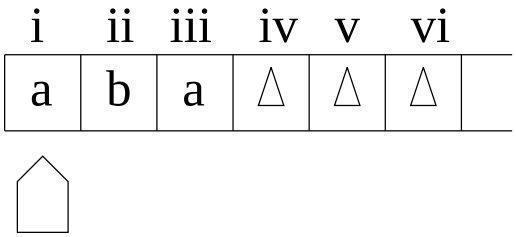
Definition: A string $w \in \Sigma^*$ is *accepted* by a Turing machine if the following occurs: when w is loaded onto the Tape and the machine is run, the TM ends in a Halt state.

Definition: The language *accepted* by a Turing machine is the set of accepted strings $w \in \Sigma^*$.

Example: Consider the following TM with input alphabet $\Sigma = \{a, b\}$ and tape alphabet $\Gamma = \{a, b\}$:



and input tape containing input *aba*



- We start in state START 1 with the Tape Head reading cell i, and we denote this by

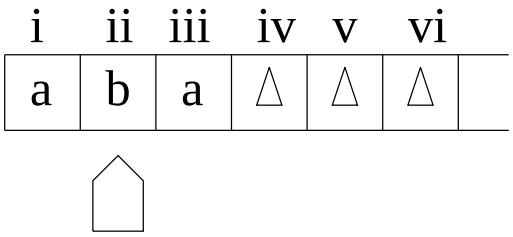
  $$\begin{matrix} 1 \\ \underline{a}ba \end{matrix}$$

  The number on top denotes the state we are currently in. The things below represent the current contents of the tape, with the letter about to be read underlined.

- After reading in a in state 1, the TM then takes the top arc from state 1 to state 2, and so it prints a into the contents of cell i and the Tape Head moves to the right to cell ii. We record this action by writing

  $$\begin{matrix} 1 \\ \underline{a}ba \end{matrix} \longrightarrow \begin{matrix} 2 \\ a\underline{b}a \end{matrix}$$

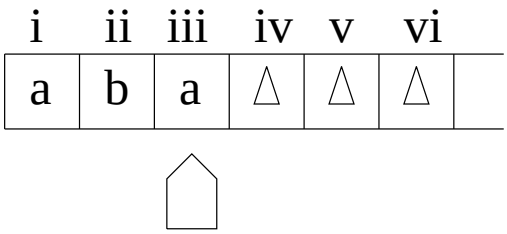
The tape now looks like



- Now we are in state 2, and the Tape Head is pointing to cell ii. Since cell ii contains b, we will take the arc from state 2 to state 3, print b in cell ii, and move the Tape Head to the right to cell iii. We record this action by writing

$$\underset{\underline{a}ba}{1} \longrightarrow \underset{a\underline{b}a}{2} \longrightarrow \underset{ab\underline{a}}{3}$$

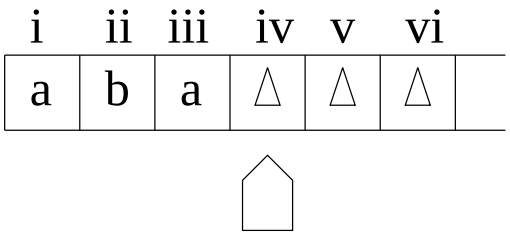
The tape now looks like



- Now we are in state 3, and the Tape Head is pointing to cell iii. Since cell iii contains a, we will take the arc labeled (a, a, R) from state 3 back to state 3, print a in cell iii, and move the Tape Head to the right to cell iv, which contains a blank Δ. We record this action by writing

$$\underset{\underline{a}ba}{1} \longrightarrow \underset{a\underline{b}a}{2} \longrightarrow \underset{ab\underline{a}}{3} \longrightarrow \underset{aba\underline{\Delta}}{3}$$

The tape now looks like



- Now we are in state 3, and the Tape Head is pointing to cell iv. Since cell iv contains Δ, we will take the arc labeled (Δ, Δ, R) from state 3 to state HALT 4, print Δ in cell iv, and move the Tape Head to the right to cell v, which contains a blank Δ. We record this action by writing

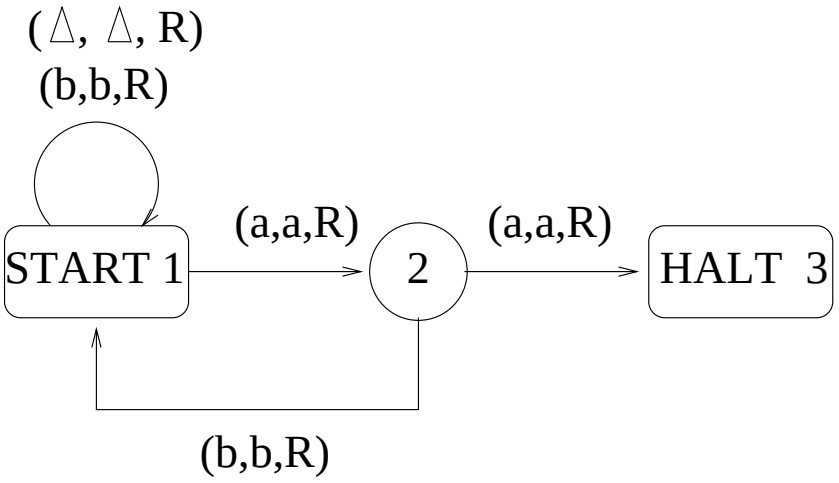
$$\begin{matrix} 1 \\ \underline{a}ba \end{matrix} \longrightarrow \begin{matrix} 2 \\ a\underline{b}a \end{matrix} \longrightarrow \begin{matrix} 3 \\ ab\underline{a} \end{matrix} \longrightarrow \begin{matrix} 3 \\ aba\underline{\Delta} \end{matrix} \longrightarrow \text{HALT}$$

  Since we reached a HALT state, the string on the input tape is accepted.

- Note that if an input string has a as its second letter, then the TM crashes, and so the string is not accepted.

- This TM accepts the language of all strings over the alphabet $\Sigma = \{a, b\}$ whose second letter is b.

Example: Consider the following TM with input alphabet $\Sigma = \{a, b\}$ and tape alphabet $\Gamma = \{a, b\}$:



- Consider processing the word *baab* on the TM
  - Note that the first cell on the TAPE contains b, and so upon reading this, the TM writes b in cell i, moves the tape head to the right to cell ii, and then the TM loops back to state 1,
  - The second cell on the TAPE contains a, and so upon reading this, the TM moves to state 2, writes a in cell ii, and moves the tape head to the right to cell iii.
  - The third cell on the TAPE contains a, and so upon reading this, the TM writes a in cell iii, moves the tape head to the right to cell iv, and moves to state 3, which is a HALT state
  - The TM now halts, and so the string is accepted. Note that the input tape still has a letter b that has not been read.
- Consider processing on the TM the word *bba*.
  - Note that each of the first two b's results in the TM looping back to state 1 and moving the tape head to the right one cell.
  - The third letter a makes the TM go to state 2 and moves the tape head to the right one cell.

  - The fourth cell of the TAPE has a blank, and so the TM then crashes. Thus, *bba* is not accepted.

- Consider processing on the TM the word *bab*.

  - Note that the first letter *b* results in the TM looping back to state 1 and moving the tape head to the right one cell.
  - The tape head then reads the *a* in the second cell, which causes the TM to move to state 2 and moves the tape head to the right one cell.
  - The tape head then reads the *b* in the third cell, which causes the TM to move back to state 1 and moves the tape head to the right one cell.
  - The fourth cell of the TAPE has a blank, and so the TM returns to state 1, and the tape head moves one cell to the right.
  - All of the other cells on the TAPE are blank, and so the TM will keep looping back to state 1 forever.
  - Since the TM never reaches a HALT state, the string *bab* is not accepted.

- In general, we can divide the set of all possible strings into three sets:

  1. Strings that contain the substring *aa*, which are accepted by the TM since the TM will reach a HALT state.
  2. Strings that do not contain substring *aa* and that end in *a*. For these strings, the TM crashes, and so they are not accepted.
  3. Strings that do not contain substring *aa* and that do not end in *a*. For these strings, the TM loops forever, and so they are not accepted.
     Note: The videotaped lecture contains an error about this point.
     - Let S_1 be the set of strings that do not contain the substring *aa* and that do not end in *a*.
     - Let S_2 be the set of strings that do not contain the substring *aa* and that end in *b*.
     - In the videotaped lecture, I said that S_2 is the set of strings for which the TM loops forever, but actually, S_1 is the set of strings for which the TM loops forever.

  - Note that $S_1 \neq S_2$ since $\Lambda \in S_1$ but $\Lambda \notin S_2$.

- This TM accepts the language having regular expression $(\mathbf{a}+\mathbf{b})^*\mathbf{aa}(\mathbf{a}+\mathbf{b})^*$.

Definition: Every Turing machine T over the alphabet Σ divides the set of input strings into three classes:

1. **ACCEPT**(T) is the set of all strings $w \in \Sigma^*$ such that if the Tape initially contains w and T is then run, T ends in a HALT state. This is the *language accepted* by T.

2. **REJECT**(T) is the set of all strings $w \in \Sigma^*$ such that if the Tape initially contains w and T is then run, T crashes (by either moving left from cell i or by being in a state that has no exit edge labeled with the letter that the Tape Head is currently pointing to).

3. **LOOP**(T) is the set of all strings $w \in \Sigma^*$ such that if the Tape initially contains w and T is then run, T loops forever.

So for our last example,

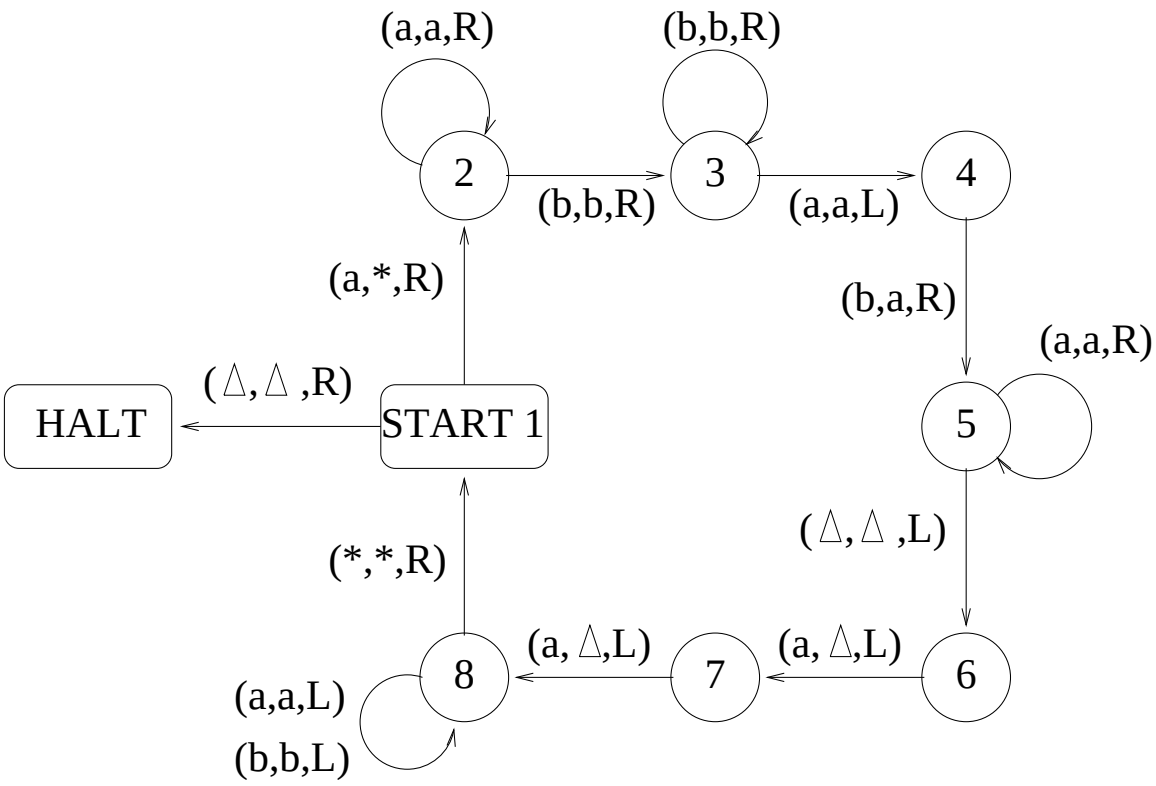
- **ACCEPT**(T) = set of strings generated by the regular expression $(\mathbf{a}+\mathbf{b})^*\mathbf{aa}(\mathbf{a}+\mathbf{b})^*$.

- **REJECT**(T) = strings in Σ^* that do not contain the substring aa and that end in a, where $\Sigma = \{a, b\}$.

- **LOOP**(T) = strings in Σ^* that do not contain the substring aa and that do not end in a, where $\Sigma = \{a, b\}$.

Example: Below is a TM for the language $L = \{a^n b^n a^n : n = 0, 1, 2, \ldots\}$:

$\Sigma = \{a, b\}$, $\Gamma = \{a, b, *\}$



We now examine why this TM accepts the language L defined above.

Step 1. Presume that we are in state 1, and we are reading the first letter of what remains on the input.

- So initially, we are reading the first letter on the input tape, but as we progress, we may find ourselves back in state 1 reading the first letter of what remains on the tape.
- If we read a blank, then we go to HALT.
- If what we read is a, then change it to $*$, and move the tape head to the right.
- If we read anything else, we crash.

Step 2. In state 2, we skip over the rest of the a's in the initial clump of a's, looking for the first b.

- When we find the first b, we move to state 3.
- As long as we keep reading b's, we keep returning to state 3.
- When we find the first a in the second clump of a's, we then go to state 4, and move the tape head to the left back to the last b in the clump of b's.
- We then change the last b to an a, move the tape head to the right, and go to state 5. So now the number of b's has reduced by 1, and the number of a's in the second clump of a's has increased by 1.
- We did all of these last few steps to find the last b in the clump of b's.

Step 3. Now we are in state 5 with the tape head pointing to the first a in the second clump of a's, and we want to find the last a in the second clump of a's.

- Each a that we now read makes us return back to state 5, and move the tape head to the right.
- If we read b, then the machine crashes.
- When we finally encounter Δ, then the tape contains no more characters to the right, and the TM goes to state 6.
- We then move to state 7 and then to state 8, and change the last two a's to Δ's.
- Thus, the number of a's in the second clump has decreased by 2. But in Step 2, we increased the number of a's in the second clump by 1, and so now the number of a's in the second clump has decreased by 1 since we started in Step 1.
- Recall that Step 2, we also reduced the number of b's by 1.
- Recall that in Step 1, we changed the first a in the first clump of a's to $*$.

Step 4. Now we are in state 8 with the tape head pointing to the last a currently in the second clump of a's, and we want to get back to the first a that is currently in the first clump of a's.

- In state 8, as long as we keep reading a's and b's, we move the tape head to the left, and return back to state 8.
- Recall that in Step 1, we changed the first a in the first clump of a's into $*$.
- So when the tape head finally reaches the rightmost $*$ by moving left, the TM goes to state 1, and we move the tape head to the right, and we repeat our 4 steps.

19.2 Stupid TM Tricks

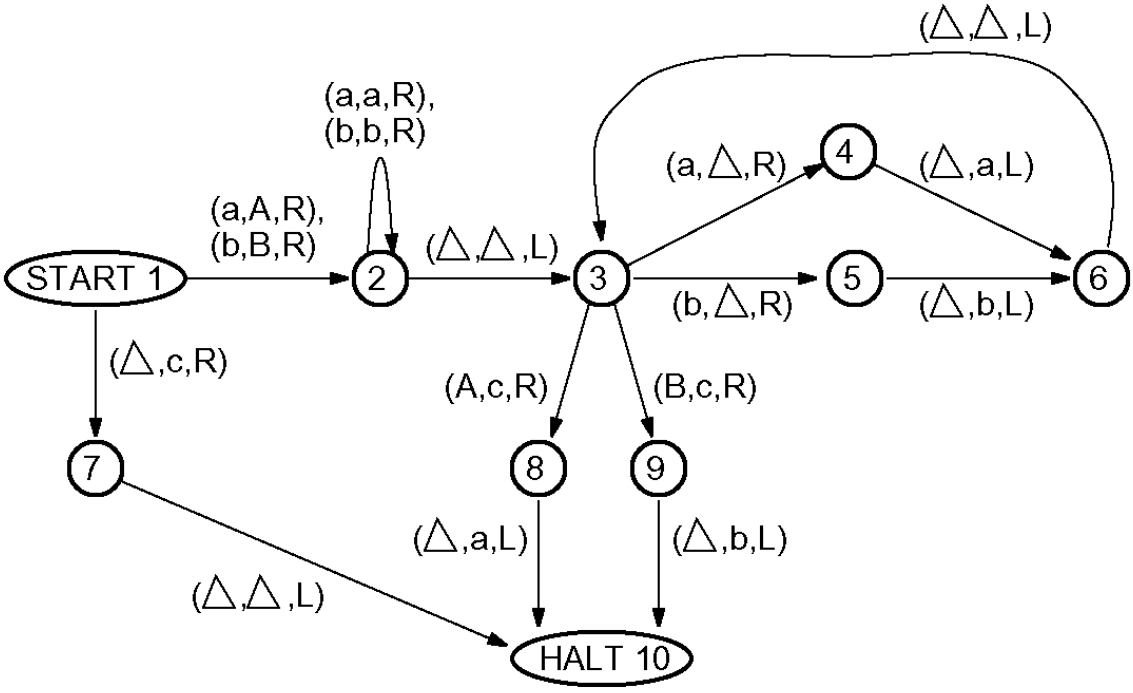
There are several tricks that one can do with Turing Machines:

1. Storing information in states; e.g., check if first letter of input string appears later in the string.
2. Multiple tracks on tape.
3. Checking off symbols on tape.
4. Inserting a character anywhere onto the input tape and shifting over the rest of the contents of the tape.

   Example: TM to

   - insert c at the beginning of Tape,
   - shift entire original contents of Tape one cell to the right, and
   - finish with Tape Head pointing at cell i.

   $\Sigma = \{a, b\}$, $\Gamma = \{a, b, A, B, c\}$.



The language of this TM is all strings over $\Sigma = \{a, b\}$ since starting the TM with any string in Σ^* loaded on the Tape and running the TM will lead to the Halt state.

5. Can design TM to delete contents of specific cell and shift contents of all cells to the right one cell to the left.

6. Subroutines.

7. Can convert any FA into a TM.

8. Can convert any PDA into a TM.

Theorem 46 *If L is a regular language, then there exists a TM for L.*