1 Turing Machines

Definition Turing Machine is a finite state machine with an infinite and unrestricted memory (denoted as "tape").

Differences from finite automata:

- A TM can both read and write on the tape.
- The tape head can move both left and right.
- The tape is infinite.
- The accept/reject states take immediate effect.

Note: A TM may not stop on every input.

Church-Turing Thesis: Everything "computable" is computable by a TM.

One transition of a TM:

- Change state of M.
- Write a symbol on the tape.
- Move the head one cell to the left or right.

Definition A TM is an 8-tuple $(Q, \Sigma, \Gamma, B, \delta, q_0, q_{accept}, q_{reject})$ where:

- Q is a finite set of states.
- Σ is the input alphabet.
- $\Gamma \supseteq \Sigma$ is the tape alphabet.
- $B \in \Gamma - \Sigma$ is a special *blank symbol* (also denoted by $\sqcup$).
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the transition function.
- $q_0 \in Q$ is the start state.
- $q_{accept} \in Q$ is the accepting state.
- $q_{reject} \in Q$ is the rejecting state.

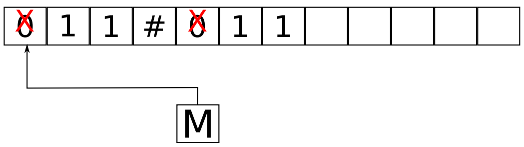
Definition If the TM is in state q, and the tape head is over symbol a, then the transition:

$$\delta(q, a) = (r, b, L)$$

tells the TM to move to state r, replace a with b, and move the head left.

Example A TM that accepts

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



$M_1 =$ on input string w

1. Zig-zag across the tape to corresponding positions on either side of the $\#$ symbol to check whether these positions contain the same symbol. If they do not, or if no $\#$ is found, *reject*. Cross off symbols as they are checked to keep track of which symbols correspond.
2. 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*.

Consider $w = 011000\#011000$

$\rightarrow$0	1	1	0	0	0	$\#$	0	1	1	0	0	0	$\sqcup$	...
x	$\rightarrow$1	1	0	0	0	$\#$	0	1	1	0	0	0	$\sqcup$	...
x	1	1	0	0	0	$\#$	$\rightarrow$x	1	1	0	0	0	$\sqcup$	...
$\rightarrow$x	1	1	0	0	0	$\#$	x	1	1	0	0	0	$\sqcup$	...
x	$\rightarrow$x	1	0	0	0	$\#$	x	1	1	0	0	0	$\sqcup$	...
x	x	x	x	x	x	$\#$	x	x	x	x	x	x	$\rightarrow\sqcup$	...
													accept	...

Configuration of a TM is specified by three things:

- state of the machine.
- contents of the tape.
- position of the tape head.

It is written as "$w_1 q w_2$", where $w_1, w_2 \in \Gamma^*$ and $q \in Q$, to denote:

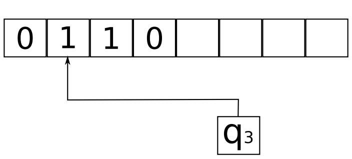
- the machine is in state q
- the tape has string $w_1 w_2$
- the tape head is on the leftmost symbol of w_2

Definition Configuration C_1 **yields** C_2 if the TM can go from C_1 to C_2.

- $C_1 = uaq_ibv$ yields $C_2 = uq_jacv$ if $\delta(q_i, b) = (q_j, c, L)$
- $C_1 = uaq_ibv$ yields $C_2 = uacq_jv$ if $\delta(q_i, b) = (q_j, c, R)$

Definition Start configuration of a TM for an input string w is denoted as $q_0 w$.

Example



is denoted as $0q_3110$

1.1 Turing-recognizable and Decidable Languages

Three possible outcomes for an input w:

- accept
- reject
- infinite loop

Accept and *Reject* states are *halting* states.

Definition A TM *accepts* w through configurations $C_1 C_{2k}$ if:

- C_1 is the starting configuration.
- C_i yields C_{i+1}, $1 \leq i \leq k-1$
- C_k is the accepting configuration.

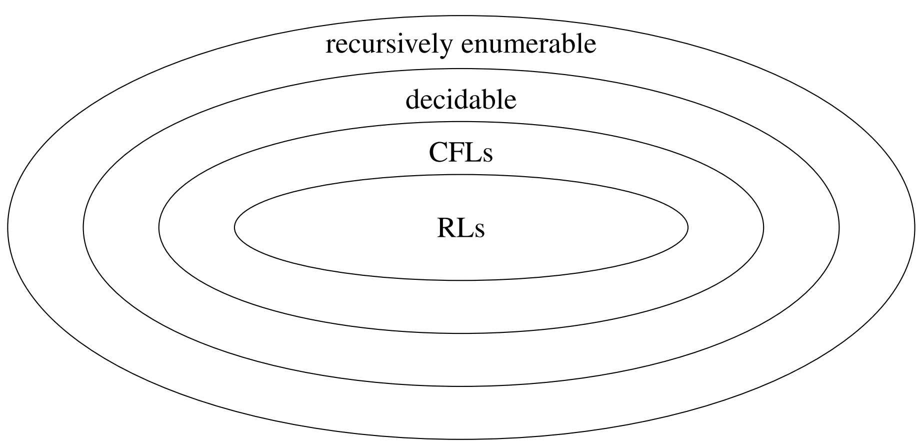
Definition The language of a TM M, denoted by $L(M)$, is the set of all string accepted by M.

Definition Languages that are recognized/accepted by some TM are called *Turing-recognizable*, or *recursively enumerable* languages.

Definition A TM that halts on every input is called a *decider*.

Definition The languages that are accepted by decider TMs are called *Turing-decidable*, or *decidable*, or *recursive* languages.

Definition



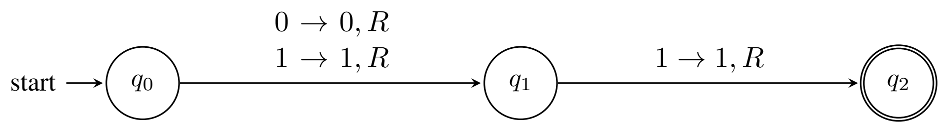
Example

- $L(M_1) = \{w \mid \textit{second character of } w \textit{ is } 1\}$

  Informal description:

  1. Read the first letter, move to the right.
  2. Read the second letter. If it is a 1, accept. Else, reject.

  Formal description (transition diagram):

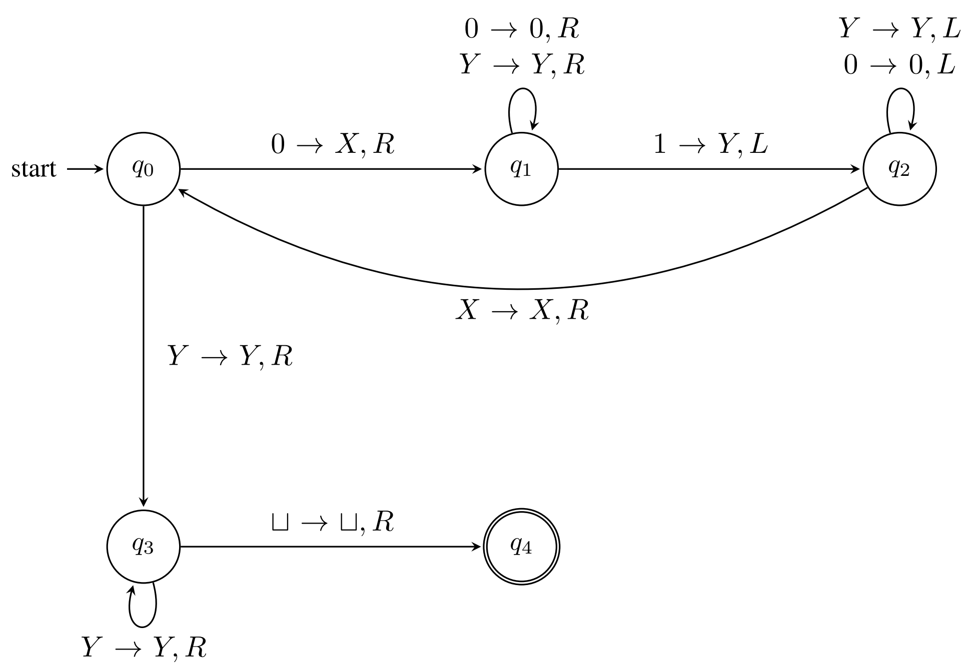


- $L(M_2) = \{0^n 1^n \mid n \geq 1\}$

  Informal description:

  1. Check the leftmost 0, replace it by X.
  2. Move to the right, skip 0s and Ys. If a 1 is found, replace it by Y.
  3. Move to the left until you find an X. If an X is found, move one cell to the right.
  4. If that symbol is 0, repeat the above procedure. If that symbol is Y, move to the right, to the end of Ys. If you find a $\sqcup$, accept.

  Formal description (transition diagram):



- $L(M_3) = \{a^i b^j c^k \mid k = i \times j\}$

  Informal description:

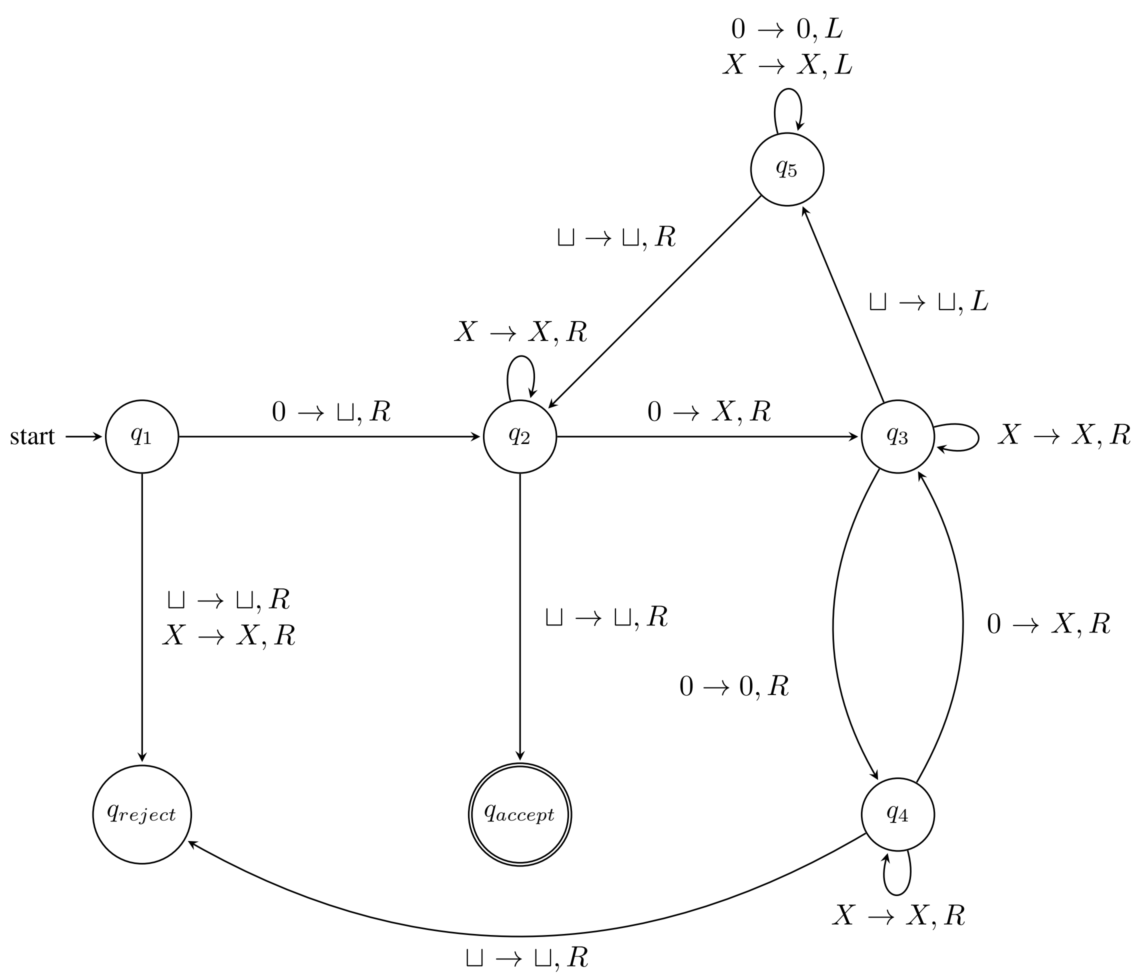
  1. Find the leftmost a, replace it with X.
  2. For every b:
     (a) replace the b with Y.
     (b) replace one c with Z.
  3. Move to the left to the leftmost a, replace each Y with a b.
  4. Repeat the above procedure for each remaining a, replace a with X.
  5. When no a can be found, check the end of the string. If no c is left, accept.

- $L(M_4) = \{0^{2^n} \mid n \geq 0\}$

  Informal description:

  1. Sweep left to right across 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 a single 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.

  Formal description (transition diagram):



Sample Run: $w = 0000$

$q_1 0000$	$\sqcup q_2 000$	$\sqcup X q_3 00$	$\sqcup X 0 q_4 0$	$\sqcup X 0 X q_3 \sqcup$	$\sqcup X 0 q_5 X \sqcup$	$\sqcup X q_5 0 X \sqcup$
$\sqcup q_5 X 0 X \sqcup$	$q_5 \sqcup X 0 X \sqcup$	$\sqcup q_2 X 0 X \sqcup$	$\sqcup X q_2 0 X \sqcup$	$\sqcup X X q_3 X \sqcup$	$\sqcup X X X q_3 \sqcup$	$\sqcup X X q_5 X \sqcup$
$\sqcup X q_5 X X \sqcup$	$\sqcup q_5 X X X \sqcup$	$q_5 \sqcup X X X \sqcup$	$\sqcup q_2 X X X \sqcup$	$\sqcup X q_2 X X \sqcup$	$\sqcup X X q_2 X \sqcup$	$\sqcup X X X q_2 \sqcup$
						$\sqcup X X X \sqcup q_{accept}$

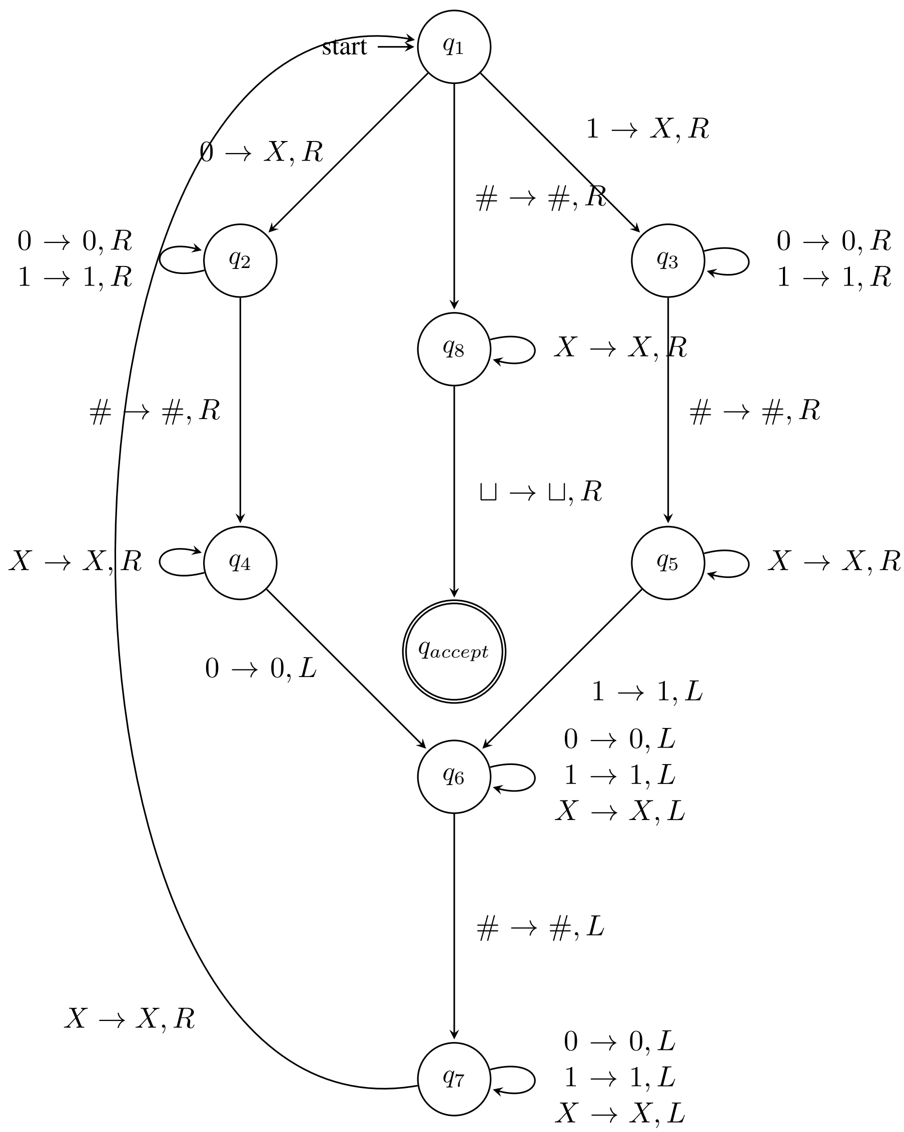
$L(M_5) = \{w\#w \mid w \in \{0,1\}^*\}$

Informal description:

1. Zig-zag across the tape to corresponding positions on either side of the $\#$ symbol to check whether these positions contain the same symbol. If they do not, or if no $\#$ is found, *reject*. Cross off symbols as they are checked to keep track of which symbols correspond.

2. 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*.

Formal description (transition diagram):



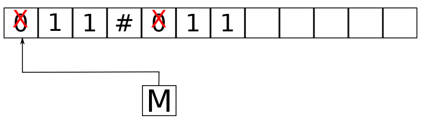
1.2 Convenient techniques in TM programming

We can assume the following when we talk about TMs:

1. Control unit (i.e. state machine) has a finite memory.
2. The tape has multiple tracks.
3. The TM can have another TM as a subroutine.

1.2.1 Control unit with a finite memory

Example A TM that accepts $\{w\#w \mid w \in \{0,1\}^*\}$



Here we remember we read 0 so we will process the next element of the second part if it is 0.

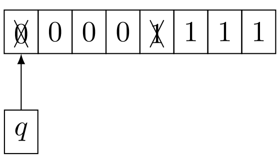
Note: This argument is valid only when the variable to be remembered has a finite domain. E.g. to accept $\{0^n1^n \mid n \geq 0\}$, we cannot say "read the 0s and remember n".

1.2.2 Multi-track tape

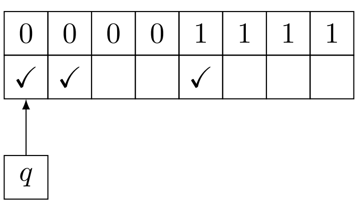
We can assume that the tape has multiple tracks.

E.g. a TM that accepts $\{0^n1^n \mid n \geq 0\}$

Instead of this:



We can have:



If track-1 has alphabet Γ_1, and track-2 has alphabet Γ_2, a TM with a single-track tape with alphabet $\Gamma = \Gamma_1 \times \Gamma_2$ can do the same thing. E.g.

$$(0, \checkmark) = X$$

$$(1, \checkmark) = Y$$

$$(0, \sqcup) = 0$$

$$(1, \sqcup) = 1$$

1.2.3 Subroutines

A TM can use another TM as a subroutine. E.g. a TM that does multiplication can call another TM that does addition as a subroutine.

1.3 Variations of TMs

1. Multitape TM
2. Two-way infinite tape TM
3. Non-deterministic TM
4. ...

can all be emulated by a simple 1-tape, deterministic TM.

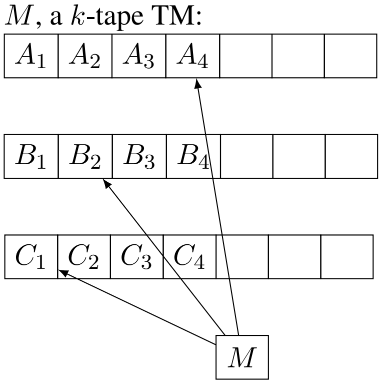
1.3.1 Multitape TM

- k tapes, k independent tape heads.
- At every transition:
  - a symbol is read from each tape.
  - state is changed.
  - on each tape, a new symbol is written, and the head is moved to the left or right.

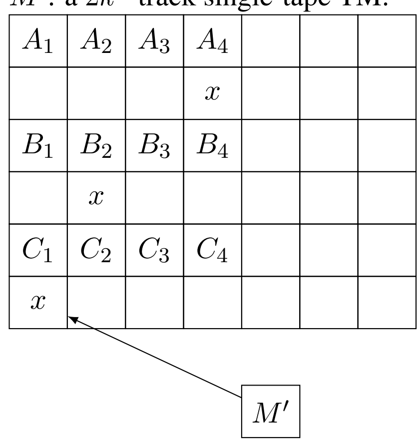
Theorem 1.1 *Every multitape TM has an equivalent single-tape TM.*

Proof by construction.

M, a k-tape TM:



M': a $2k-$track single-tape TM:



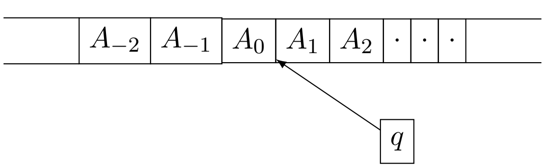
We write x a the position of our virtual tape. Then we can find where we left off on our tape easily.

M' emulates M as follows:

- The tape head makes k passes, once for each tape, locating the tape head emulator, and reading its symbol.
- Updates the state accordingly. Accepts if it is an accept state.
- Makes another k passes over the tape, locating each head emulator, updating its symbol, and moving the head emulator to the left or right.

∎

1.3.2 Two-way infinite tape TM



It can easily be emulated by a 2-tape TM.

1.3.3 Non-deterministic TM

- Defined as expected:
$$\delta : Q \times \Gamma \rightarrow 2^Q \times \Gamma \times \{L, R\}$$

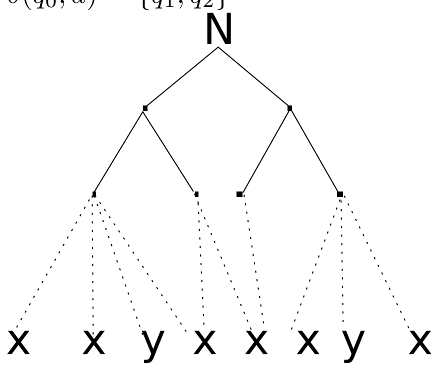
- The computation of an NTM is a tree where different branches are different alternative paths.

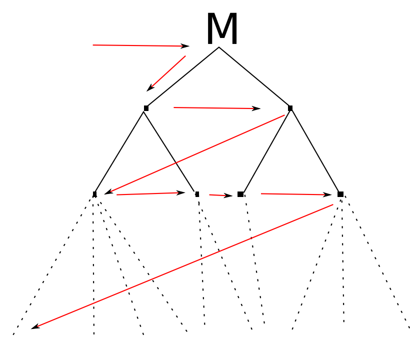
- It some branch leads to an accept state, the input is accepted.

Theorem 1.2 *Every NTM has some equivalent DTM.*

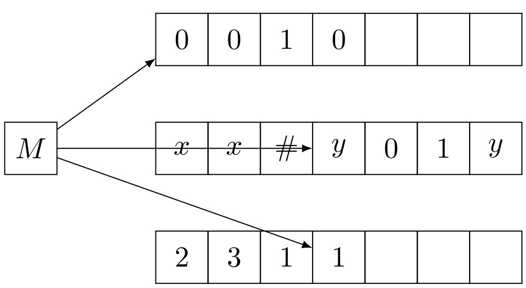
Proof by construction.

$\delta(q_0, a) = \{q_1, q_2\}$





For an NTM N, we can construct a 3-tape TM M that emulates N and searches tree by **BFS** for an accepting configuration. Note that DFS may continue forever and never find an accepting state that is on a different branch.



- T_1: "input tape" to preserve the original input
- T_2: "evaluation tape" emulates N on the input according to the branches specified by the address tape.
- T_3: "address tape" to keep track of the BFS. 231 on the address tape means: go to the 2^{nd} child of the root, then to its 3^{rd} child, then to its 1^{st}.

Steps:

1. Copy the input to T_1. Set T_2 and T_3 to be empty.
2. Copy T_1 to T_2, set $T_3 = \epsilon$.
3. Evaluate T_2, before each step, consult T_3 what to do (find the configurations). If $\sqcup$ is reached in T_3, abort, go to step 4.
4. Replace the string in T_3 with the next string (BFS steps), then go to step 2.

1.4 Other models equivalent to TMs

- Multi-stack PDA
- Counter machines with multiple counters