ToC - Turing Machine Variants

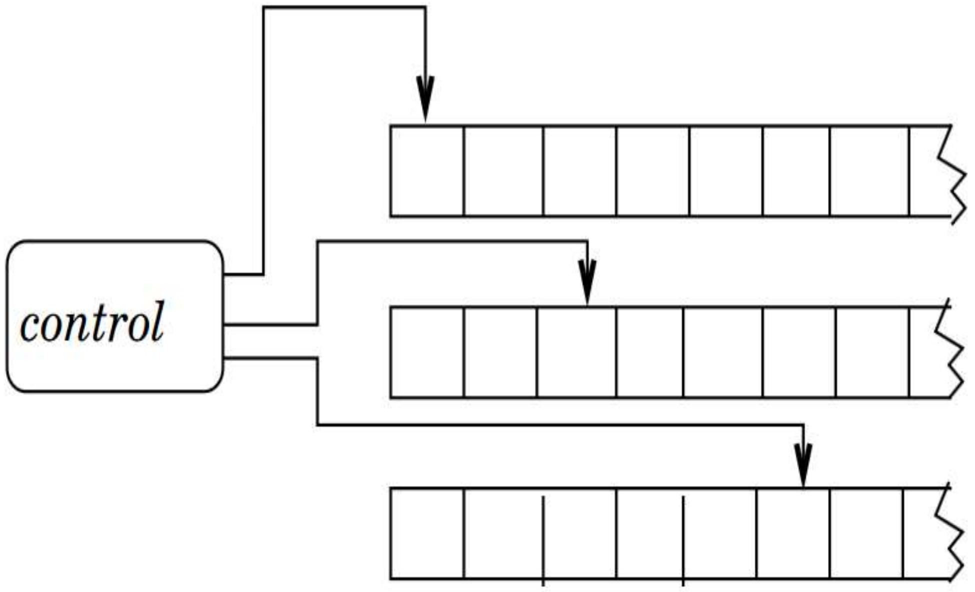
Course Instructor: Jibi Abraham

TM Variants

- A standard TM consists of a single tape with a single track, closed at the left end and infinite (unbounded) at the right end
- There are variants to give more flexibility without affecting the computing power
  - Multiple Tracks (Multi-Track) TM
  - Two-way Infinite Tape TM
  - Multiple Tapes (Multi-Tape) TM
  - Nondeterministic TM
  - TM with output (Enumerator)
- Languages accepted all these Turing Machines are the same recursively enumerable languages

Multi – Tape TM

- A multi-Tape TM consists of k tapes and k independent tape heads
- States and alphabets of a Multi-Tape TM are the same as in a std TM
- TM reads the tapes simultaneously, but has only one state
- This is depicted by connecting each of the independent tape heads to a single control indicating the current state

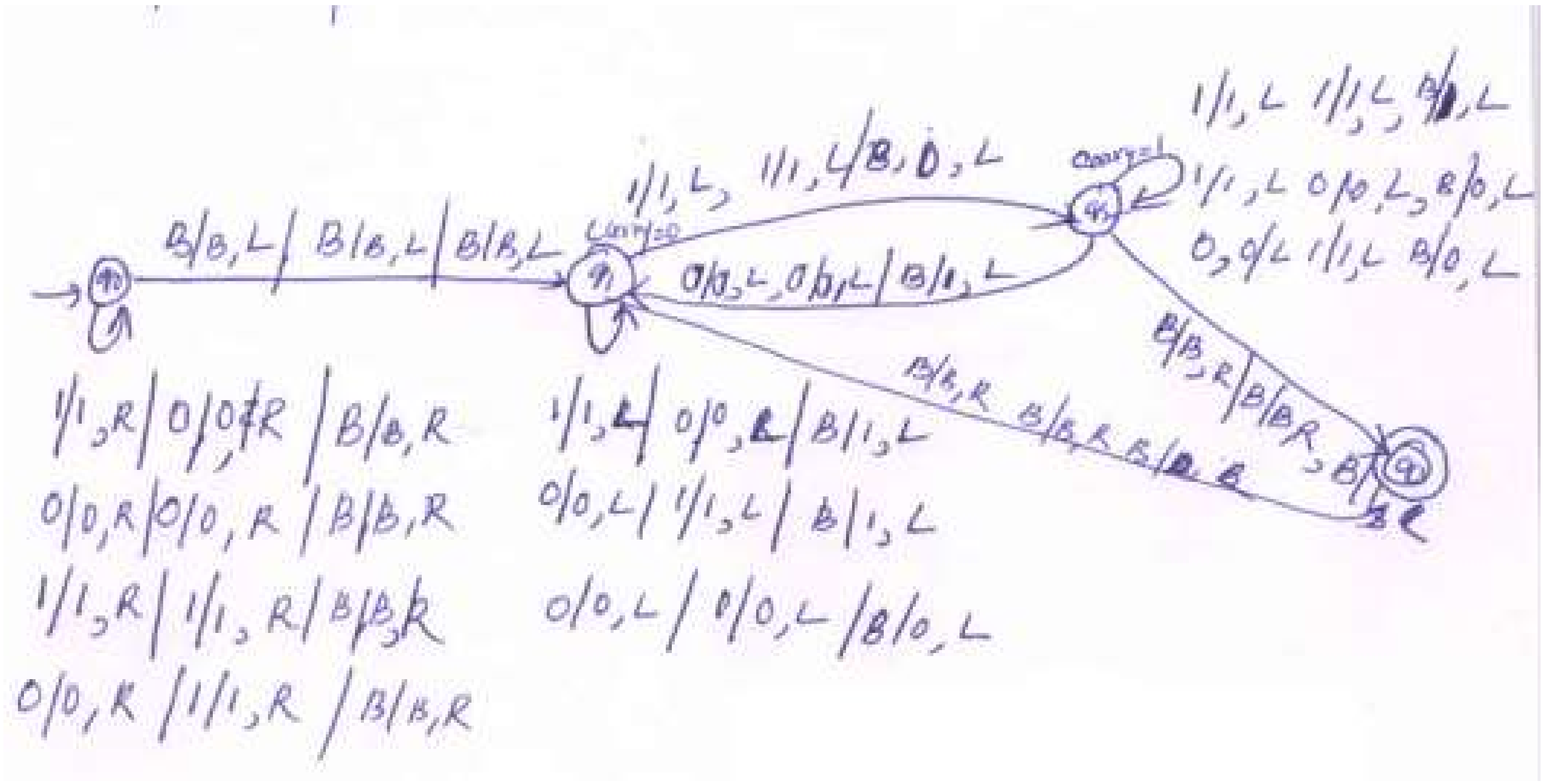


Multi – Tape TM Contd..

- A transition is determined by the state and symbols scanned by each of the tape head
- A transition in a multi-tape TM may
  1. Change the state
  2. Write a symbol on each of the tapes
  3. Independently reposition each of the tape heads
- Repositioning consists of moving the tape head
  - One cell to the left (L) or
  - One cell to the right (R) or
  - Leaving it at its current position (S)
- Input to a multi-tape TM is placed in the std position on tape 1
- All the other tapes are assumed to be blank initially
- Tape heads originally scan the leftmost position of each tape
- Any tape head attempting to move to the left of the boundary of its tape terminates the computation abnormally

Multi-Tape Example

- 1st tape – m in unary
- 2nd tape – n in unary
- 3rd tape m+n in unary

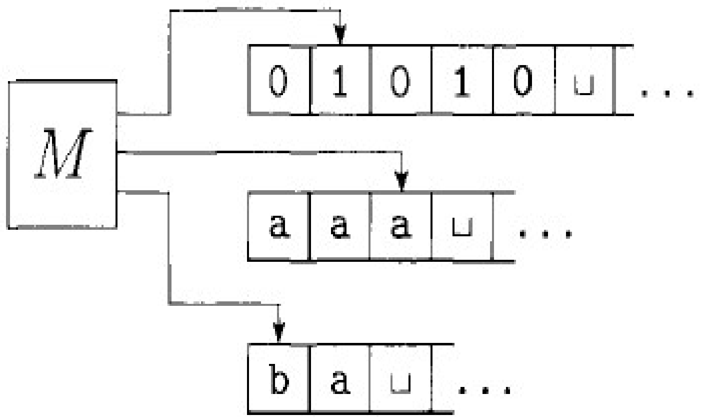


Equivalence with Std TM

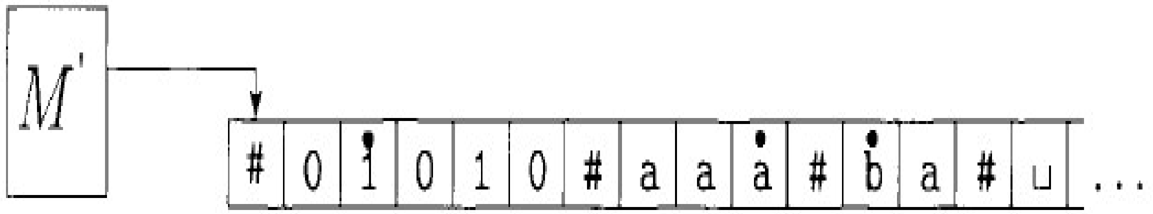
- Theorem: Every Multi-ape TM has an equivalent one-tape TM
- Proof:
- Let the Multi-tape TM M be having k tapes
- The one-tape TM M′ simulates the effect of k tapes by storing their information on a single tape
- M′ uses a delimiter symbol, # to separate the contents of different tapes
- If the input string is a_0, a_1, … , a_n, then M′ represents all the k tapes as # a_1, … , a_n#B#B … #
- In addition, M′ has to take care of different head positions of M in each tape

Multi – Tape Proof Contd..

- M′ simulates the multiple virtual tape heads by marking a dot on top of the corresponding tape symbol on the tape cell
- For example, for the configuration of M



- The equivalent tape configuration of M′

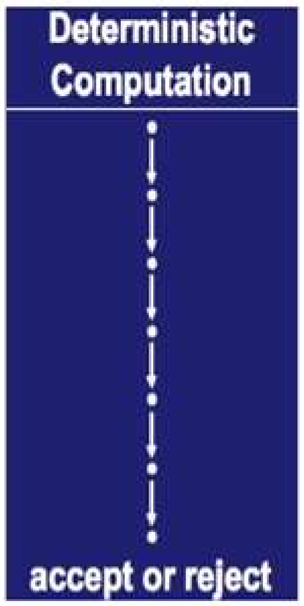


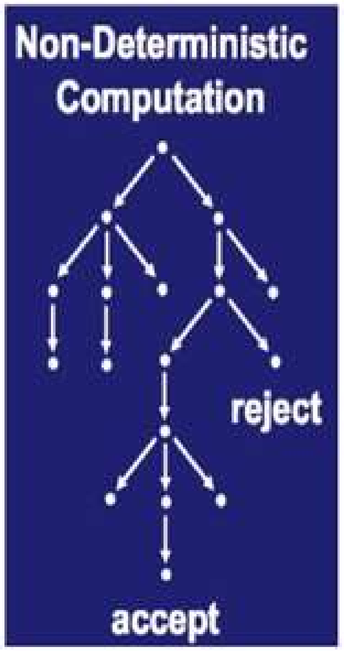
Multi – Tape Proof Contd..

- When M makes a transition, M′ makes two passes through the tape contents
- In the first pass from left most # symbol to (k+1)st # symbol, it finds the positions of the tape heads
- In the 2nd pass, it updates the tape cells under the tape head as defined by the transition function of M
- Some times M reaches at the end of a tape, reads a Blank symbol, may replace it with another tape symbol as defined by δ, moves to the right and continues similar processing
- Similar action is simulated by M′ when a virtual tape head reads a # symbol
  - Then a Blank cell will be created by moving all the tape symbols one position from this current # position till the rightmost #
  - Then this blank cell will be replaced with tape symbol as per δ.

Non-deterministic TM

- A deterministic TM after reading an input tape symbol at a particular state, makes one definite transition and reading after all input, arrive at an accept or reject





- A Non-deterministic TM is like Non-deterministic FA
- Being at any state and for the tape symbol it is reading, it can take any transition selecting from a set of specified transitions
- One of the branches may be arriving at accept while others arrive at the reject state

Non-deterministic TM Cond..

- Transitions in a non-deterministic TM:
  - δ: Q x Γ to the **power set** of Q x Γ x {L, R}
- For example: L = {ww: w ϵ $\{a, b\}^*$}
- Given a string x, a non-deterministic TM that accepts this language L would first guess the midpoint of x, which is the place where the second half of x starts
- Then it would compare the first half of x with the second half by comparing the ith symbol of the first half with the ith symbol of the second half for i = 1, 2, ...
- A Deterministic TM, on the other hand, cannot guess the midpoint of the string x
- It must find the midpoint by for example pairing off symbols from either end of x

Non-deterministic TM Cond..

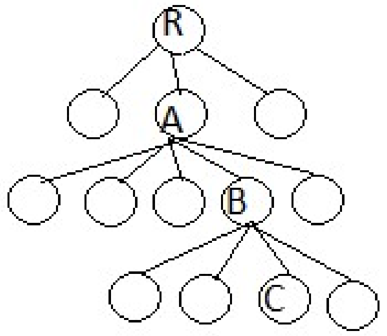
- Computations of a Non-deterministic TM on any input string are represented as a tree
- Each branch of the tree is a branch of non-determinism
- Each node is a configuration of the TM at a particular instance, root being the start configuration
- A breadth-first traversal has to be conducted to search for a successful path for acceptance of the input string
- One cannot proceed by depth-first search as the traversal may lead to an infinite branch while missing the acceptance configurations on some other branches

Equivalence with Std TM

- Theorem: Every non-deterministic TM can be simulated by a deterministic TM
- Proof: Let M be the non-deterministic TM
- It can be simulated by using a deterministic TM M′ with multi-tape
- M′ has 3 tapes known by input tape, simulation tape and address tape
- Initially the input tape contains the input string to be recognized and this tape’s contents never get altered
- Simulation tape contains a configuration of M at some branch of its non-deterministic computation
- Address tape keeps track of the location (node position) of M′ in the computation tree of M

Non-deterministic TM Proof Cond..

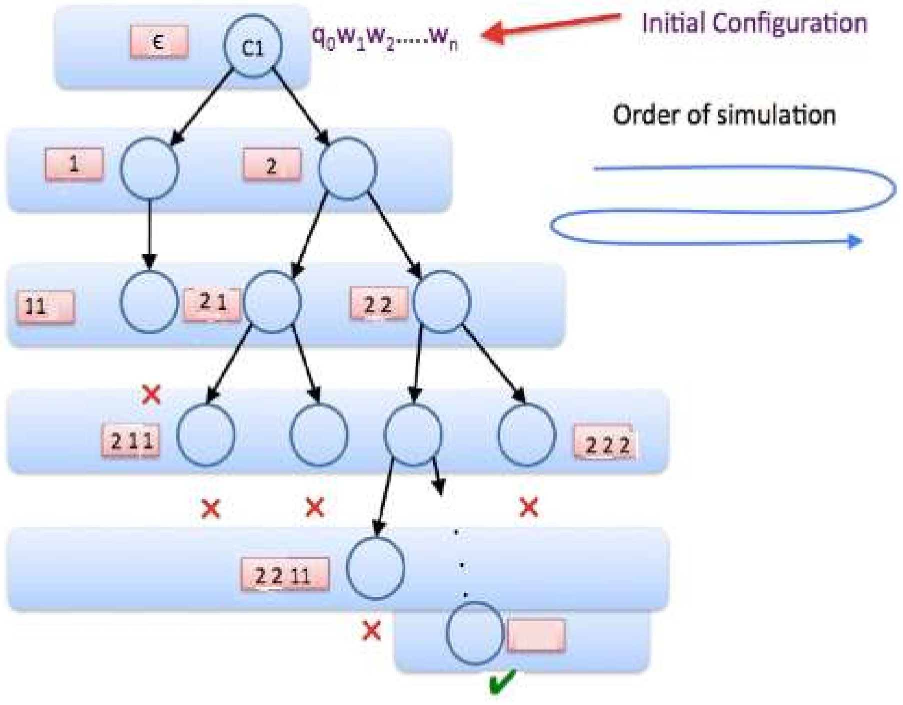
- Suppose every node in the Non-deterministic computation tree has at most b children
- Let every node in the tree has an address which is a string over the alphabet Γ_b = {1, 2, .. b}
- The root node is having address of ϵ



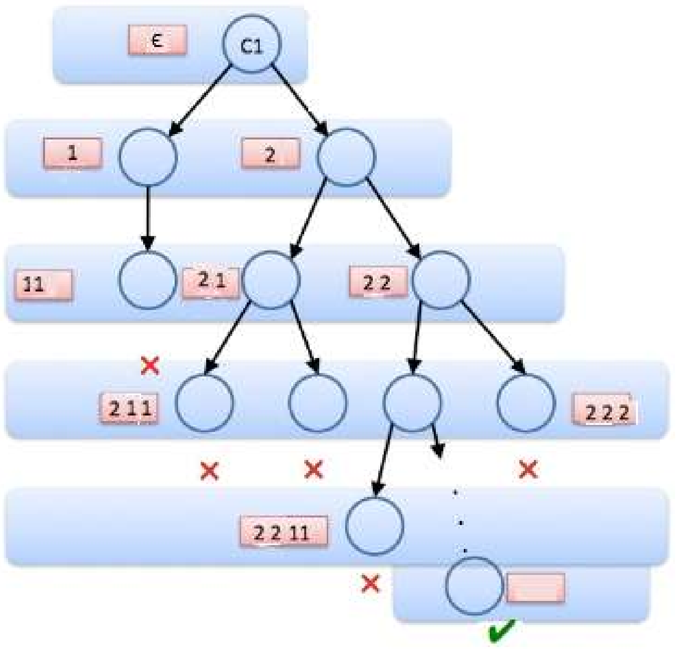
- To obtain a node with address 243 for example, start at the root node, move to 2nd child of root (with name A), then move to 4th child of A (with name B) and then move to 3rd child of B (with name C)
- When the BFS reaches at root, the address tape stores the node location as ϵ
- When the BFS reaches at A, B, C, the address tape stores the node location as 2, 24, 243 respectively

Non-deterministic TM Proof Cond..

- A BFS explores all the branches at the same depth before going on to explore any branch in the next depth
- This guarantees that M′ will visit every node in the tree until it encounters an accepting configuration
- BFS checks the nodes in lexicographical ordering as ϵ, 1, 2, … , b, 11, 12, … 1b, 21, 22, … 2b, ….



Non-deterministic TM Proof Cond..



- M′ works on the input string by copying the input string to the 1st tape and keeping the other two tapes empty
- Initial configuration of TM is copied into tape 2 and address of root (ϵ) to tape 3
- On the simulation tape (Tape 2), M′ needs to maintain a queue of configurations of M
- On the address tape (Tape3), a queue of addresses of nodes of the tree in lexicographical ordering are maintained
- For example, while reaching at the level 1, the configuration corresponding to node 1 and node 2 are stored at the simulation tape and address of node 1 and node 2 are stored on the address tape

Non-deterministic TM Proof Cond..

- M′ takes the next configuration from the head of the queue from the simulation tape and the next address form the address tape
- It makes as many copies of the configuration as needed by the number of transitions on M
- On each copy of the configuration, M′ simulates one the non-deterministic move of M and gets a new resulting configuration and new address of the node
- TM will be halted if the resulting configuration is an accepting configuration
  - In that case the address will give the branch of TM’s transitions

Non-deterministic TM Proof Cond..

- If the TM has not halted due to acceptance, place the resulting configuration to the end of the queue of the simulation tape and its address to the end of queue of the address tape
- Abort a branch of simulation if:
  - Address queue is empty
  - Rejecting configuration is encountered
  - Non-deterministic choice is not a valid choice
- Once the present branch is aborted, replace the string on the address tape with the next in the lexicographical ordering
- Continue simulation of this branch of Non-deterministic TM

Enumerator

- Normally, a TM recognizes a language
- An *enumerator* provides a different way of using a TM
- An enumerator can be thought as a TM which generates recursively enumerable languages
- It is like a TM with an attached printer
- TM starts with a blank tape (since it is does not define a language by accepting input strings)
- As the TM runs, it prints strings belonging to the language on the printer
- If the TM does not halt, it may perform printing infinitely

Language of Enumerator

- Theorem: A language is Turing-Recognizable if and only if some enumerator enumerates it
- Proof:
- We must show that enumerator *E* enumerates language *L* iff some TM *M* recognizes *L*
- Part 1: If enumerator *E* enumerates language *L* then there exists TM *M* that recognizes *L*
- Use *E* to define TM *M* as follows: For input *w*
  - examine each strings enumerated by *E*
  - if *w* appears, then accept *w*
- Every string printed by *E* will be accepted by *M*
- Strings not printed by *E* will be rejected by M

Language of Enumerator

- Part 2: If TM *M* recognizes language *L* then there exists an enumerator *E* that enumerates language *L*
- Use *M* to define enumerator *E* as follows:
  - Assume that *E* prints strings s_1, s_2, ... s_i
  - For i in 1, 2, 3 ...
    - Run *M* for *i* steps on strings s_1, s_2, ... s_i
    - If *M* enters an accept state, print it
- If *M* accepts a particular string *s*, eventually it will appear on the list generated by *M*. Every string in *L* will be printed many times.
- Strings not in *L* will not be printed.

?