Turing Machines

Reading: Chapter 8

Turing Machines are…

- Very powerful (abstract) machines that could simulate any modern day computer (although very, very slowly!)

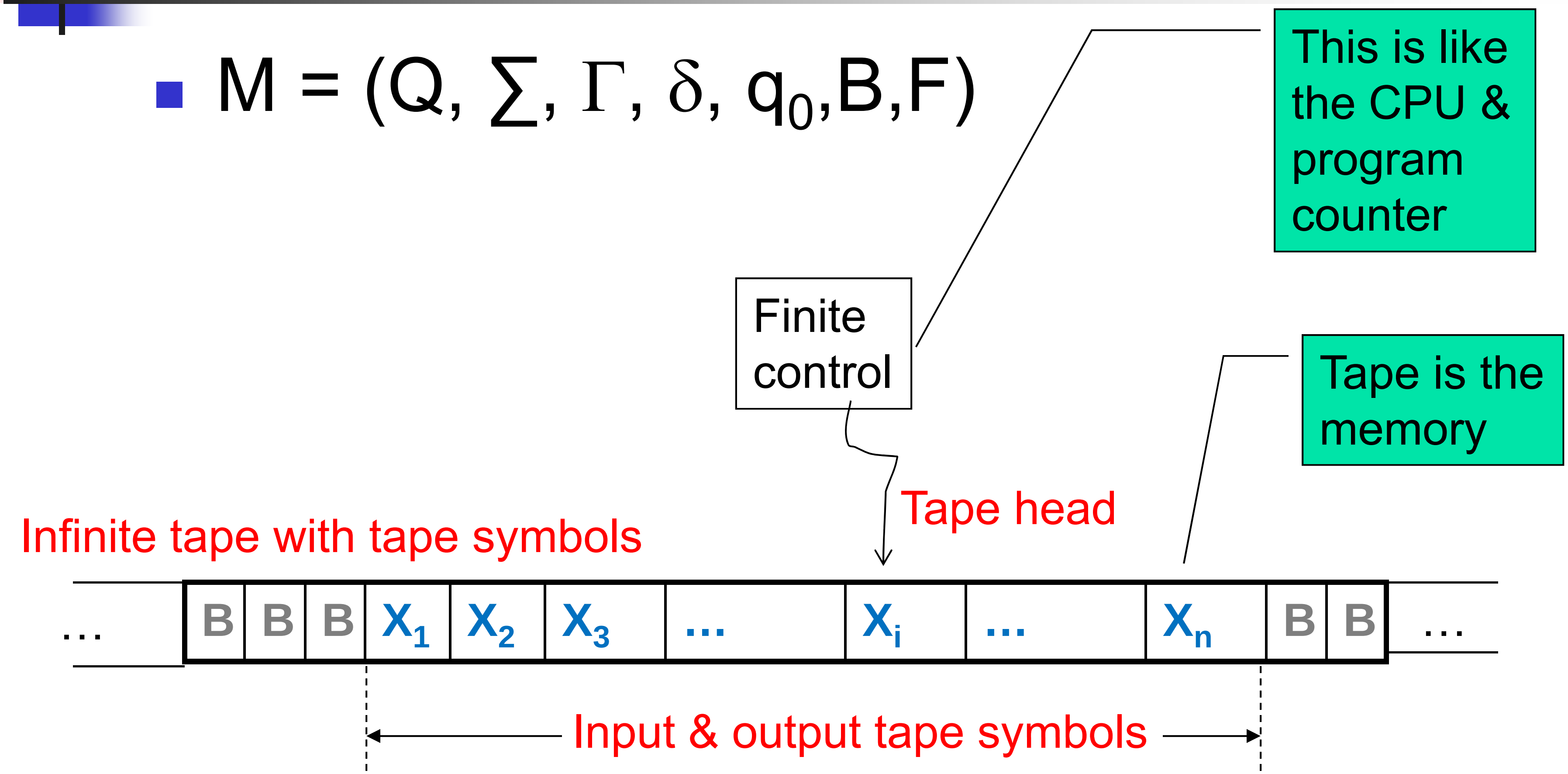
For every input, answer YES or NO

- Why design such a machine?
  - If a problem cannot be "solved" even using a TM, then it implies that the problem is ***undecidable***

- Computability vs. Decidability

A Turing Machine (TM)

- $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$

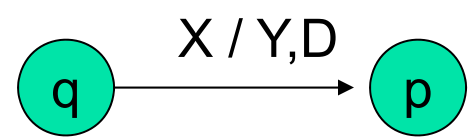


B: blank symbol (special symbol reserved to indicate data boundary)

Transition function

You can also use:
➔ for R
🡐 for L

- One move (denoted by |---) in a TM does the following:
  - $\delta(q,X) = (p,Y,D)$
    - q is the current state
    - X is the current tape symbol pointed by tape head
    - State changes from q to p
    - After the move:
      - X is replaced with symbol Y
      - If D="L", the tape head moves "left" by one position.
        Alternatively, if D="R" the tape head moves "right" by one position.



ID of a TM

- Instantaneous Description or ID :
  - $X_1X_2\ldots X_{i-1}qX_iX_{i+1}\ldots X_n$
    means:
    - q is the current state
    - Tape head is pointing to X_i
    - $X_1X_2\ldots X_{i-1}X_iX_{i+1}\ldots X_n$ are the current tape symbols

- $\delta(q,X_i) = (p,Y,R)$ is same as:
  $X_1\ldots X_{i-1}qX_i\ldots X_n \quad |\text{----} \quad X_1\ldots X_{i-1}YpX_{i+1}\ldots X_n$
- $\delta(q,X_i) = (p,Y,L)$ is same as:
  $X_1\ldots X_{i-1}qX_i\ldots X_n \quad |\text{----} \quad X_1\ldots pX_{i-1}YX_{i+1}\ldots X_n$

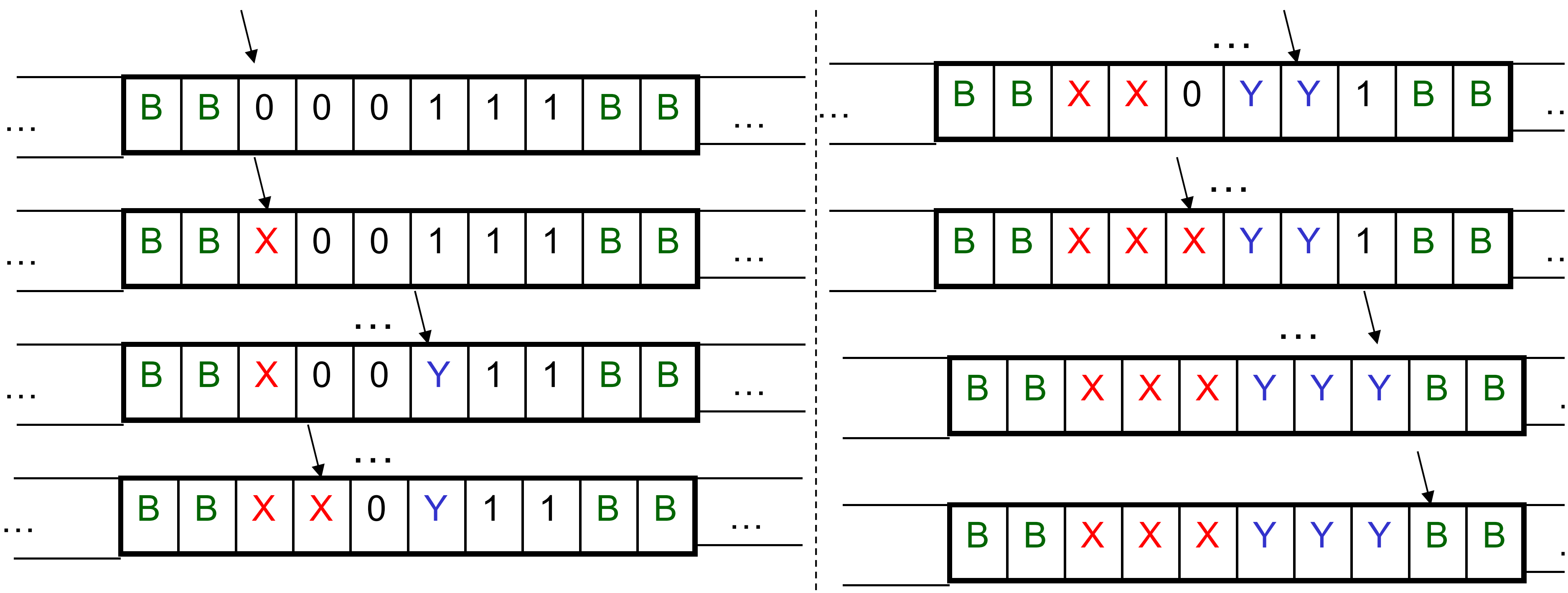
Way to check for Membership

- Is a string *w* accepted by a TM?

- Initial condition:
  - The (whole) input string *w* is present in TM, preceded and followed by infinite blank symbols
- Final acceptance:
  - Accept *w* if TM enters final state and halts
  - If TM halts and not final state, then reject

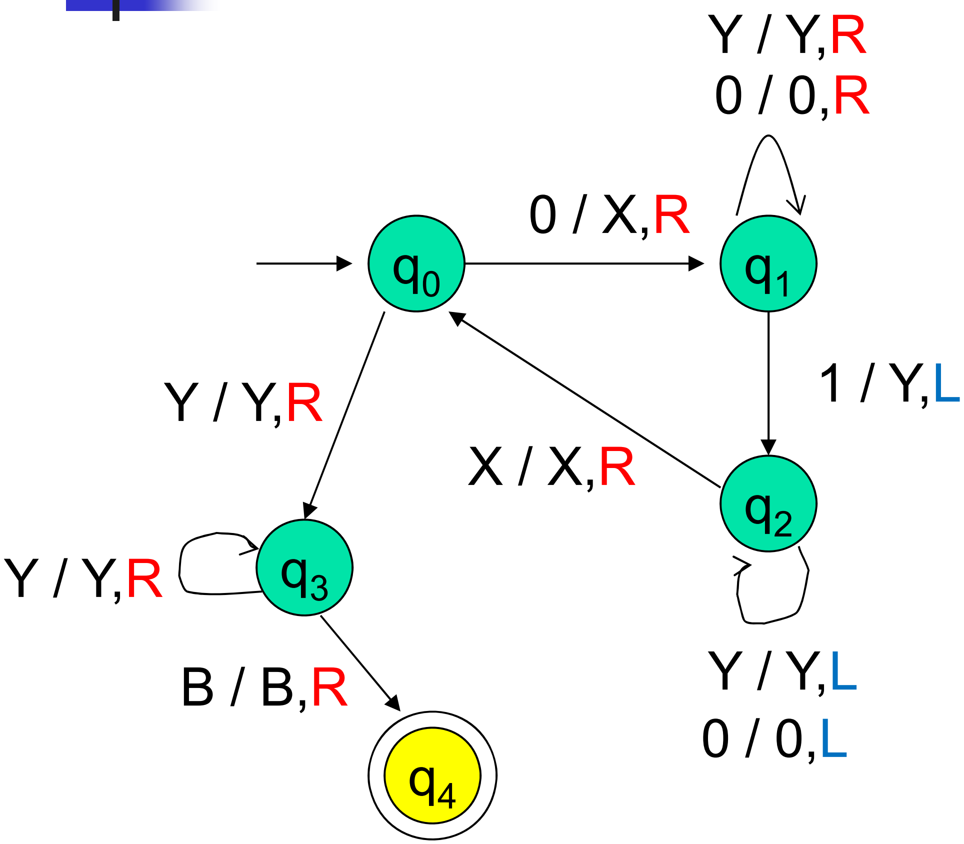
Example: $L = \{0^n1^n \mid n \geq 1\}$

- Strategy: w = 000111



Accept

TM for $\{0^n1^n \mid n\geq 1\}$



1. Mark next unread 0 with X and move right
2. Move to the right all the way to the first unread 1, and mark it with Y
3. Move back (to the left) all the way to the last marked X, and then move one position to the right
4. If the next position is 0, then goto step 1.
   Else move all the way to the right to ensure there are no excess 1s. If not move right to the next blank symbol and stop & accept.

*state diagram representation preferred

TM for $\{0^n1^n \mid n\geq 1\}$

	Next Tape Symbol				
Curr. State	0	1	X	Y	B
→ q_0	(q_1,X,R)	-	-	(q_3,Y,R)	-
q_1	$(q_1,0,R)$	(q_2,Y,L)	-	(q_1,Y,R)	-
q_2	$(q_2,0,L)$	-	(q_0,X,R)	(q_2,Y,L)	-
q_3	-	-	-	(q_3,Y,R)	(q_4,B,R)
$*q_4$	-	--	-	-	-

Table representation of the state diagram

TMs for calculations

- TMs can also be used for calculating values
  - Like arithmetic computations
  - Eg., addition, subtraction, multiplication, etc.

Example 2: monus subtraction

"m -- n" = max{m-n,0}

$0^m 1 0^n \rightarrow$ ...B 0^{m-n} B.. (*if m>n*)
...BB…B.. (*otherwise*)

1. For every 0 on the left (mark X), mark off a 0 on the right (mark Y)
2. Repeat process, until one of the following happens:
   1. // No more 0s remaining on the left of 1
      Answer is 0, so flip all excess 0s on the right of 1 to Bs (and the 1 itself) and halt
   2. //No more 0s remaining on the right of 1
      Answer is m-n, so simply halt after making 1 to B

Give state diagram

Example 3: Multiplication

- 0^m10^n1 (input), $0^{mn}1$ (output)

- <u>Pseudocode</u>:
  1. Move tape head back & forth such that for every 0 seen in 0^m, write n 0s to the right of the last delimiting 1
  2. Once written, that zero is changed to B to get marked as finished
  3. After completing on all m 0s, make the remaining n 0s and 1s also as Bs

Give state diagram

Calculations vs. Languages

A “calculation” is one that takes an input and outputs a value (or values)

The “language” for a certain calculation is the set of strings of the form “<input, output>”, where the output corresponds to a valid calculated value for the input

A “language” is a set of strings that meet certain criteria

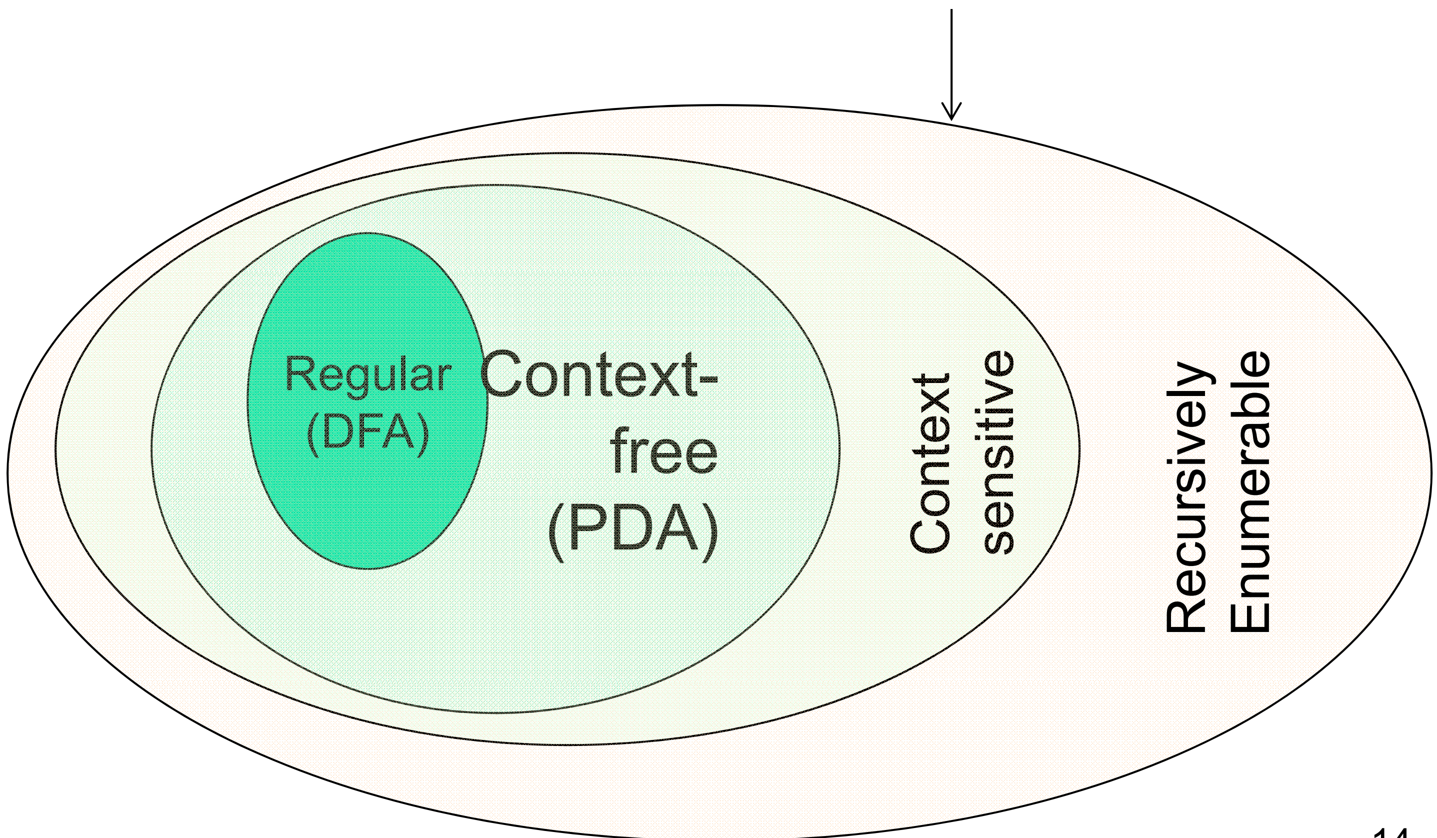
E.g., The language L_{add} for the addition operation

“<0#0,0>”
“<0#1,1>”
…
“<2#4,6>”
…

Membership question == verifying a solution
e.g., is “<15#12,27>” a member of L_{add} ?

Language of the Turing Machines

- *Recursive Enumerable (RE) language*

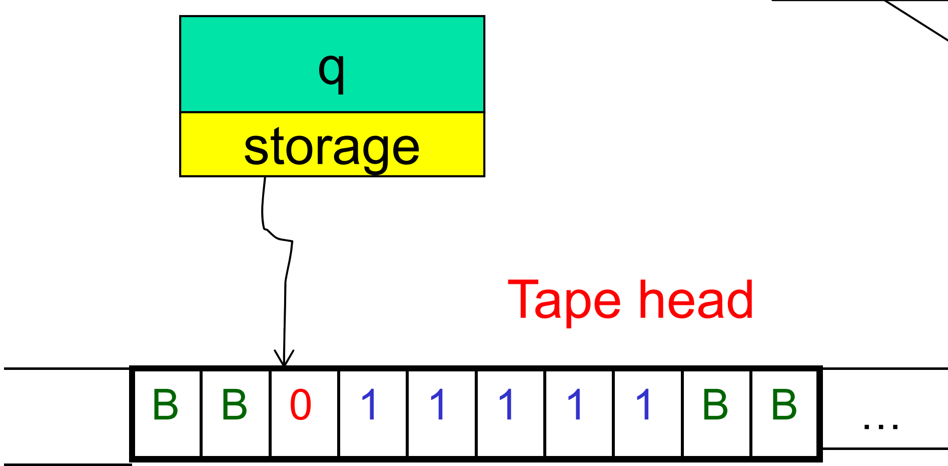


Variations of Turing Machines

TMs with *storage*

Generic description
Will work for both a=0 and a=1

- E.g., TM for 01* + 10*



Current state

Current Storage symbol

Tape symbol

Next state

New Storage symbol

Transition function δ:

- $\delta([q_0,B],a) = ([q_1,a], a, R)$
- $\delta([q_1,a],\overline{a}) = ([q_1,a], \overline{a}, R)$
- $\delta([q_1,a],B) = ([q_2,B], B, R)$

[q,a]: where q is current state, a is the symbol in storage

Are the standard TMs equivalent to TMs with storage?

Yes

Standard TMs are equivalent to TMs with storage - Proof

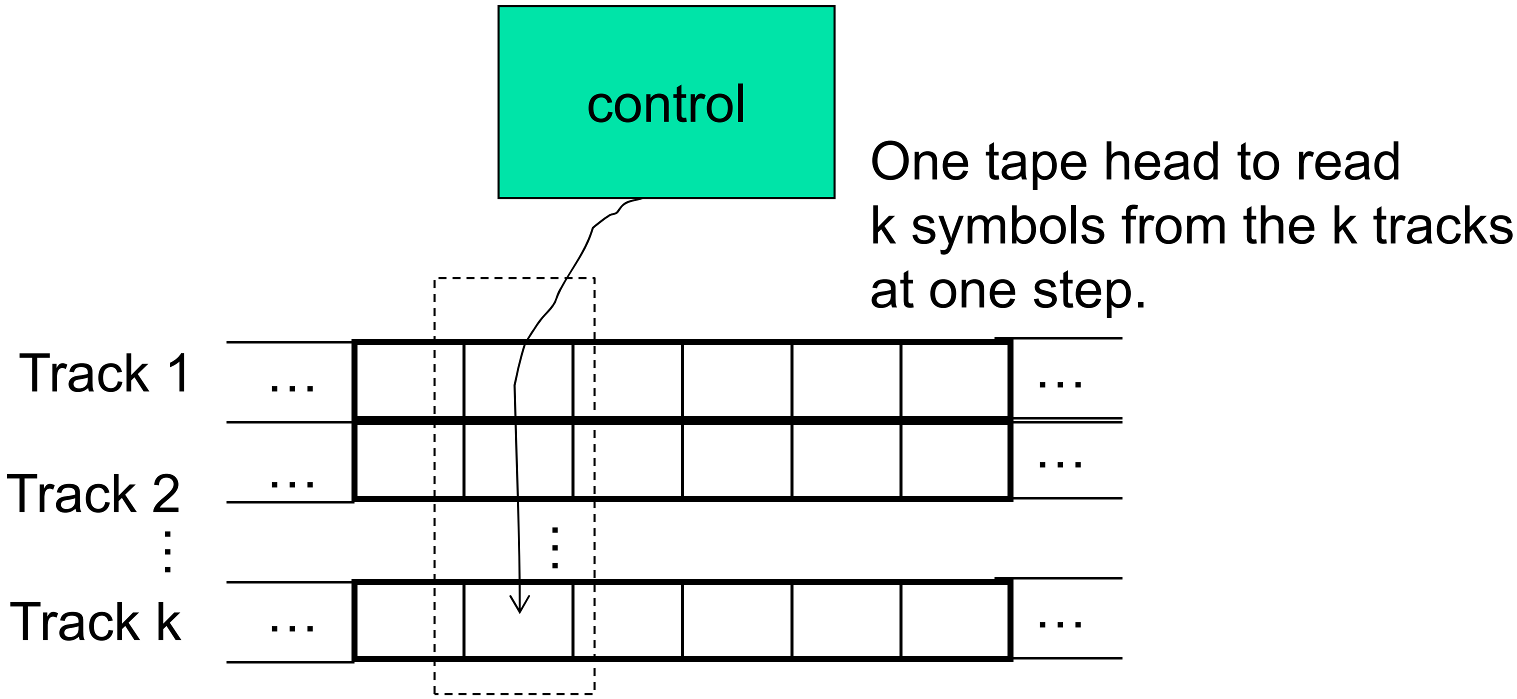
<u>Claim:</u> Every TM w/ storage can be simulated by a TM w/o storage as follows:

- For every [state, symbol] combination in the TM w/ storage:
  - Create a new state in the TM w/o storage
  - Define transitions induced by TM w/ storage

Since there are only finite number of states and symbols in the TM with storage, the number of states in the TM without storage will also be finite

Multi-track Turing Machines

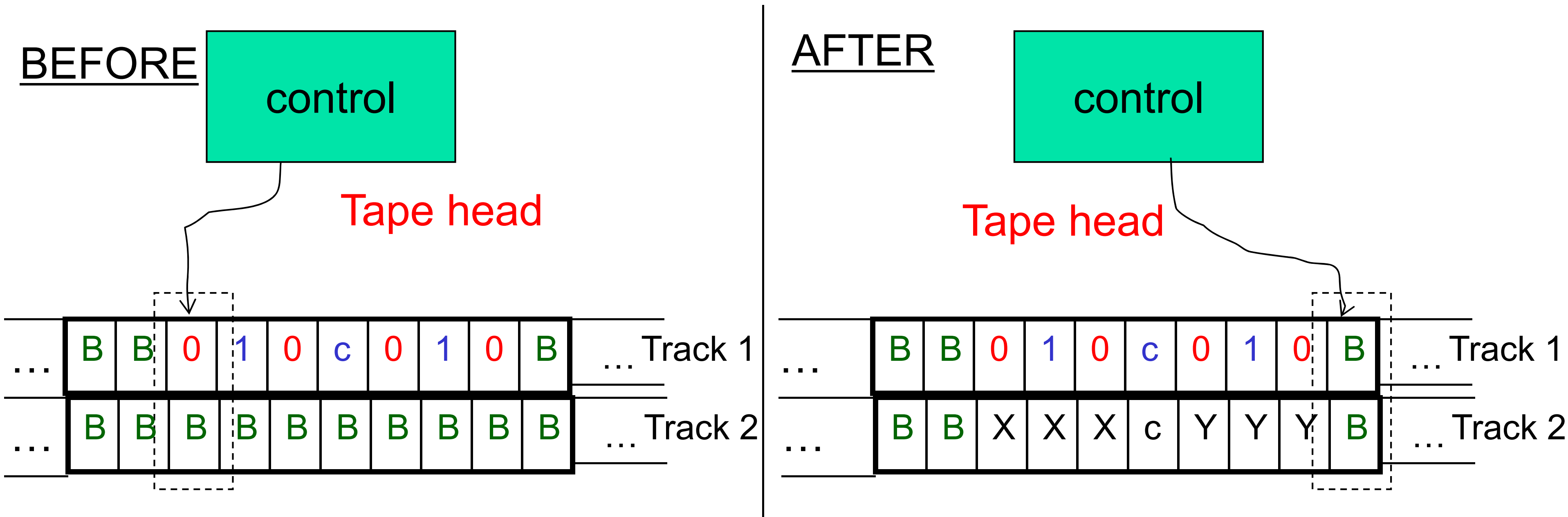
- TM with multiple tracks,
  but just one unified tape head



Multi-Track TMs

- TM with multiple "tracks" but just one head

E.g., TM for $\{wcw \mid w \in \{0,1\}^* \}$ but w/o modifying original input string



Second track mainly used as a scratch space for marking

Multi-track TMs are equivalent to basic (single-track) TMs

- Let M be a single-track TM
  - $M = (Q, \sum, \Gamma, \delta, q_0, B, F)$

- Let M' be a multi-track TM (*k* tracks)
  - $M' = (Q', \sum', \Gamma', \delta', q'_0, B, F')$
  - $\delta'(q_i, <a_1, a_2, \ldots a_k>) = (q_j, <b_1, b_2, \ldots b_k>, L/R)$

- Claims:
  - *For every M, there is an M' s.t. L(M)=L(M').*
    - (proof trivial here)

Multi-track TM ==> TM (proof)

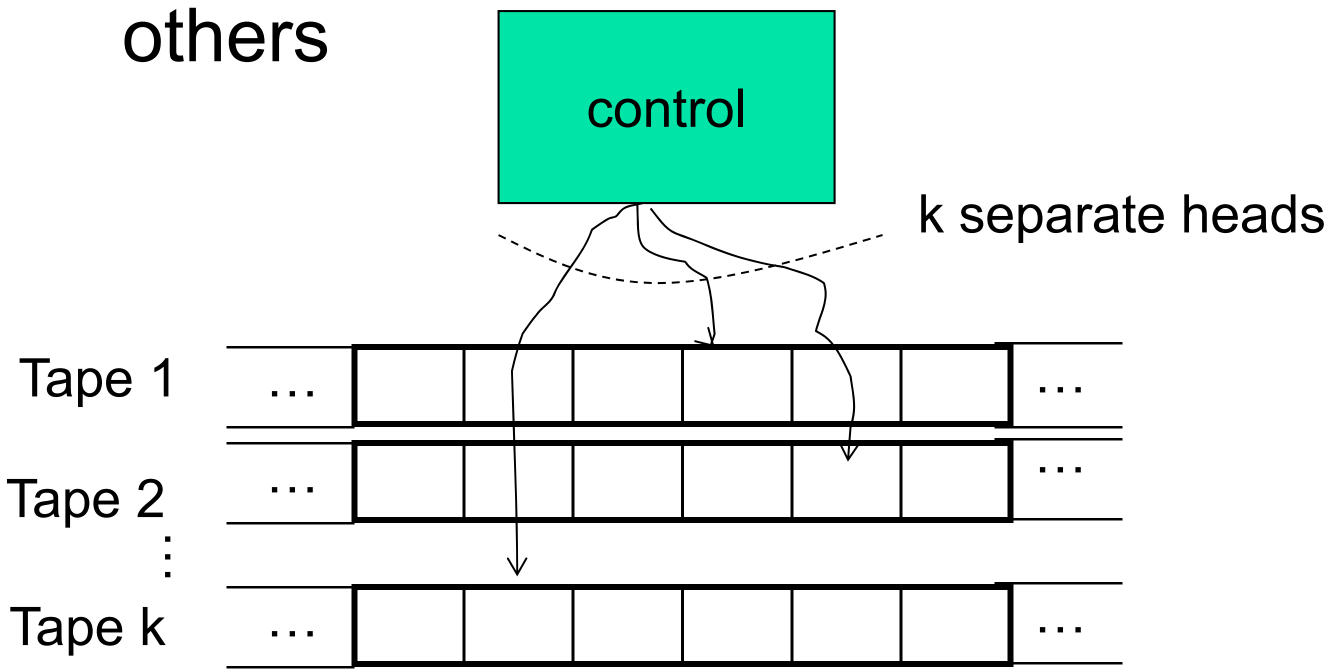
- *For every M', there is an M s.t. L(M')=L(M).*
  - $M = (Q, \sum, \Gamma, \delta, q_0, [B,B,\ldots], F)$
  - Where:
    - $Q = Q'$
    - $\sum = \sum ' \times \sum ' \times \ldots$ (k times for k-track)
    - $\Gamma = \Gamma' \times \Gamma' \times \ldots$ (k times for k-track)
    - $q_0 = q'_0$
    - $F = F'$
    - $\delta(q_i,[a_1,a_2,\ldots a_k]) = \delta'(q_i, <a_1,a_2,\ldots a_k>)$

<u>Main idea:</u>
Create one composite symbol to represent every combination of k symbols

- Multi-track TMs are just a different way to represent single-track TMs, and is a matter of design convenience.

Multi-tape Turing Machines

- TM with multiple tapes, *each tape with a separate head*
  - Each head can move independently of the others



On how a Multi-tape TM would operate

- <u>Initially:</u>
  - The input is in tape #1 surrounded by blanks
  - All other tapes contain only blanks
  - The tape head for tape #1 points to the 1st symbol of the input
  - The heads for all other tapes point at an arbitrary cell (doesn't matter because they are all blanks anyway)

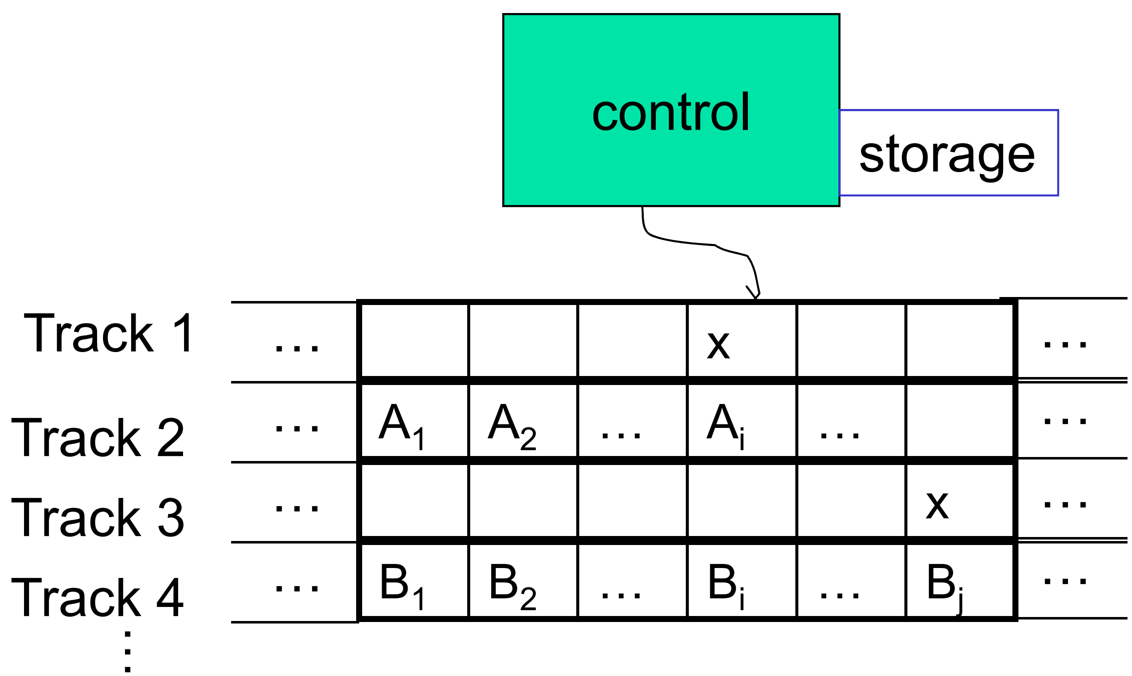
- <u>A move:</u>
  - Is a function (current state, the symbols pointed by <u>all</u> the heads)
  - After each move, each tape head can move independently (left or right) of one another

Multitape TMs ≡ Basic TMs

- Theorem: Every language accepted by a k-tape TM is also accepted by a single-tape TM
- Proof by construction:
  - Construct a single-tape TM with 2k tracks, where each tape of the k-tape TM is simulated by 2 tracks of basic TM
  - k out the 2k tracks simulate the k input tapes
  - The other k out of the 2k tracks keep track of the k tape head positions

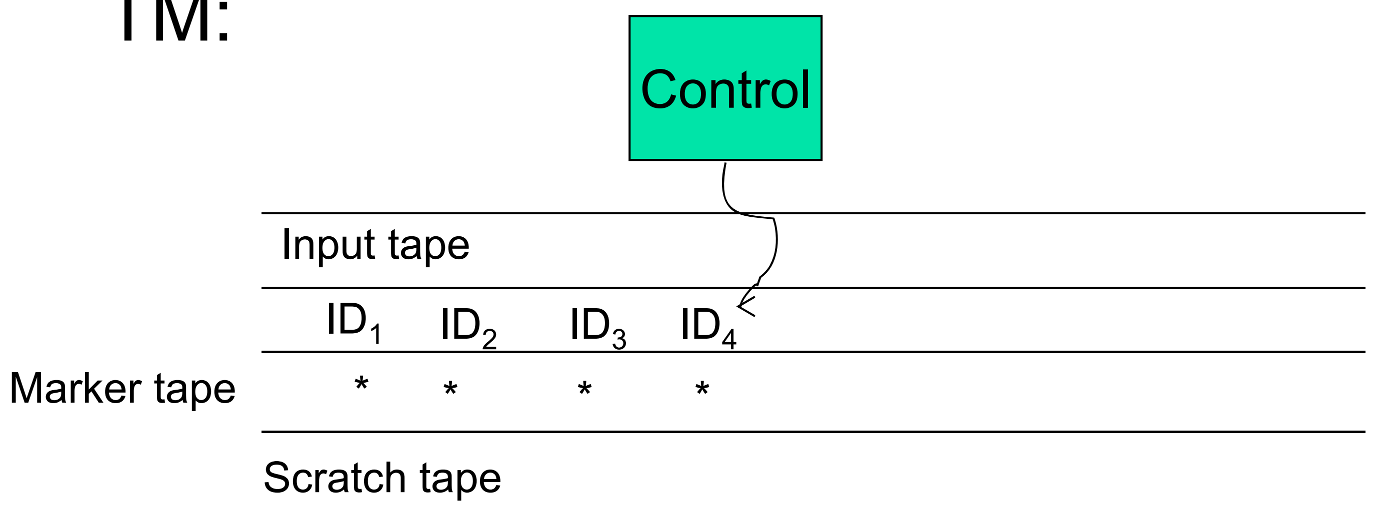
Multitape TMs ≡ Basic TMs …

- To simulate one move of the k-tape TM:
  - Move from the leftmost marker to the rightmost marker (k markers) and in the process, gather all the input symbols into storage
  - Then, take the action same as done by the k-tape TM (rewrite tape symbols & move L/R using the markers)



Non-deterministic TMs

- A TM can have non-deterministic moves:
  - $\delta(q,X) = \{ (q_1,Y_1,D_1), (q_2,Y_2,D_2), \ldots \}$
- Simulation using a multitape deterministic TM:



Summary

- TMs == Recursively Enumerable languages
- TMs can be used as both:
  - Language recognizers
  - Calculators/computers
- ***Basic TM is <u>equivalent</u> to all the below:***
  1. *TM + storage*
  2. *Multi-track TM*
  3. *Multi-tape TM*
  4. *Non-deterministic TM*
- TMs are like universal computing machines with unbounded storage