PART III
Chapter 2

Other Equivalent models of Standard Turing machine

Outline

- Equivalent models of Turing machine
  - multi-tape TMs
  - 2way TMs
  - multi-head TMs
  - 2Dimensional TMs
  - 2Stack machine
  - Counter machine
  - Nondeterministic TMs
- Universal TM

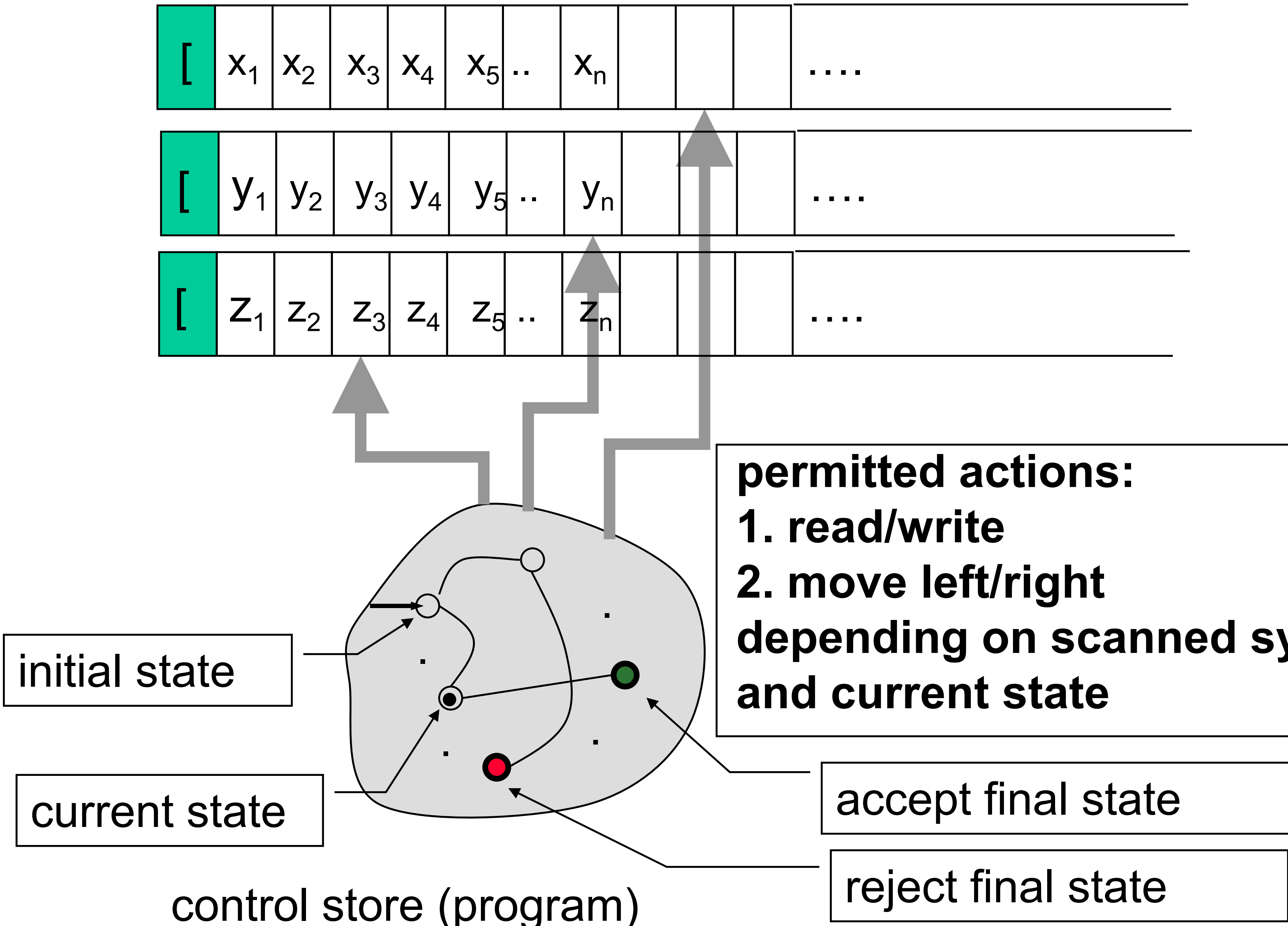
multi-tape TM

- A k-tape ($k \geq 1$) Turing machine is a 9-tuple

  $M = (Q, \Sigma, \Gamma, [, \square, \delta, s, t, r)$ where

  - Q : is a finite set of (program) states like labels in traditional programs
  - Γ : tape alphabet
  - $\Sigma \subset \Gamma$: input alphabet
  - $[\in \Gamma - \Sigma$: The left end-of-tape mark
  - $\square \in \Gamma - \Sigma$ is the blank tape symbol
  - $s \in Q$: initial state
  - $t \in Q$: the accept state
  - $r \neq t \in Q$: the reject state and
  - $\delta: (Q - \{t,r\}) \times \Gamma^k \rightarrow Q \times (\Gamma \cup \{L,R\})^k$ is a *total* transition function with the restriction: if $\delta(p, x_1,\ldots,x_k) = (q, y_1,\ldots,y_k)$ then if $x_j = [\Rightarrow y_j = [$ or R. i.e., the TM cannot overwrite other symbol at left-end and never move off the tape to the left.

3-tape Turing machine



permitted actions:
1. read/write
2. move left/right
depending on scanned symbols
and current state

Equivalence of STMs and Multi-tape TMs

- **M = (Q,Σ,Γ, [, □, δ, s, t,r): a k-tape TMs**

==> M can be simulated by a k-track STM

M' = (Q',Σ,Γ', [, □ , δ', s, t',r') where

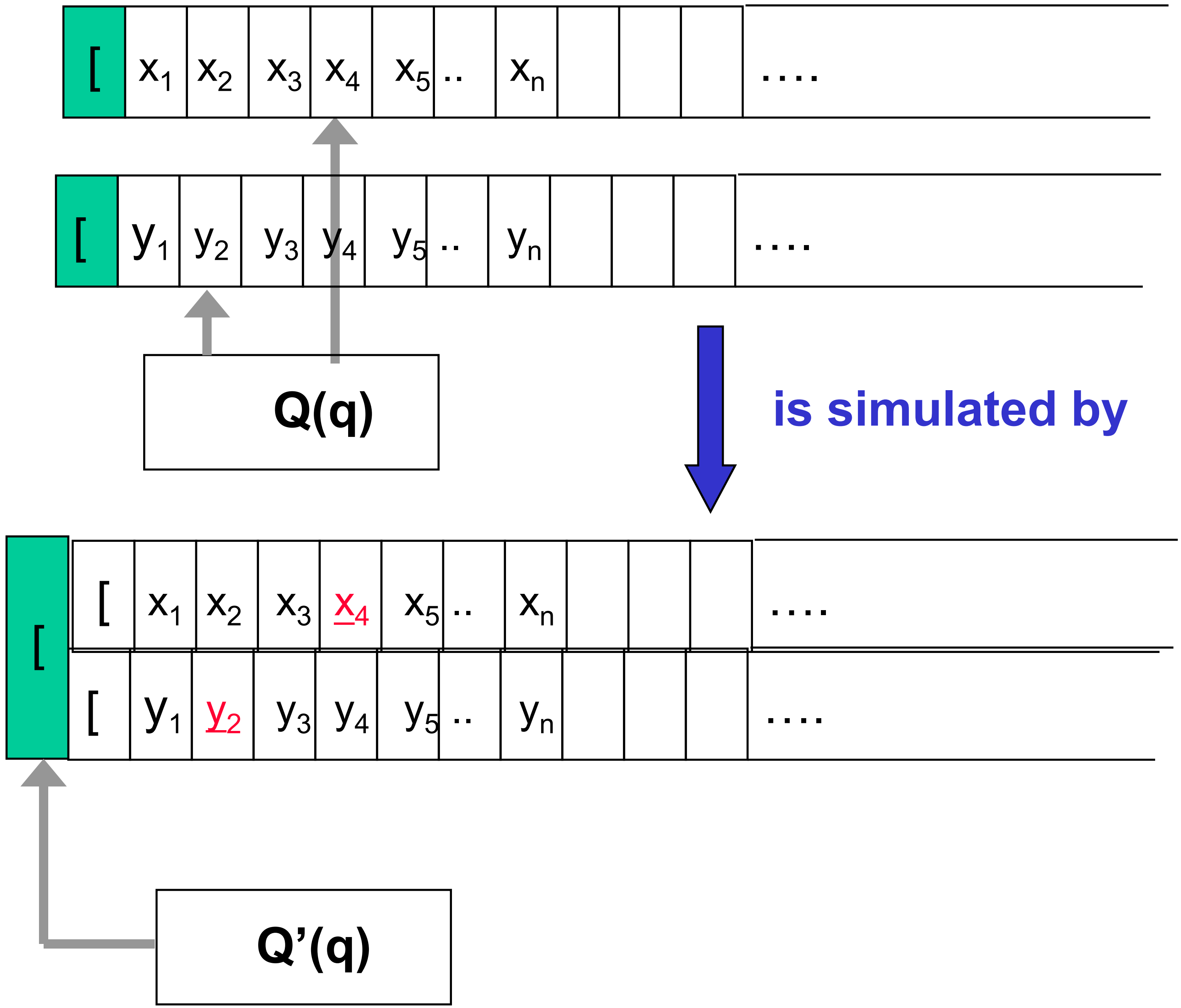
▯ $\Gamma' = \Gamma \cup (\Gamma \cup \underline{\Gamma})^k$ where $\underline{\Gamma} = \{ \underline{a} \mid a \in \Gamma\}$.

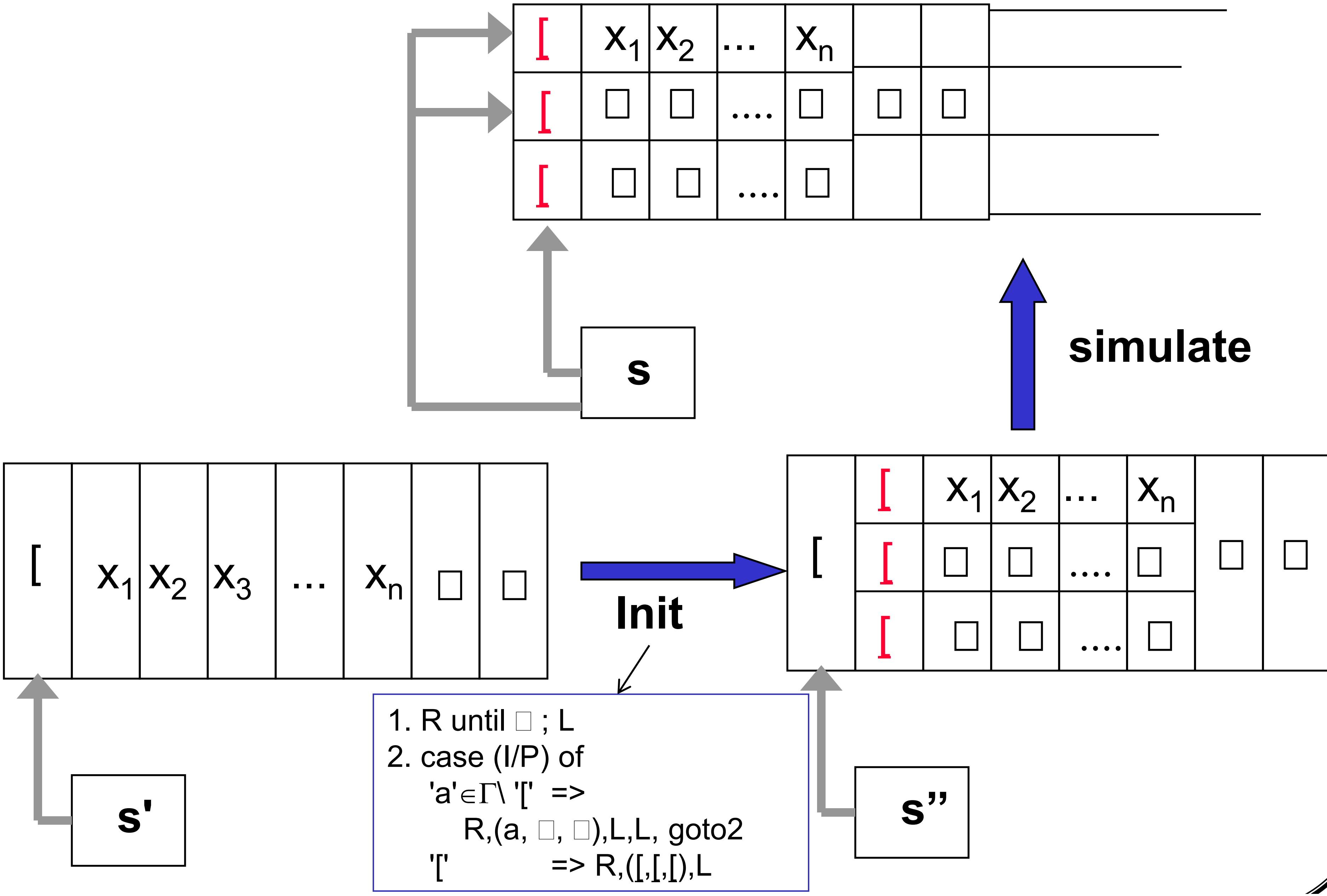
- **M' = init • M'' where the task of Init is to convert initial input tape content : $[x_1x_2...x_n \square^\omega$ into**

	[	x_1	x_2	…	x_n		
[	[	□	□	....	□	□	□
	[	□	□	…	□		

and then go to the initial state s'' of M'' to start simulation of M.

- **Each state q of M is simulated by a submachine M_q of M'' as follows:**


[x1 x2 x3 x4 x5 .. xn ....
[y1 y2 y3 y4 y5 .. yn ....
Q(q)
is simulated by
[
[x1 x2 x3 x4 x5 .. xn ....
[y1 y2 y3 y4 y5 .. yn ....
Q'(q)


[
x_1 x_2 ... x_n
□ □ .... □ □ □
□ □ .... □
s
simulate
[x_1 x_2 x_3 ... x_n □ □
Init
[[x_1 x_2 ... x_n
[□ □ .... □ □ □
[□ □ .... □
1. R until □ ; L
2. case (I/P) of
'a'∈Γ\ '[' =>
R,(a, □, □),L,L, goto2
'[' => R,([,[,[),L
s'
s''

How does M'' simulate M ?

- let $(q, x^1, p_1, y^1),\ldots,(q, x^m, p_m, y^m)$ be the set of all instructions (starting from state q) having the form $\delta(q, x^i) = (p_i, y^i)$, where $x^i, y^i = (x^i_1, x^i_2, \ldots, x^i_k), (y^i_1, y^i_2, \ldots, y^i_k)$. Then M_q behaves as follows:

0. **[terminate?]** if q = t then accept; if q = r then reject.

1. **[determine what symbols are scanned by tape heads]**

   for j = 1 to k do { // determine symbol scanned by jth head

   move right until the symbol at the jth track is underlined,

   remember which symbol is underlined (say $\underline{a}_j$) in the control store and then move to left end.}

2. **[perform action: $\delta(q, a_1,\ldots,a_k) = (p, b_1,\ldots,b_k)$ for each tape head]**

   for j= 1 to k do{ // perform b_j at the jth tape

case1. $b_j = b \in \Gamma$ ==>Move R until $\underline{a}_j$; replace symbol $\underline{a}_j$ at jth track by $\underline{b}_j$

case2. b_j = R ==>Move R unitl $\underline{a}_j$, replace $\underline{a}_j$ by a_j and underline its right neighbor symbol.

case3: b_j = L. Similar to case 2. Finally move to left end. }

3. **[go to next state]** go to start state of M_p to simulate M at state p.

Running time analysis

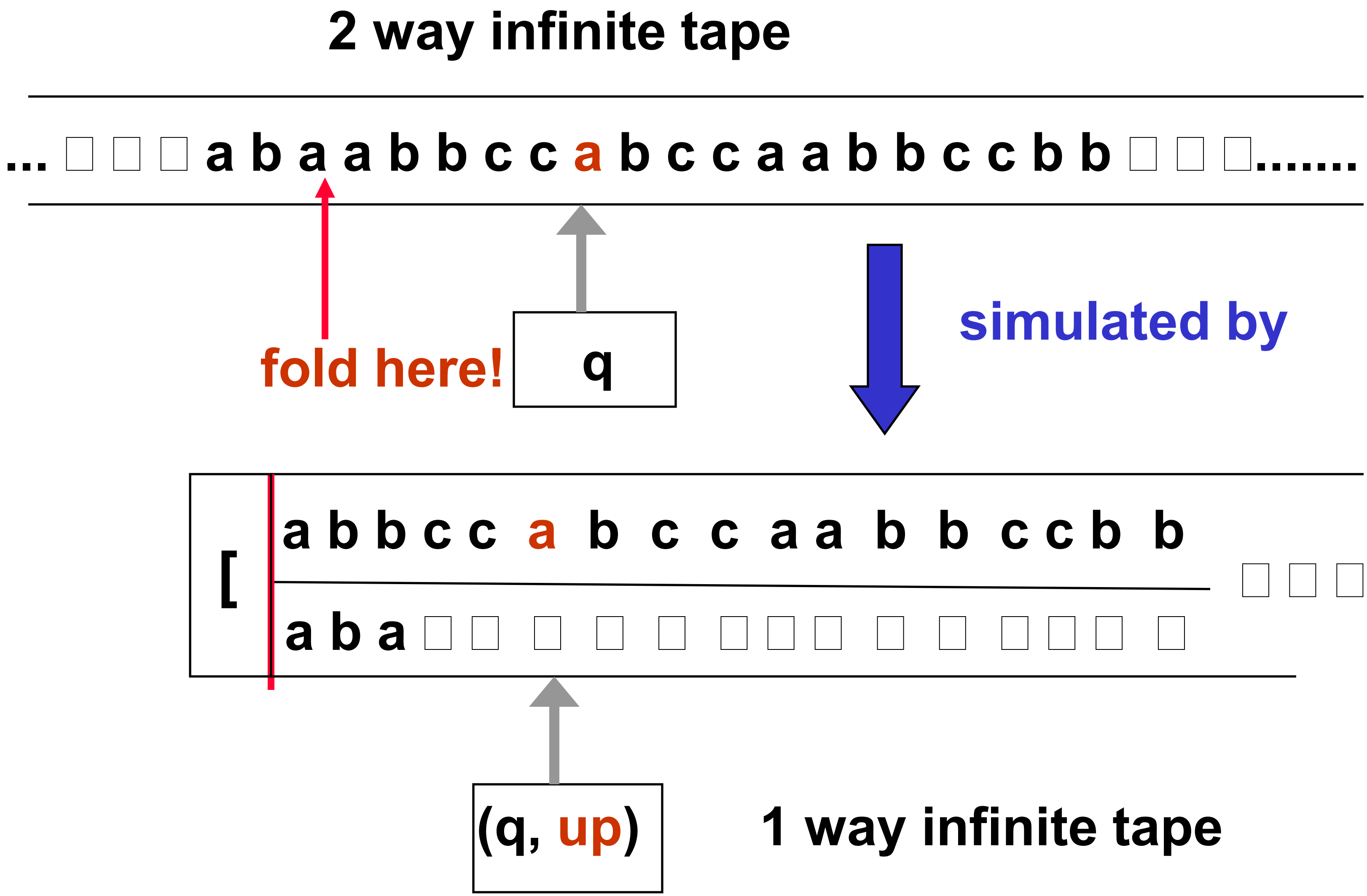
- How many steps of M'' are needed to simulate one step of execution of M ?
- Sol:
- Assume the running time of M on input x of length n is f(n).
  - step 1 requires time : O(k x 2 f(n))
  - Step 2 requires time: O(k x 2 f(n))
  - Step 3 requires O(1) time
  - => Each step requires time O(4k x f(n)).
  - and total time required to simulate M = f(n) x O(4k f(n))
  - = O $(f(n)^2)$.

Turing machine with 2 way infinite tape

- **A 2way single tape Turing machine is a 8-tuple**

  M = (Q,Σ,Γ, [, □, δ, s, t,r) where

  - **Q : is a finite set of (program) states like labels in traditional programs**
  - **Γ : tape alphabet**
  - **$\Sigma \subset \Gamma$: input alphabet**
  - **$[\in \Gamma - \Sigma$: The left end-of-tape mark (no longer needed!!)**
  - **$\square \in \Gamma - \Sigma$ is the blank tape symbol**
  - **$s \in Q$: initial state**
  - **$t \in Q$: the accept state**
  - **$r \neq t \in Q$: the reject state and**
  - **δ: (Q - {t,r})x Γ --> Qx(Γ∪{L,R}) is a *total* transition function.**


2 way infinite tape
... □ □ □ a b a a b b c c a b c c a a b b c c b b □ □ □.......
fold here!
q
simulated by
[
a b b c c a b c c a a b b c c b b
a b a □ □ □ □ □ □ □ □ □ □ □ □ □ □
□ □ □
(q, up)
1 way infinite tape

Equivalence of STMs and 2way TMs

- M = (Q,Σ,Γ, □ , δ, s, t,r): a 2way TM

==> M can be simulated by a 2-track STM:

M' = (Q',Σ,Γ', [, □ , δ', s, t',r') where

- Q' = Q U (Qx{u,d}) U {...},
- Γ'= Γ U Γ^2 U{ [},
- M' = init • M'' where the task of Init is to convert initial input tape content : $□^\omega x_1 x_2 \ldots x_n □^\omega$ into

[	x_1	x_2	...	x_n	□	□	....
	□	□	...	□			

and then go to the initial state s'' of M'' to start simulation of M.

- Each instruction of M is simulated by one or two instructions of M'' as follows:

How to simulate 2way tape TM using 1way tape TM

Let $\delta(q,x) = (p, y)$ be an instruction of M then:

case 1: $y \in \Gamma$

- ==> $\delta''((q,u), (x,z)) = ((p,u),(y,z))$ and
- $\delta''((q,d),(z,x)) = ((p,d), (z,y))$ for all $z \in \Gamma$

case2 : $y = R$.

- ==> $\delta''((q,u), (x,z)) = ((p,u),R)$ and $\delta''((q,d),(z,x)) = ((p,d), L)$
- for all $z \in \Gamma$.

case 3: $y = L$.

- ==> $\delta''((q,u), (x,z)) = ((p,u),L)$ and $\delta'((q,d),(z,x)) = ((p,d), R)$
- for all $z \in \Gamma$.

+additional conditions

- 1. left end => change direction:
- $\delta''((q,u), [) = ((q,d),R)$, $\delta''((q,d),[) = ((q,u),R)$ for all $q \notin \{t,r\}$.
- 2. □ → (□, □): $\delta''((q, \alpha), □) = ((q, \alpha), (□, □))$

Properties of r.e. languages

- **Theorem: If both L and ~L are r.e., then L (and ~L) is recursive.**

Pf: Suppose $L=L(M_1)$ and $\sim L = L(M_2)$ for two STM M_1 and M_2.

Now construct a new 2 -tape TM M as follows:

on input: x

1. copy x from tape 1 to tape 2. // COPY
2. Repeat { execute 1 step of M_1 on tape 1;
   execute 1 step of M_2 on tape 2 }
   until either M_1 halts or M_2 halts.
3. If M_1 accepts or M_2 rejects then [M] accept
   If M_2 accepts or M_1 rejects then [M] reject. // 2+3 = $M_1 \| M_2$ defined later

So if $x \in L$ => M_1 accept x or M_2 rejects => M accept
if $x \notin L$ => M_2 accept or M_1 rejects ==> M reject.
Hence M is total and L =L(M) and L is recursive.

Interleaved execution of two TMs

- $M_1 = (Q_1,\Sigma,\Gamma, [, \square , \delta_1, s_1, t_1,r_1)$; $M_2 = (Q_2,\Sigma,\Gamma, [, \square , \delta_2, s_2, t_2,r_2)$ where $\delta_1: Q_1 \times \Gamma \text{-->} Q_1 \times (\Gamma \cup \{L,R\})$; $\delta_2: Q_2 \times \Gamma \text{-->} Q_2 \times \Gamma \cup (\{L,R\})$;

==> $M =_{def} M_1 \| M_2 = (Q_1 \times Q_2 \times \{1,2\} \cup \{T,R\} , \Sigma,\Gamma, [, \square, \delta,s,T,R)$ where

1. $\delta: Q_1 \times Q_2 \times \{1,2\} \times \Gamma^2 \text{-->} (Q_1 \times Q_2 \times \{1,2\}) \times (\Gamma \cup \{L,R\})^2$ is given by
   - Let $\delta_1(q_1,a) = (p_1,a')$ and $\delta_2(q_2,b) = (p_2,b')$ then
   - $\delta((q_1,q_2,1),(a,b)) = ((p_1,q_2,2), a',b)$ and
   - $\delta((q_1,q_2,2),(a,b)) = ((q_1,p_2,1), a,b')$
2. M has initial state $s = (s_1,s_2,1)$.
3. M has an accepting state T from states $\{(t_1,q_2,1) \mid q_2 \in Q_2\} \cup \{(q_1,r_2,2) \mid q_1 \in Q_1\}$ and a rejecting state R from states $\{(q_1,t_2,2) \mid q_1 \in Q_1\} \cup \{(r_1,q_2,2) \mid q_2 \in Q_2\}$.
   - $\delta((t_1,q_2,1),(a,b)) \rightarrow (T,(a,b))$, $\delta((q_1,r_2,2),(a,b)) \rightarrow (T,(a,b))$
   - $\delta((r_1,q_2,1),(a,b)) \rightarrow (R,(a,b))$, $\delta((q_1,t_2,2),(a,b)) \rightarrow (R,(a,b))$
4. By suitably designating halting states of M as accept or reject states, we can construct machine accepting languages that are boolean combination of $L(M_1)$ and $L(M_2)$. Ex: $T = \{(t_1,q_2,1) \mid q_2 \in Q_2\}$ and $R = \{(q_1,t_2,2) \mid q_1 \in Q_1\}$ in previous example.

A programming Language for TMs and Universal TM

- Proposed Computation models
  - TMs (DTM, NDTM, RATM, multi-tape, 2way, multi Dimensional, multi-head, and their combinations,...)
  - Grammars
  - u-recursive function,
  - λ-calculus
  - Counter Machine
  - 2STACK machine
  - Post system,...
- All can be shown to have the same computation power
- Church-Turing Thesis:
  - A language or function is computable iff it is Turing-computable (i.e., can be computed by a total TM).
  - An algorithm is one that can be carried out by a total TM.

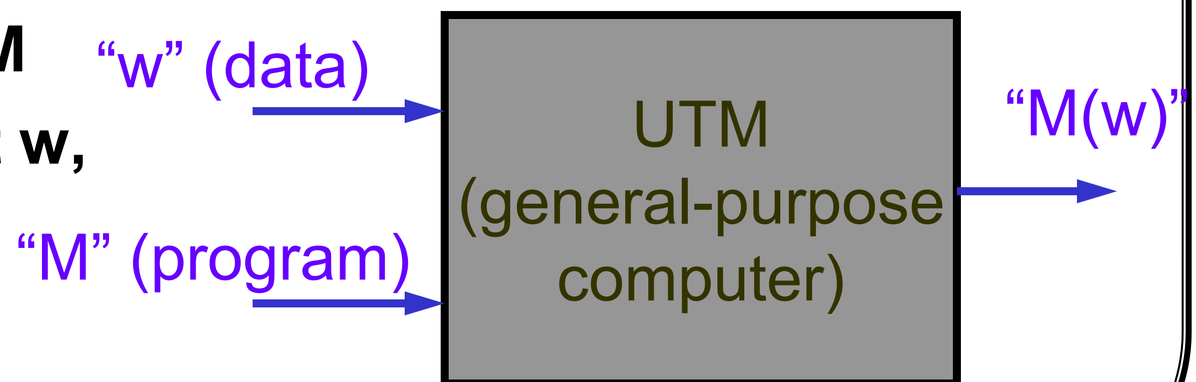
Universal Turing machine

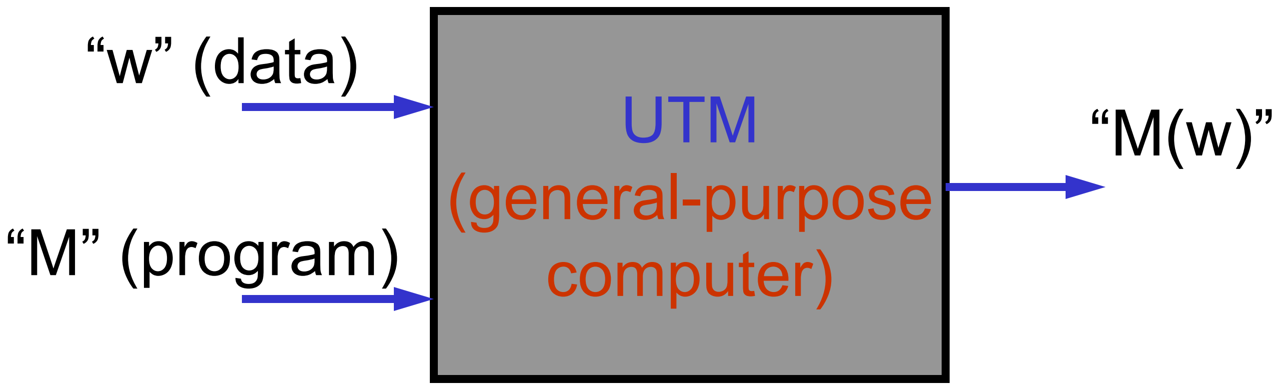
- TMs considered so far are dedicated in the sense that each of their control store is hard-wired to solve one particular problem
  - e.g., TMs for +, x, copy,...
- Problem: Is there any TM that can compute what all TMs can compute ?

Yes!! we call it universal TM (UTM), which is nothing special but a general-purpose TM.

- UTM is a TM simulator,i.e.,

given a spec “M” of a TM M and a desc “w”of an input w, UTM can simulate the execution of M on w.




“w” (data)
“M” (program)
UTM
(general-purpose
computer)
“M(w)”

TL : a programming language for TMs

- M = (Q,Σ,Γ, [, □, δ, s, t,r) : a STM.
- TM M can be described by a string (i.e., a TL-program) as follows:
- Tape symbols of M are encoded by strings from a{0,1}*
  - blank(□) ==> a0 left-end [==> a1
  - R ==> a00 L ==> a01
  - others => a10,a11,a000,a001,….
- State symbols of M are encoded by strings from q{0,1}*
  - start state s ==> q0;
  - accept state t ==> q1, reject state r ==> q00;
  - other states => q01,q10,q11, q111,…
- For b ∈ ΓU{R,L} U Q, we use e(b) ∈ a{0,1}* U q{0,1}* to denote the encoding of b.

An example

- M = (Q,Σ,Γ, [, □ , δ, s, t,r) where
  - Q = {s, g, r,t }, Γ = { [, a, □} and
  - δ = { (s, a, g, □), (s, □ ,t, □), (s, [, s, R),
  - (g, a, s, a), (g, U, s, R), (g, [, r, R) }

==>

- Suppose state and tape symbols are represented in TL as follows:
- s => q0 ; t => q1; r => q00; g => q01
- □ => a0; [=> a1; R => a00; L => a01;
- a => a10
- Hence a string: '[aa□a' ∈ Γ* can be encoded in TL as
- e([aa□a) = "[aa□a" $=_{def}$ a1a10a10a0a10

Describe a TM by TL

- Let $\delta = \{ \alpha_j \mid \alpha_j = (p_j, a_j, q_j, b_j); j = 1 ..n\}$ be the set of all instructions. ==>
- M can be encoded in TL as a string
  - e(M) = "M" $\in$ {q,a,0,1,[, (,), ‚ }*
  - $=_{def}$ e(α_1),e(α_2),e(α_3),…,e(α_n)
  - where for j = 1 to n,
  - e(α_j) $=_{def}$ '(' e(p_j) ',' e(a_j) ',' e(q_j) ',' e(b_j) ')'
  - ex: for the previous example: we have
  - δ = { (s, a, g, □), (s, □,t, □), (s, [, s, R),
  - (g, a, s, a), (g, U, s, R), (g, [, r, R) } hence
  - e(M) ="M" = '(q0,a10,q01,a0),(q0,a0,q1,a0),...
  - ...,(q01,a1,q00,a00)'

TL and UTM U

- Let Σ_0= {q,a,0,1,(,), , } ==> the set of TL-programs,

  TL $=_{def}$ { x | x = e(M) for some STM M }

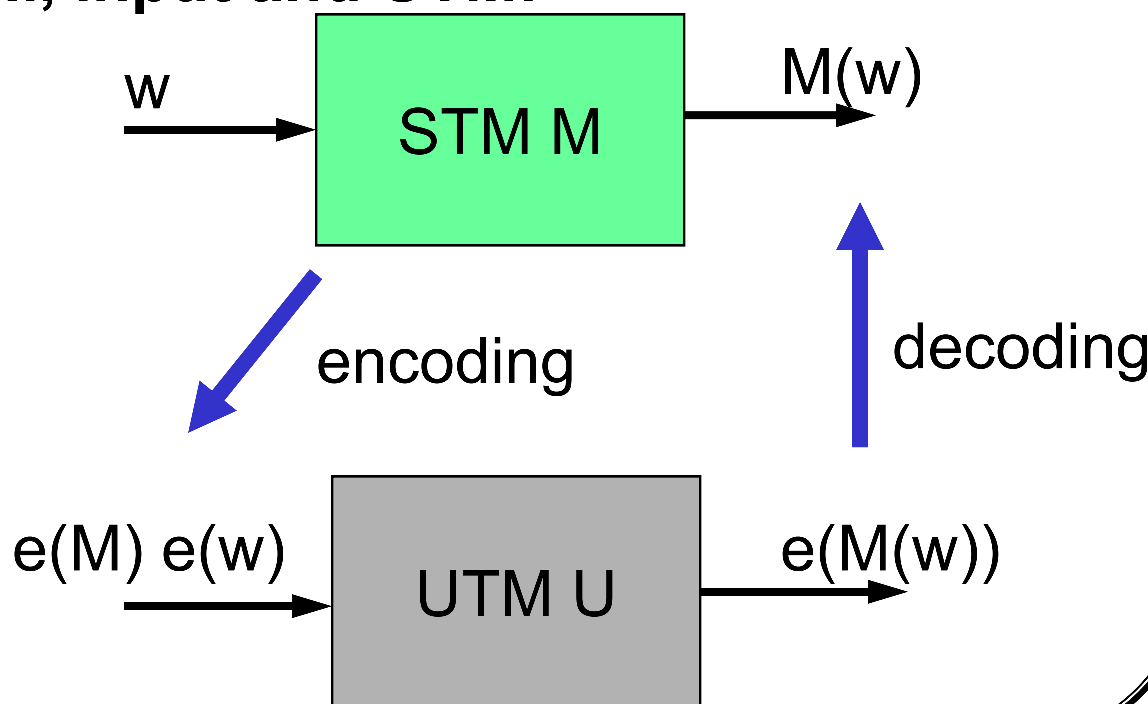
  is the set of all string representations of STMs.

and $\Gamma_0 = \Sigma_0$ U { □,[} is the tape alphabet of UTM U.

Note: Not only encoding TMs, TL can also encode data.

- Relationship between TM, input and UTM:

Note: if such U exists, then we need not implement other TMs



The design of UTM U

- We use U(e(M)e(w)) to denote the result of UTM U on executing input e(M)e(w).
- U will be shown to have the property: for all machine M and input w,

  M halts on input w with result M(w) iff

  U halts on input e(M)e(w) with result e(M(w))

  i.e., e(M(w)) = U(e(M)e(w)).
- U can be designed as a 3-tape TM.
  - 1st tape : first store input "M" "w"; and then used as the [only working] tape of M and finally store the output.
  - 2nd tape: store the program "M" (instruction table)
  - 3rd tape: store the current state of M (program counter)

- U behaves as follows:

1. [Initially:]
    - 1. copy "M" from 1st tape to 2nd tape.
    - 2. shift "w" at the 1st tape down to the left-end
    - 3. place 'q0' at the 3rd tape (PC).
2. [simulation loop:] // between each simulation step, 2,3rd tape heads point to left-end; and 1st tape head points to the a pos of the encoded version of the symbol which the simulated machine M would be scanning.

Each step of M is simulated by U as follows:

2.1 [halt or not] If PC =e(t) or e(r) ==> acept or reject, respectively.

2.2 [Instruction fetch] U scans its 2nd tape until it finds a tuple (qα,aβ,qγ,aλ) s.t. (1) qα matches PC and (2) aβ matches 1st tape's encoded scanned symbol

2.3 [Instruction execution] :

- 1. change PC to qγ.
- 2. perform action suggested by aλ.
- if aλ = e(b) with b∈Γ ==> write aλ at the 1st tape head pos.
- if aλ = e(L) ==> U move 1st tape head to the previous
- a position.
- if aλ = e(R) ==> U move 1st tape head to the next a
- position or append $a0^J$ to the 1st tape in
- case such an 'a' cannot be found.

Theorem: *M(w) accepts, rejects or does not halt iff U("M""w") accepts, rejects or does not halt, respectively.*