Chapter 6

Turing Machines

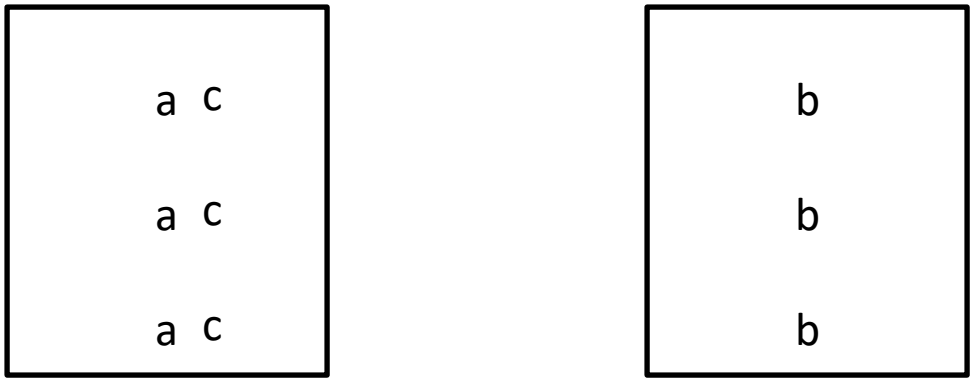
A General Model of Computation

- Both finite automata and pushdown automata are models of computation
  - Each receives an input string and executes an algorithm to obtain an answer, following a set of rules specific to the machine type
- It is easy to find examples of languages that cannot be accepted because of the machine's limitations:
  - An FA cannot accept *SimplePal* $\{xcx^r \mid x \in \{a, b\}^*\}$
  - A PDA cannot accept *AnBnCn* = $\{a^n b^n c^n \mid n \geq 0\}$ or *XcX* = $\{xcx \mid x \in \{a,b\}^*\}$

- What can be done with PDA model to accept *AnBnCn?*

- A PDA-like machine with *two* stacks can accept *AnBnCn*

Input: aaabbbccc



- What can be done with FA (or PDA) model to accept *XcX?*

$$XcX = \{xcx \mid x \in \{a,b\}^*\}$$

- Let's use a *queue* instead of a stack to accept *XcX*

Input: aabbcaabb

a	a	a	a	a	a	b	b	c
	a	a	a	a	b	b	c	
		b	b	b	b	c		
			b	b	c			
				c				

Input:

a	a	b	b	c	a	a	b	b

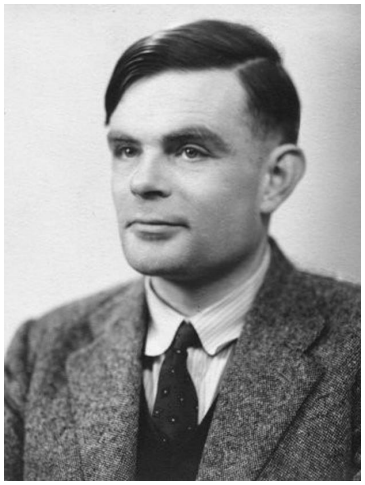
A General Model of Computation (cont'd.)

- In both cases, it might seem that a machine is being specifically developed to handle one language, but it turns out that both these devices have substantially more computing power than either an FA or a PDA

- Either one is a reasonable candidate for a model of general-purpose computation

A General Model of Computation

- The abstract model we will study instead is the Turing machine
  - It is not obtained by adding data structures onto a finite automaton
  - Rather, it predates the FA and PDA models (Alan Turing's contributions date from the 1930's)
- A Turing machine is not *just* the next step beyond a pushdown automaton
  - According to the Church-Turing thesis, **it is a general model of computation, potentially able to execute any algorithm**



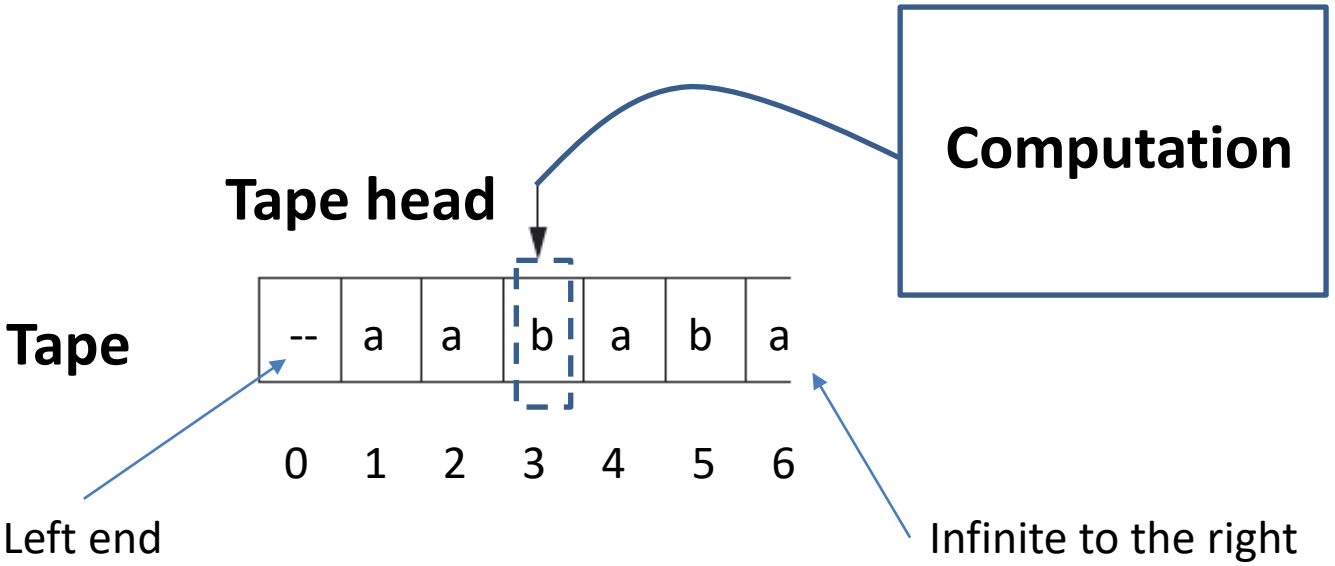
Alan Turing
1912-1954

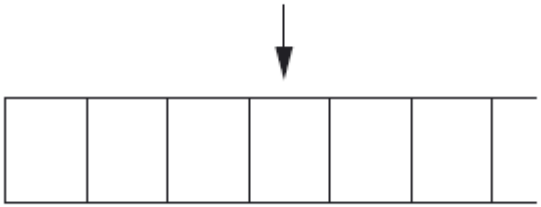
This is our main requirement for the new model

A General Model of Computation (cont'd.)

- Turing's objective was to demonstrate the inherent limitations of algorithmic methods. This is why he wanted his device to be able to execute any algorithm that a human computer could
- To formulate his computational model, he considered a human being working with a pencil and paper
- As a result, he postulated that the steps a computer takes should include these:
  - Examine an individual symbol on the paper
  - Erase a symbol or replace it by another
  - Transfer attention from one symbol to a nearby one

- For simplicity, Turing specified a linear *tape* which has a left end and is potentially infinite to the right
  - The tape is marked off into squares
  - Each square holds one symbol
  - We can enumerate the squares, but that's not part of the model
- We visualize the reading and writing as being done by a *tape head*, which at any time is centered on a single square
- The tape serves as input, output, and memory





- **One crucial difference between a Turing machine and an FA or PDA is that a Turing machine is not restricted to a single pass through the input**
- We will focus on two primary objectives of a Turing machine
  - Accepting a language
  - Computing a function
- The first is similar to what we've done so far

A General Model of Computation (cont'd.)

- A Turing machine will have two *halt* states, one denoting **acceptance** (h_a) and the other **rejection** (h_r)
  - More than two are unnecessary; unlike an FA, the complete input string is on the tape initially, and a *separate answer for each prefix is not required*

- Unlike FAs and PDAs (or at least PDAs without Λ-transitions), **Turing machines may never stop**
  - *Very important for the analysis of algorithms, and (in)tractable problems*

- Definition: A Turing Machine (TM) is a 5-tuple $T = (Q, \Sigma, \Gamma, q_0, \delta)$, where:
  - Q is a finite set of states
    - The two halt states h_a and h_r are not elements of Q
  - The input alphabet Σ and the tape alphabet Γ are both finite sets, with $\Sigma \subseteq \Gamma$
    - The blank symbol Δ is not an element of Γ
  - q_0, the initial state, is an element of Q
  - The transition function is

$$\delta : Q \times (\Gamma \cup \{\Delta\}) \rightarrow (Q \cup \{h_a, h_r\}) \times (\Gamma \cup \{\Delta\}) \times \{\mathrm{R}, \mathrm{L}, \mathrm{S}\}$$

Current state

Current symbol on tape

New state

Accept

Reject

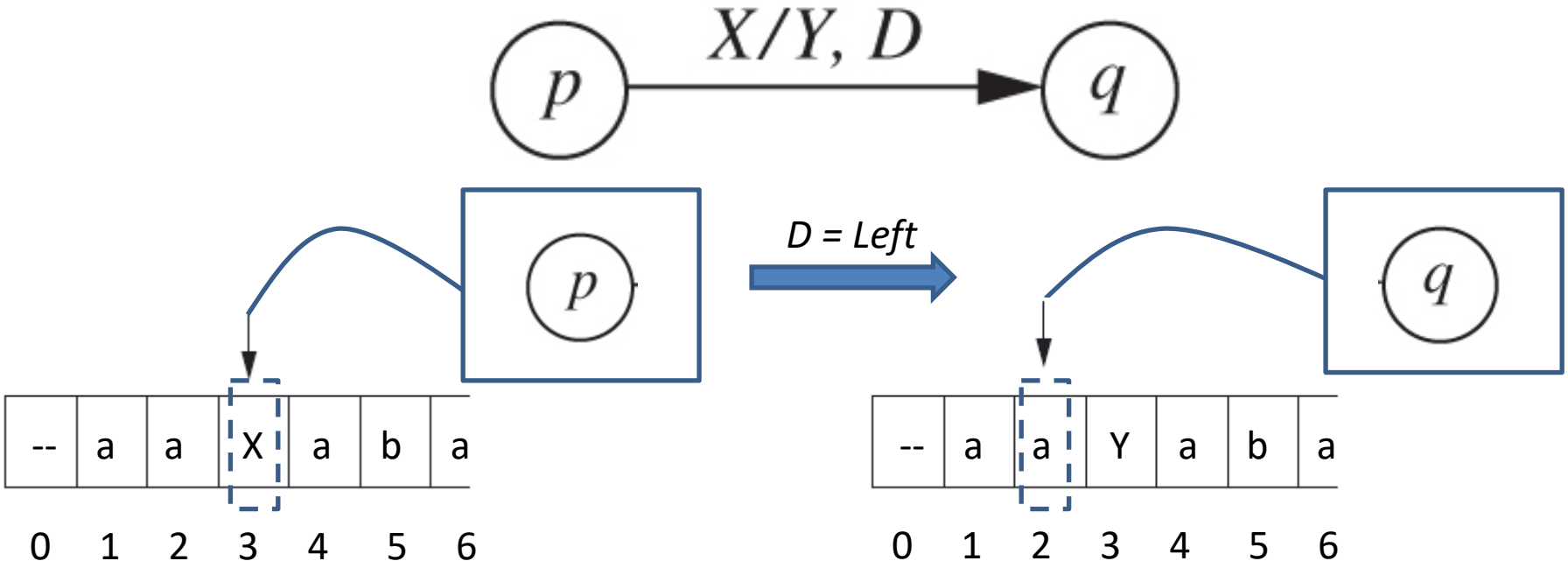
Output symbol

Right

Left

Stop

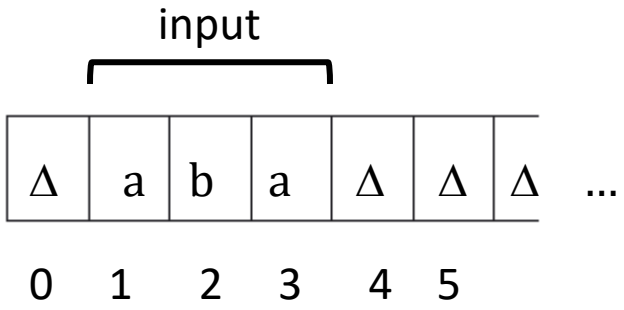
- We interpret $\delta(p, X) = (q, Y, D)$ to mean:
  - when T is in state p and
  - the symbol in the **current square** is X,
  - the TM replaces X by Y in that square,
  - changes to state q,
  - and moves the tape head one square to the right, or moves one square to the left, or doesn't move (D)



- We interpret $\delta(p, X) = (q, Y, D)$ to mean:
  - when T is in state p and
  - the symbol in the **current square** is X,
  - the TM replaces X by Y in that square,
  - changes to state q,
  - and moves the tape head one square to the right, or moves one square to the left, or doesn't move (D)

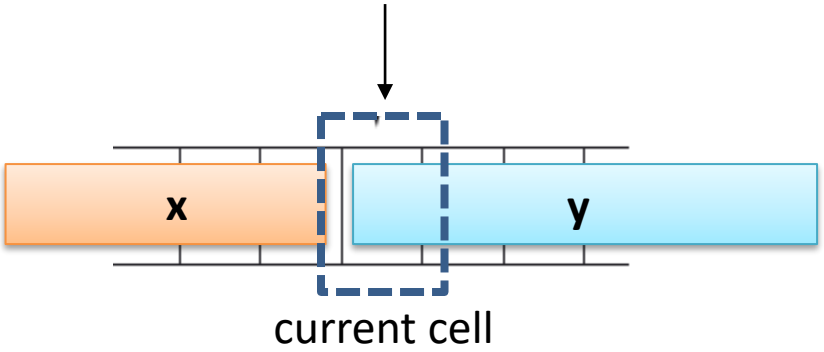
- If the state q is either h_a or h_r then T halts forever
- If T attempts to move left when it is on square 0, we will say that it halts in state h_r , leaving the tape head in square 0 and leaving the tape unchanged

- Normally a TM begins with an input string starting in square 1 and all other squares (square 0 and all the ones following the input string) blank



- In any case, the set of nonblank squares on the tape must always be finite

The current *configuration* of a TM is described by a single string *xqy*, where



- *q* is the current state
- *x* is the string of symbols to the left of the current cell (may be null)
- *y* starts in the current cell, includes everything to the right, and everything after *xy* on the tape is blank

xqy = *xqy* Δ = *xqy* Δ Δ = *xqy* Δ Δ Δ = … same configuration

If the head points to Δ, and all following cells are Δ then the configuration is written as *xq* Δ

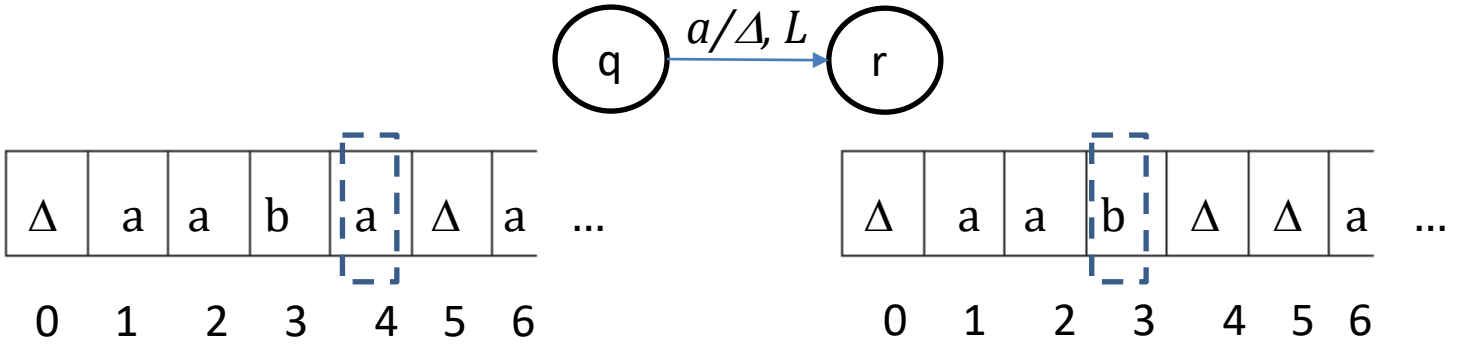
- We trace a sequence of moves by specifying the configuration at each step
- If q is a non-halting state and r is any state, we write

$$xqy \vdash_T zrw \quad \text{or} \quad xqy \vdash_T^* zrw$$

to mean that T moves from the first configuration to the second in one move, or in zero or more moves, respectively

Example: T is in nonhalting state q, and the configuration is $aabqa\,\Delta\,a$, and $\delta(q, a)=(r, \Delta, \mathrm{L})$ we could write

$$aabqa\,\Delta\,a \vdash_T aarb\,\Delta\,\Delta a$$



- The initial configuration corresponding to input x is given by $q_0\Delta x$ (if not defined otherwise)

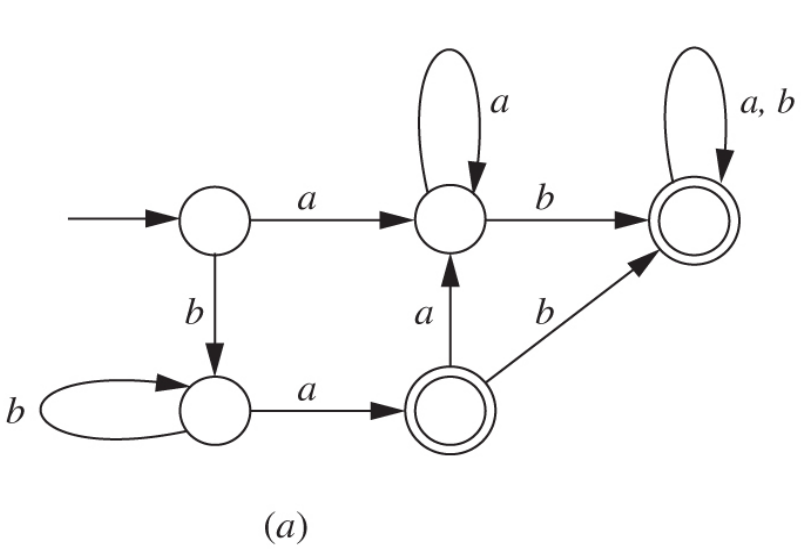
Turing Machines as Language Acceptors

- Definition: If $T = (Q, \Sigma, \Gamma, q_0, \delta)$ is a TM and $x \in \Sigma^*$, x is accepted by T if $q_0 \Delta x \vdash_T^* w h_a y$ for some $w, y \in (\Gamma \cup \{\Delta\})^*$

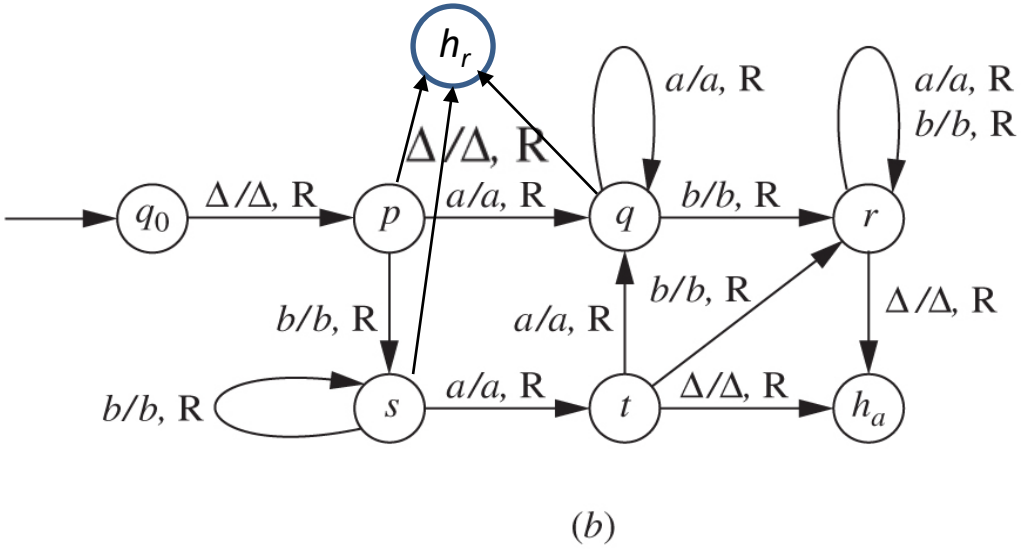
- A language $L \subseteq \Sigma^*$ is accepted by T if
$$L = L(T) = \{x \in \Sigma^* \mid x \text{ is accepted by } T\}$$

TM simulating FA

- The following transition diagrams show an FA and a TM that accept the same language



(a)



(b)

$L = \{a,b\}^*\{ab\}\{a,b\}^* \cup \{a,b\}^*\{ba\}$

Example: TM for Palindromes

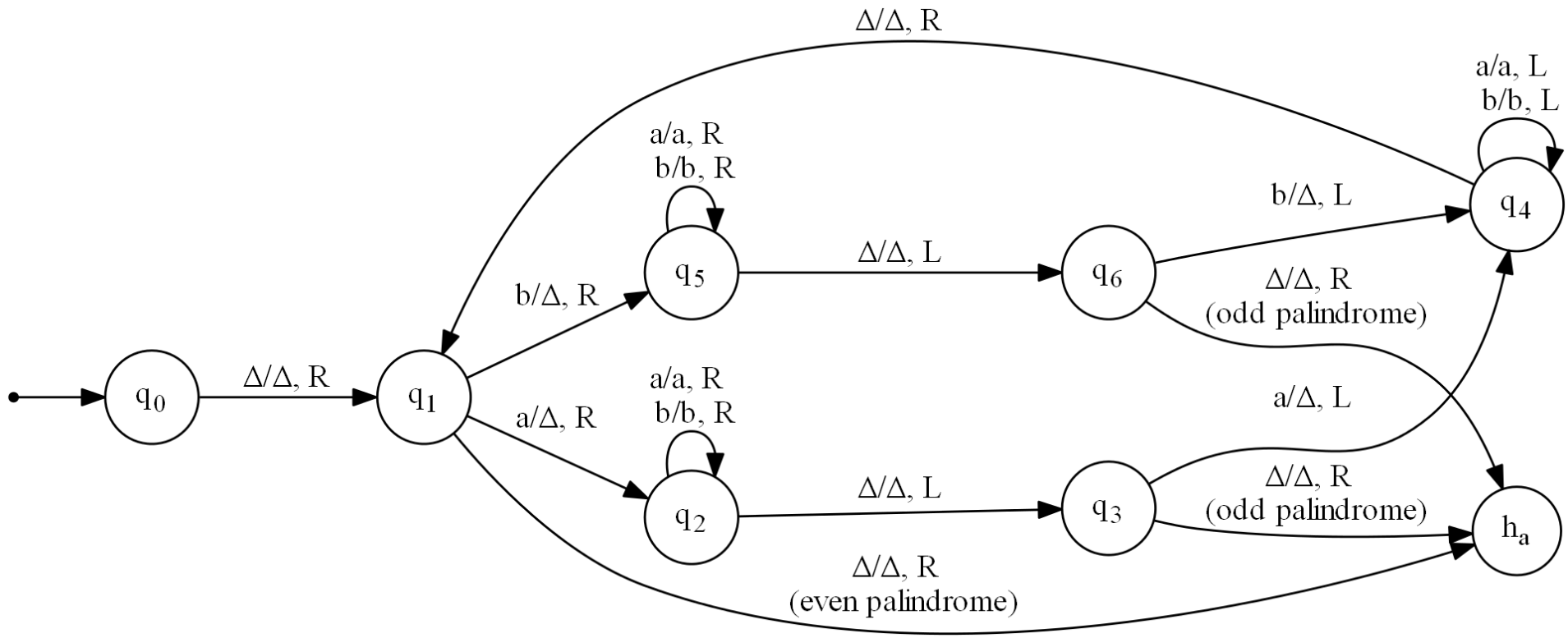
- We don't simulate non-deterministic PDA for Pal (however, there exist non-deterministic TMs).
- Basic idea:
  - Compare the first σ_1 with the last σ_n.
  - If they match, compare the σ_2 with σ_{n-1} and so on …
  - For a palindrome, we will end up with 0 or 1 unmatched letter
- Comparison:

  1) read σ and replace it with Δ;

  2) move across the tape to first Δ;

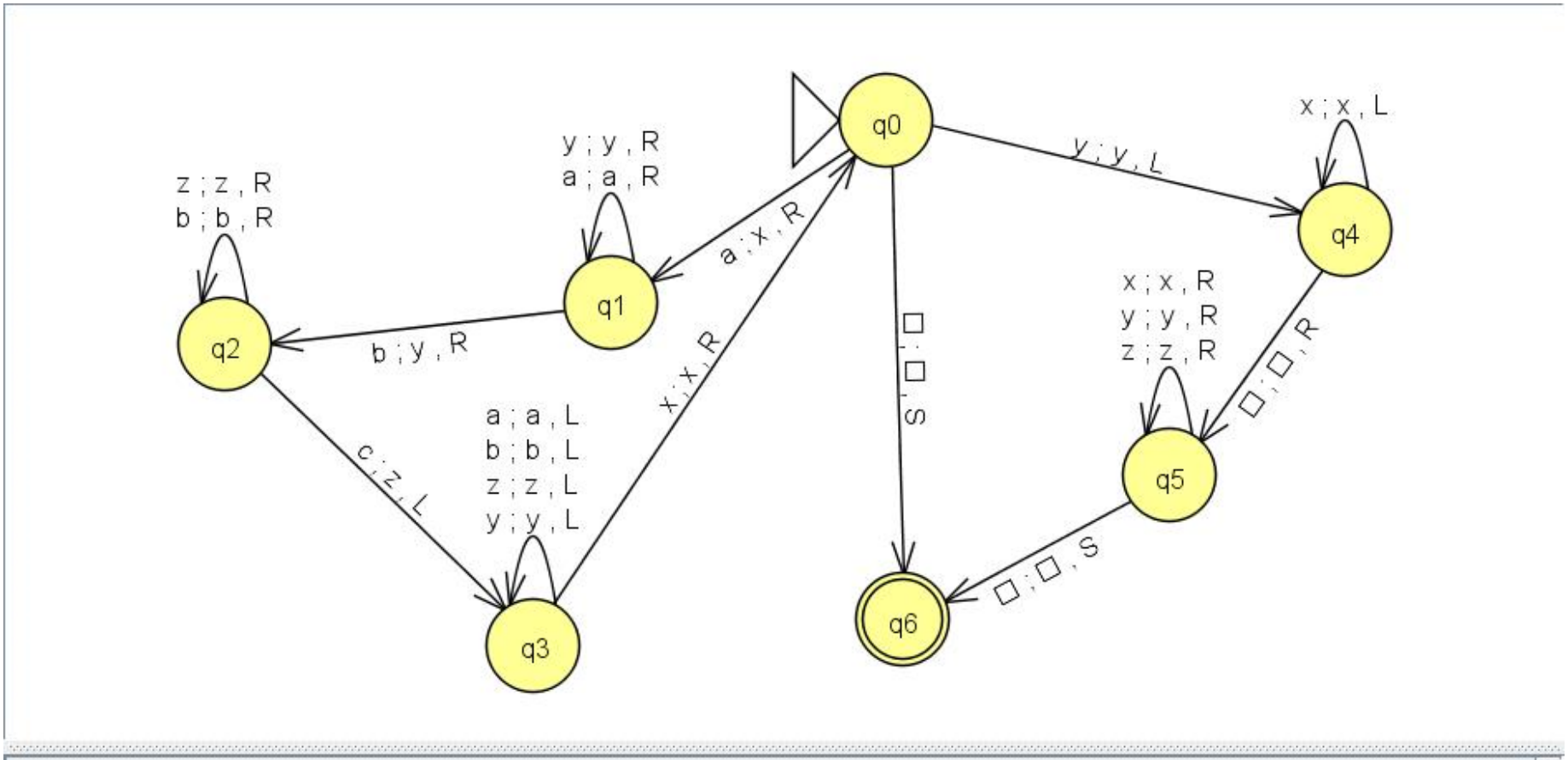
  3) check the character to the left and replace it with Δ if it matches.

- Halting condition: If when you moved left/right after finding the first blank, the character found is a blank, the string is a palindrome.



Add h_r and transitions to it from q_3 and q_6

Demo of AnBnCn in JFLAP

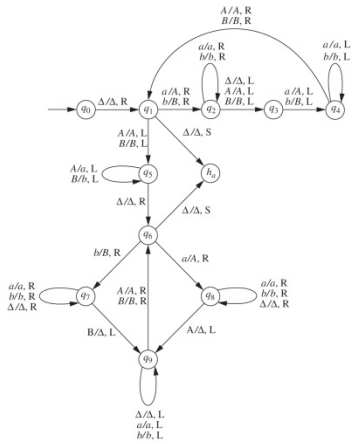


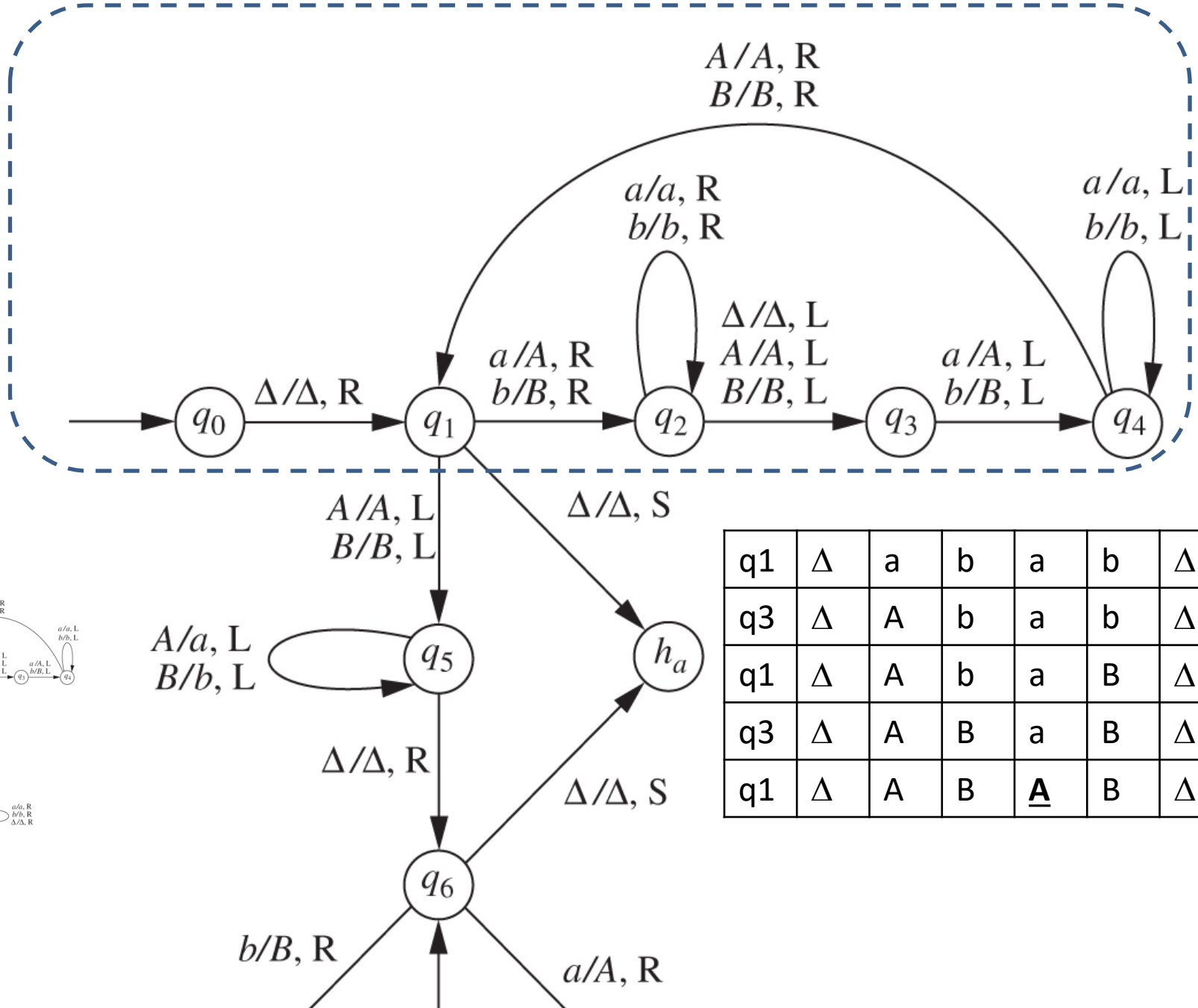
Example: TM for L={xx} (not CFL)

- Basic idea:
  - Find and mark the middle of the string
  - Compare characters starting from the start of the string with characters starting from the middle of the string.

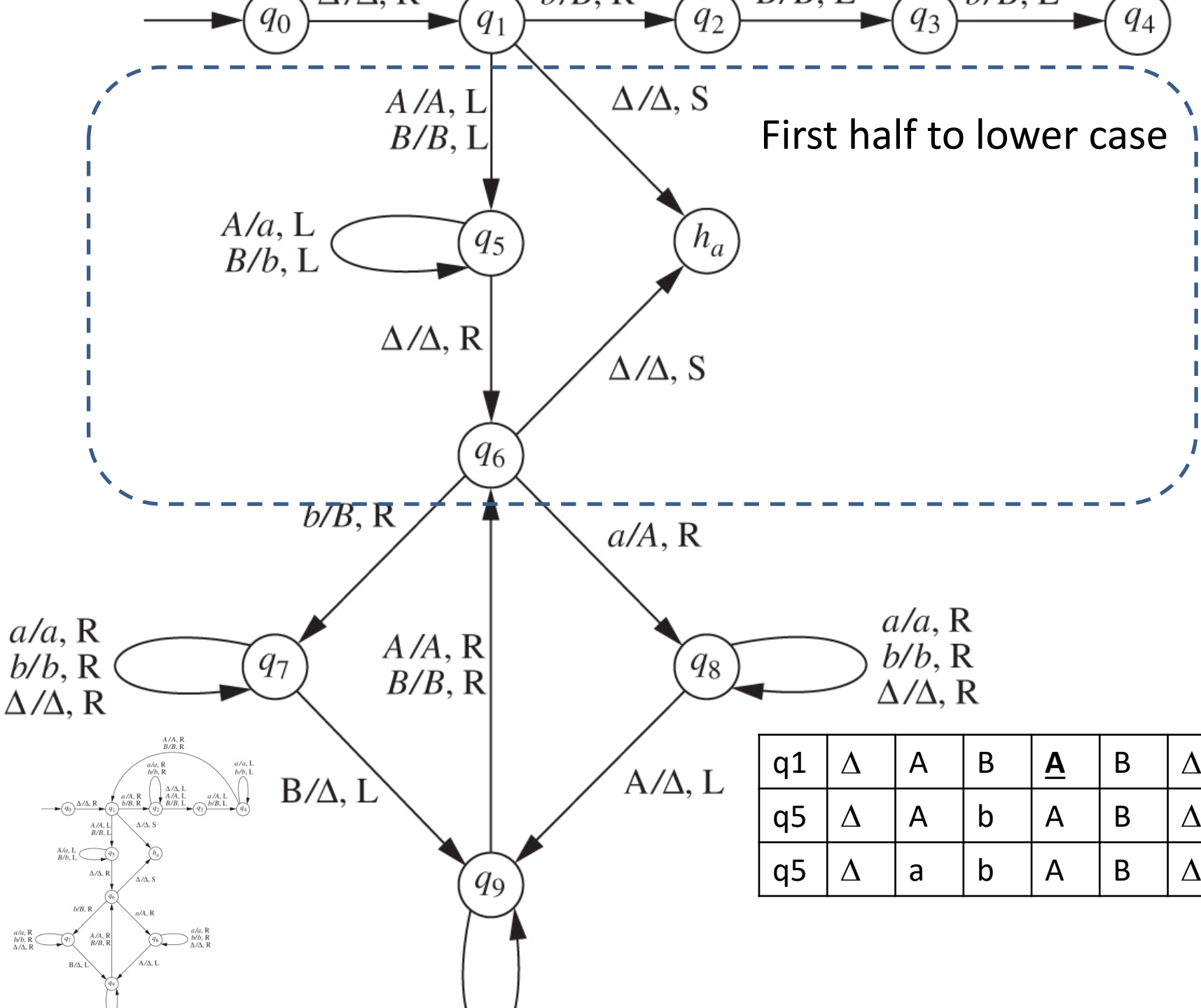
- Finding the middle: similar to palindromes but now *replace lower case letters with the corresponding upper case letters*
- Once we have found the middle,
  - convert the 1st half of the string to lower case letters, and
  - match lower case letters (1st half of xx) to the upper case letters (2nd half of xx).
- Matching will be done by replacing letters with blanks.

Find middle of string

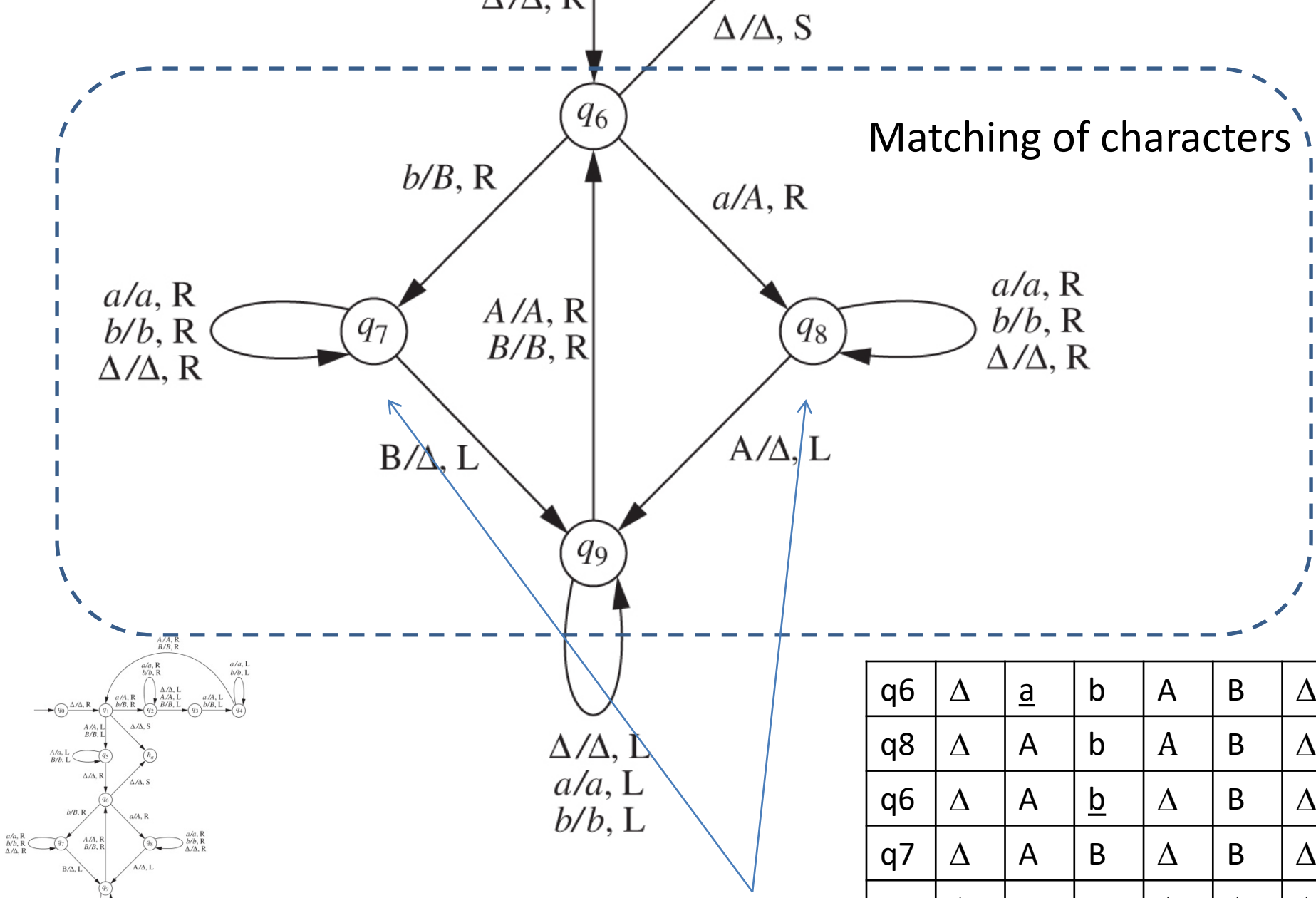




q1	Δ	a	b	a	b	Δ
q3	Δ	A	b	a	b	Δ
q1	Δ	A	b	a	B	Δ
q3	Δ	A	B	a	B	Δ
q1	Δ	A	B	**A**	B	Δ

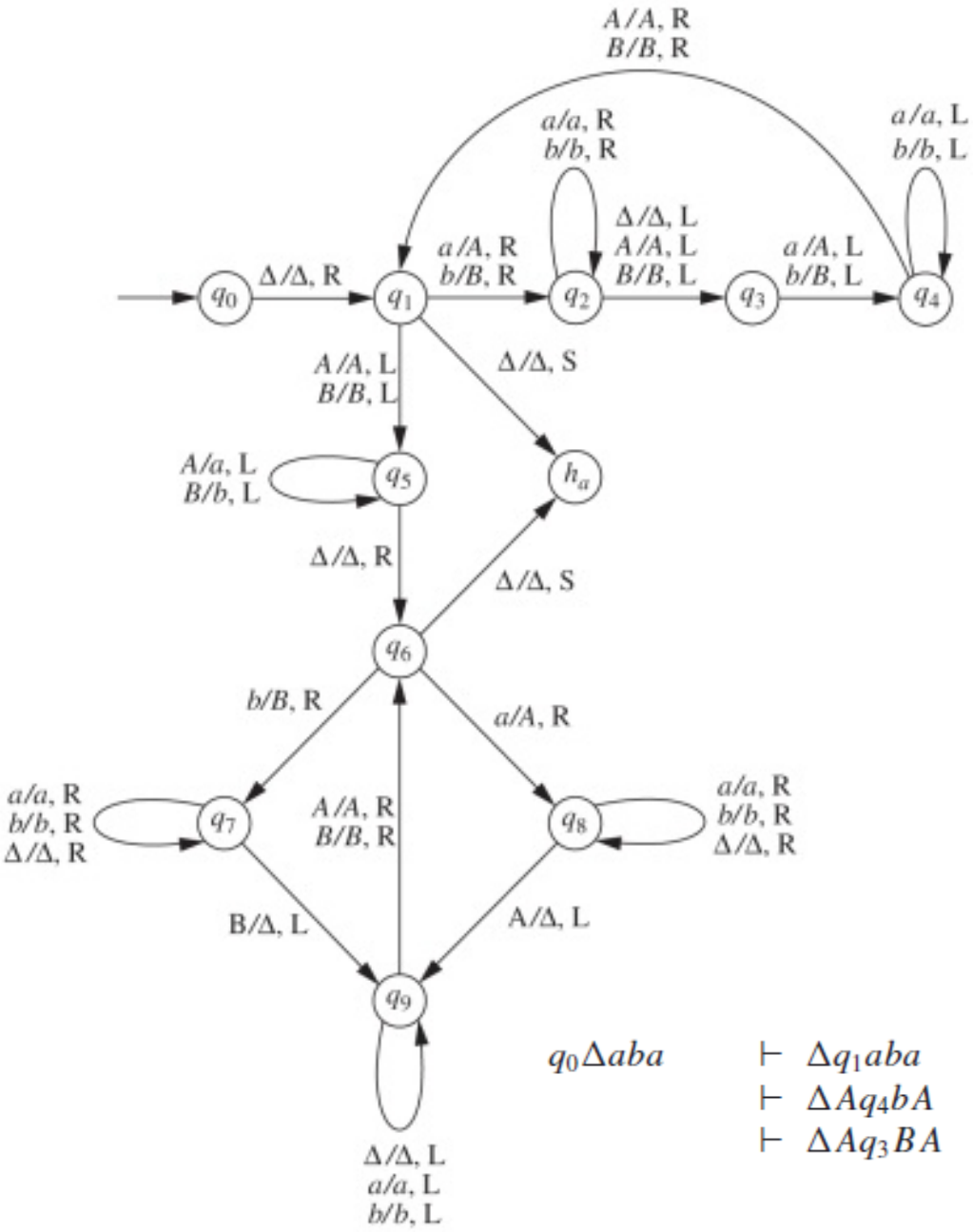


q1	Δ	A	B	**A**	B	Δ
q5	Δ	A	b	A	B	Δ
q5	Δ	a	b	A	B	Δ



add h_r for the opposite capital letters

q6	Δ	a	b	A	B	Δ
q8	Δ	A	b	A	B	Δ
q6	Δ	A	b	Δ	B	Δ
q7	Δ	A	B	Δ	B	Δ
	Δ	A	B	Δ	Δ	Δ



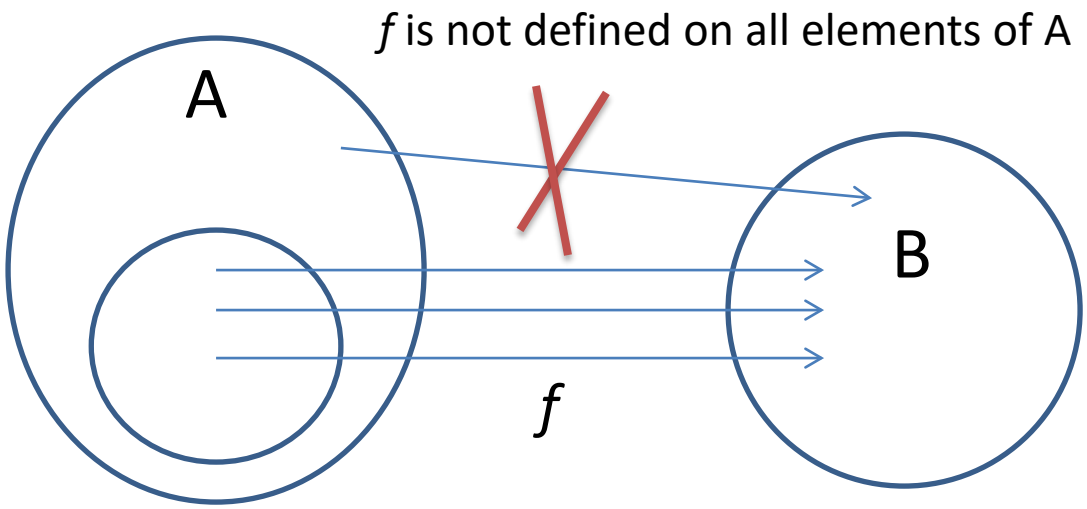
$q_0\Delta aba \vdash \Delta q_1 aba \vdash \Delta A q_2 ba \vdash^* \Delta A b a q_2 \Delta \vdash \Delta A b q_3 a$

$\vdash \Delta A q_4 b A \vdash \Delta q_4 A b A \vdash \Delta A q_1 b A \vdash \Delta A B q_2 A$

$\vdash \Delta A q_3 B A \vdash \Delta A h_r B A$ (reject)

Turing Machines that Compute Partial Functions

- A Turing machine that produces an output string for every legal input string is said to compute a partial function on Σ^*



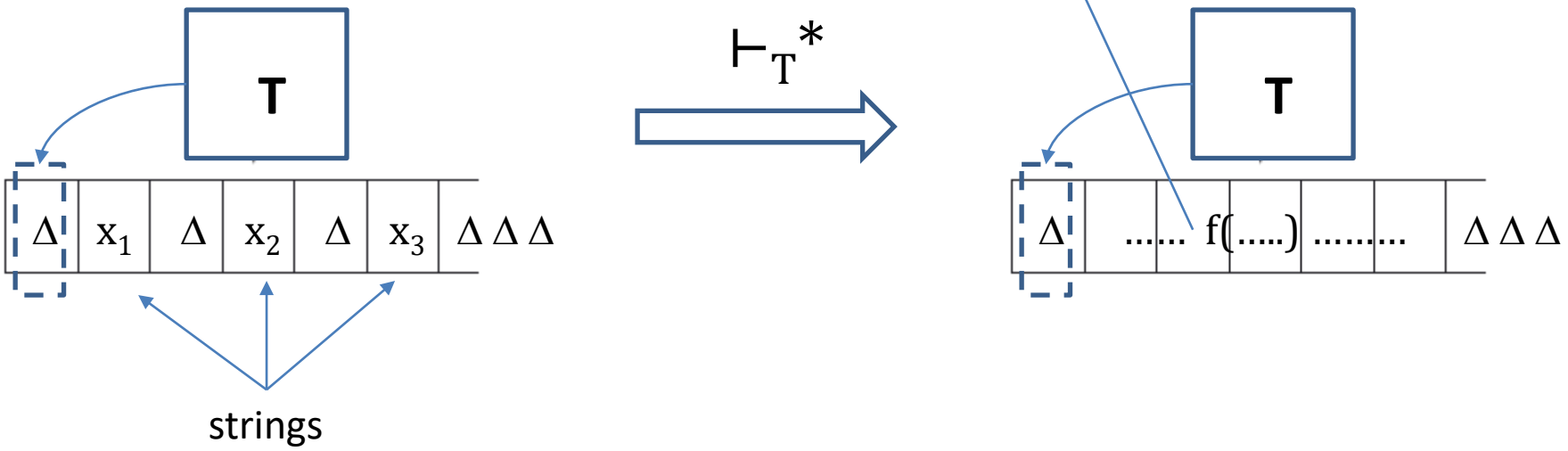
Turing Machines that Compute Partial Functions

- A Turing machine that produces an output string for every legal input string is said to compute a partial function on Σ^*
- We consider TMs that compute partial functions on $(\Sigma^*)^k$, i.e., functions of k variables
- The most important issue is what output strings are produced for input strings in the domain of f
- However, we want the TM to accept only inputs in the domain of f, in order to be able to say that it computes f and not some other function with larger domain

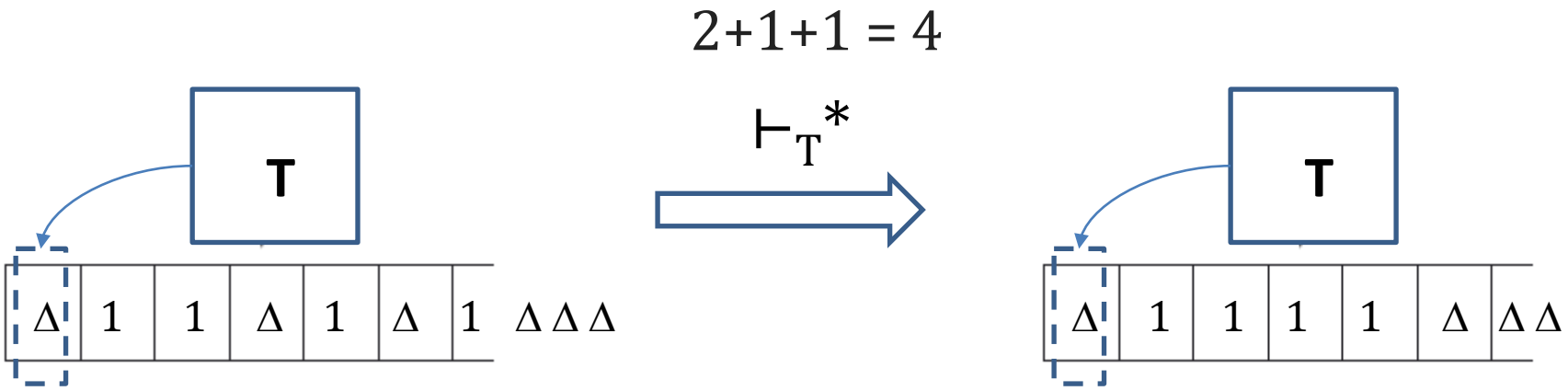
- Definition:
  - Let $T = (Q, \Sigma, \Gamma, q_0, \delta)$ be a Turing machine, k a natural number, and f a partial function from $(\Sigma^*)^k$ to Γ^*
  - We say that T computes f if for every $(x_1, x_2, \ldots, x_k)$ in the domain of f,

    $$q_0 \Delta x_1 \Delta x_2 \Delta \ldots \Delta x_k \vdash_T^* h_a \Delta f(x_1, x_2, \ldots x_k)$$

    and no other input that is a k-tuple of strings is accepted by T
- A partial function f is Turing-computable if there is a TM that computes f

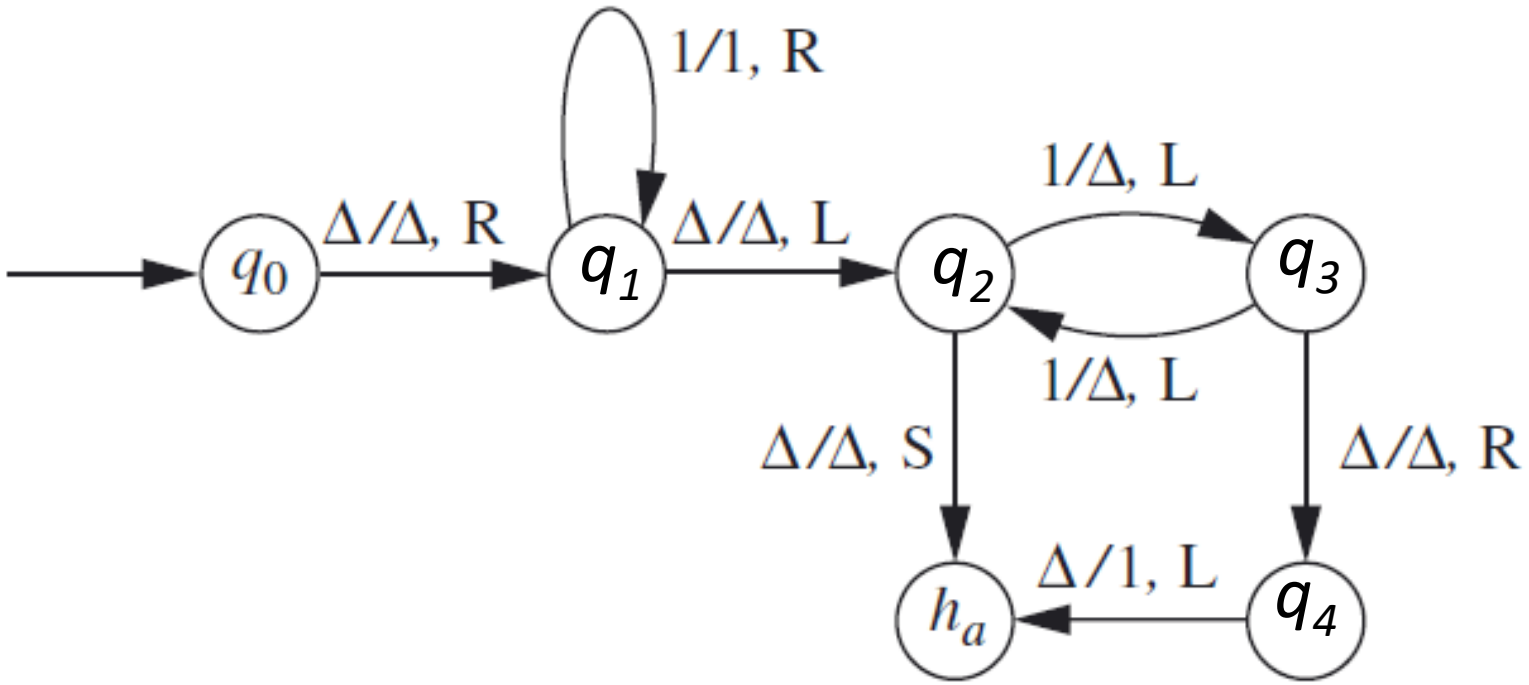


- For our purposes, it will be sufficient to consider partial functions on $\mathbb{N}^k$ with values in $\mathbb{N}$
- We will use unary notation for numbers
- The official definition is similar to the previous definition, except that the input alphabet is {1}, and the initial configuration looks like $q_0\Delta 1^{n_1}\Delta 1^{n_2}\Delta\ldots\Delta 1^{n_k}$
- Example: f(x,y,z) = x+y+z



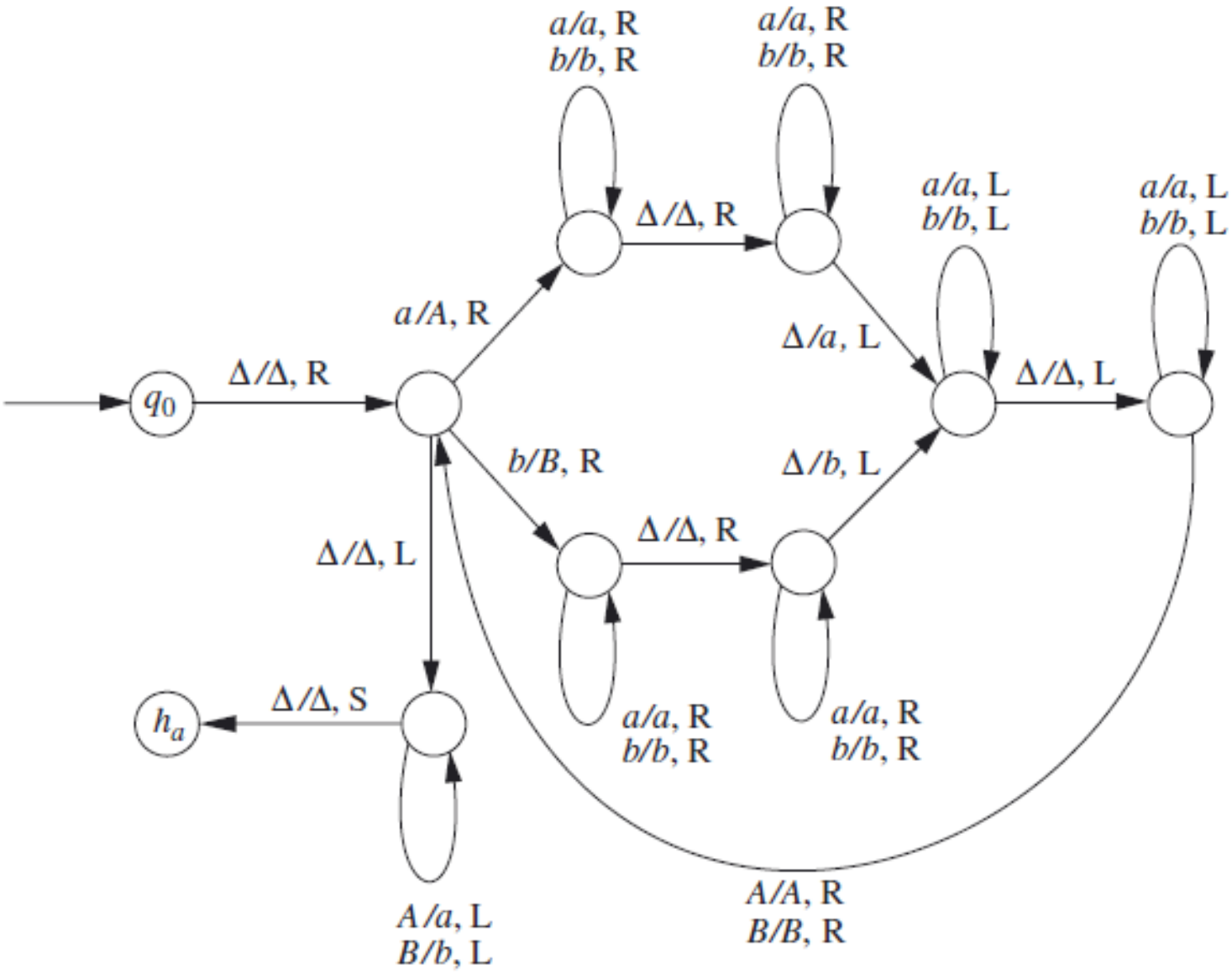
Example: TM for computing the remainder Mod 2

Moves the tape head to the end of the string, then makes a pass from right to left in which the 1's are counted and simultaneously erased. The final output is a single 1 if the input was odd and nothing otherwise.

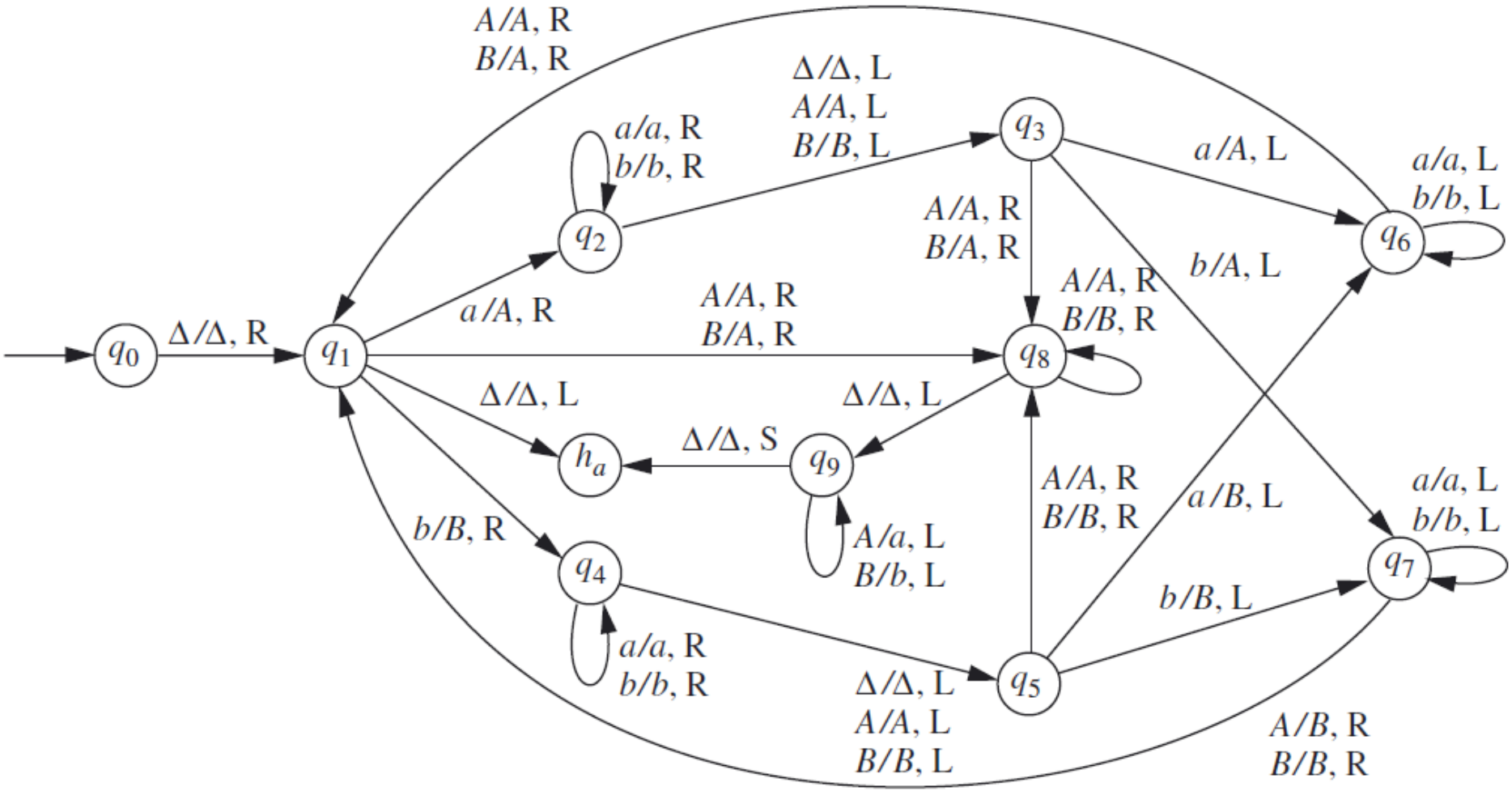


Example: TM for copying a string

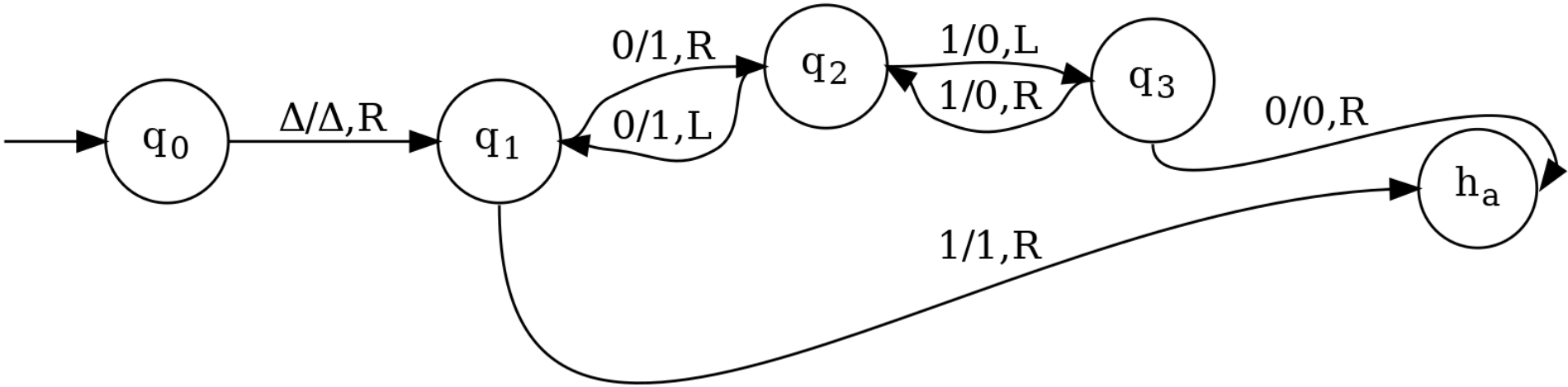
Δ̲abΔΔ
Δa̲bΔΔ
ΔAb̲ΔΔ
ΔAbΔ̲Δ
ΔAbΔΔ̲
ΔAbΔ̲a
ΔAb̲Δa
…



Example: TM for computing reverse function



Example: TM that never halts on input 01

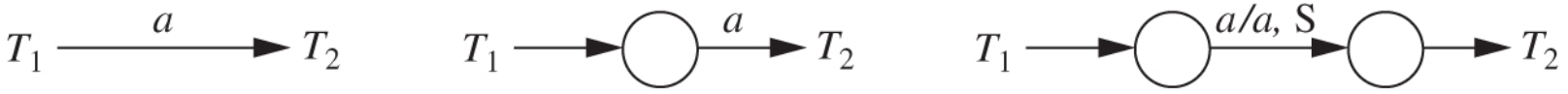


Combining Turing Machines

- Just as a large algorithm can be described as a number of subalgorithms working in combination, we can combine several Turing machines into a larger composite TM
- If T_1 and T_2 are TMs, we can consider the composition T_1T_2: "first execute T_1, then execute T_2 on the result"
  - The set of states of T_1T_2 is the union of the sets of states of T_1 and T_2 (relabeled if necessary)
  - The initial state is the initial state of T_1

- The transitions of $T_1 T_2$ include all of those of T_2 and all of those of T_1 that don't go to h_a
- A transition in T_1 that goes to h_a is replaced by a similar transition that goes to the start state of T_2
- The output of T_1 must be a valid input configuration for T_2
- We may use transition diagrams containing notations such as $T_1 \rightarrow T_2$, in order to avoid showing all the states.

- We might use any of the above notations to mean "in state *p*, if the current symbol is *a*, then execute *T*"
- We might use any of the following to mean "execute T_1, and if T_1 halts in h_a with current symbol *a*, then execute T_2"



Example: Accepting the language of palindromes

A *Copy* TM starts with tape $\underline{\Delta}x$, where x is a string of nonblank symbols, and ends up with $\underline{\Delta}x\Delta x$.

NB (PB) denotes the Turing machine that moves the tape head to the *next (previous) blank*

Copy→NB → R → PB → Equal

R denotes the Turing machine that computes the reverse of a string

Equal denotes the Turing machine that starts with $\underline{\Delta}x\Delta y$ and determines whether $x=y$.

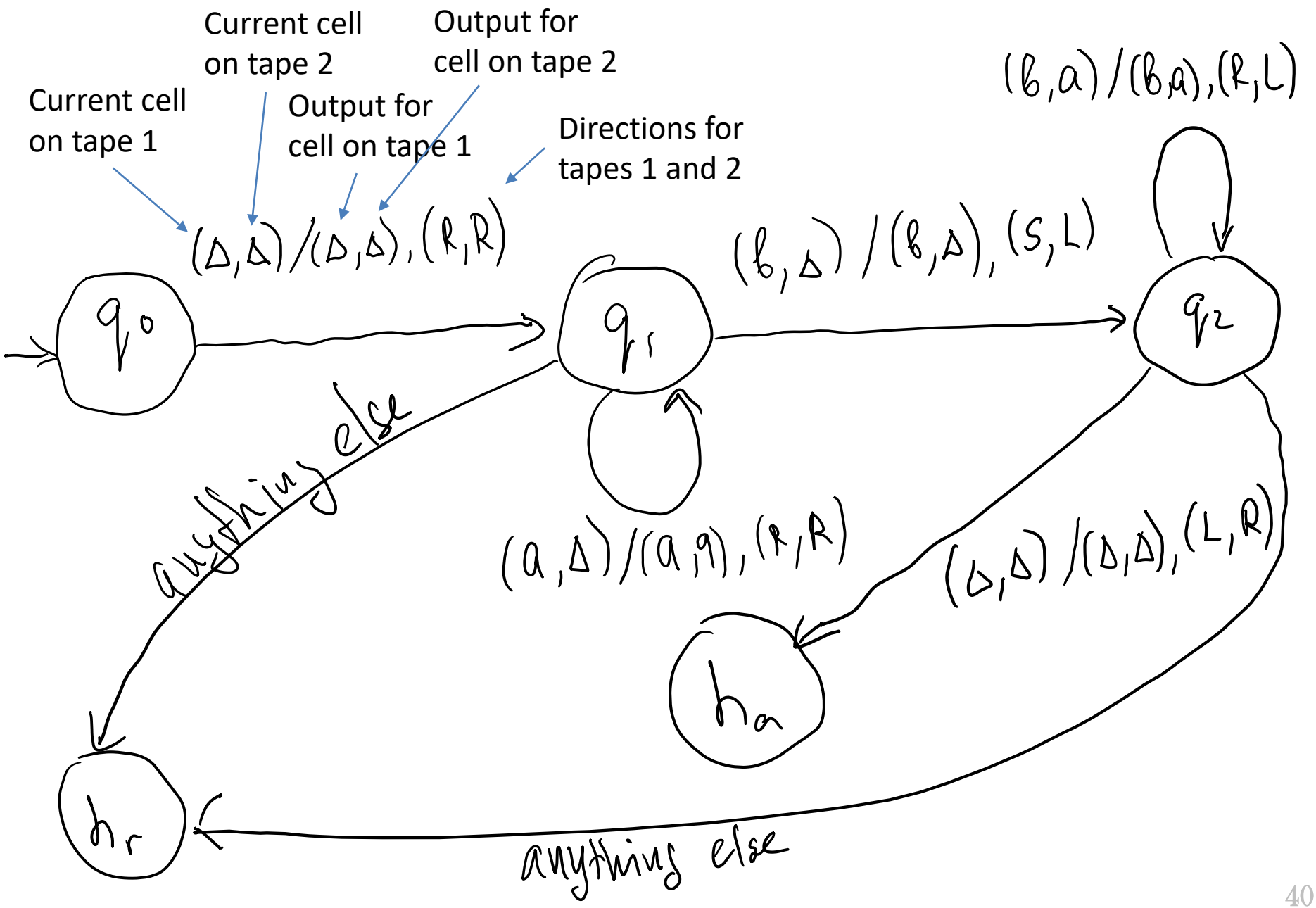
Multitape Turing Machines

- Some algorithms can be unwieldy to implement on a TM, because of the bookkeeping necessary
- A way of simplifying them is to have multiple tapes with independent heads
- This is a different model of computation, a multitape TM

Multitape Turing Machines

- A 2-tape TM can also be described by a 5-tuple $T = (Q, \Sigma, \Gamma, q_0, \delta)$, where this time
  $\delta : Q \times (\Gamma \cup \{\Delta\})^2 \rightarrow (Q \cup \{h_a, h_r\}) \times (\Gamma \cup \{\Delta\})^2 \times \{\mathrm{R}, \mathrm{L}, \mathrm{S}\}^2$
- A single move can change the state, the symbols in the current squares on both tapes, and the positions of the two tape heads
- We will represent a configuration by a 3-tuple $(q, x_1 \underline{a_1} y_1, x_2 \underline{a_2} y_2)$ where q is the current state, $x_i a_i y_i$ is the contents of tape i, and a_i is in the current square of tape i
- The input configuration for input x will be $(q_0, \underline{\Delta}\, x, \underline{\Delta})$

Example of two tape TM for $AnBn^+ = \{a^n b^n \mid n \geq 1\}$



Multitape Turing Machines

- It turns out that that, just as nondeterminism and Λ-transitions do not increase the power of FAs, allowing a Turing machine multiple tapes does not increase its power
- We can prove that for every multitape TM *T* there is a single-tape TM that accepts exactly the same strings as *T*, rejects the same strings, and produces exactly the same output for every input string it accepts
- To simplify, we will only consider two-tape machines
- We need to simulate 2 tapes using 1 tape only. This can be done by enumerating the cells and using even squares for tape 2, and odd for tape 1.
- Another approach: use a separator symbol to concatenate the contents of two tapes on one tape.

The Church-Turing Thesis

- To say that the TM is a general model of computation implies that any algorithmic procedure that can be carried out at all, by a human computer or a team of humans or an electronic computer, can be carried out by a TM
  - The statement was formulated by Alonzo Church in the 30's
  - It is referred to as Church's thesis or the Church-Turing thesis
  - **It is not a statement that can be proved (i.e., you cannot use it to prove something else)**, but there is a lot of evidence for it

The Church-Turing Thesis

- The nature of the model makes it seem that a TM can execute any algorithm a human can
- Enhancements to the TM (stacks, queues, multi-tape, multi-head, etc.) have been shown not to increase its power
- Other theoretical models (even those that are faster) of computation proposed have been shown to be equivalent to a TM
- No one has ever suggested any kind of computation that cannot be implemented on a TM
- **From now on, we will consider that by definition, an "algorithmic procedure" is what a TM can do**

Nondeterministic Turing Machines

- We can add nondeterminism to Turing machines: as usual, $\delta(q, a)$ becomes a subset, not an element

- **Theorem**: For every nondeterministic TM (NTM) T there is an ordinary (deterministic) TM T_1 with $L(T_1) = L(T)$

- Proof: The idea is to use an algorithm that can test, if necessary, every possible sequence of moves of T on an input string x

For example, we can simulate T by scanning all its possible steps using the tree of all possible configurations, i.e., the nodes are configurations, the children of a node are all possible steps of T.

T accepts the input iff there is a finite branch from its root to an accepting configuration.

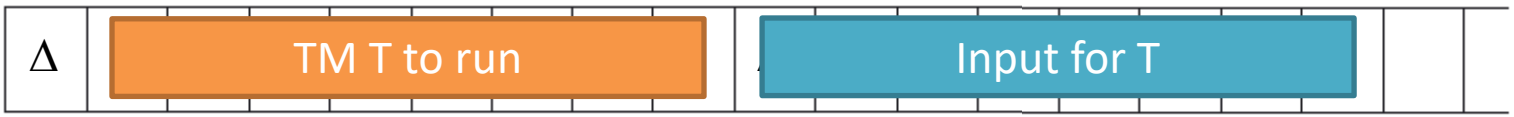
In the simulation use BFS (not DFS!). Running time explodes but this simulator will reach all accepting configurations.

Equivalence of models

- In order to show that two TM models (e.g., ordinary deterministic TM T and double-head deterministic TM DHT) are equivalent, you need to prove that each model can be simulated with another (e.g., given T for some problem, you can design DHT for the same problem and vice versa).
- Same idea works if you need to show equivalence of any models
- You cannot use Church-Turing thesis to say that these models are equivalent.

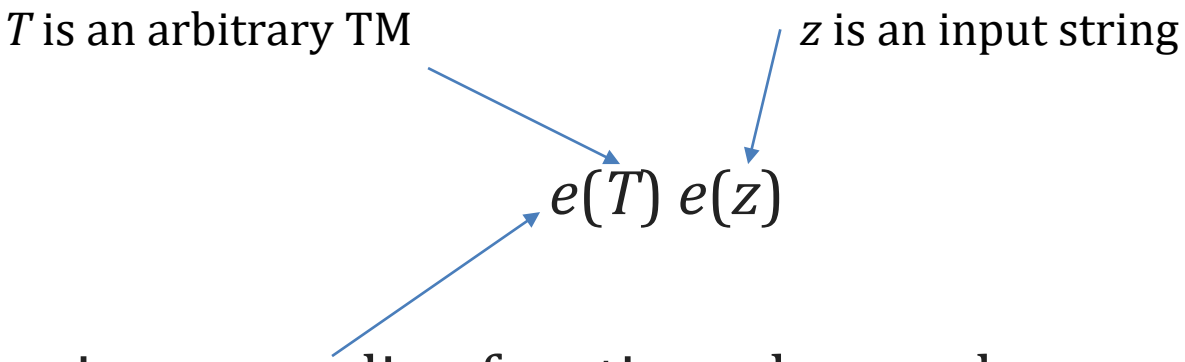
Universal Turing Machines

- The TMs we have studied so far have been special-purpose computers capable of executing a single algorithm
- We need a "universal" Turing machine, which can execute a program stored in its memory
  - It receives an input string that specifies
  1. the algorithm it is to execute, i.e., another TM, and
  2. the input that is to be provided to the algorithm



Universal Turing Machines

- **Definition**: A *universal* Turing machine is a Turing machine T_u that works as follows
  - It is assumed to receive an input string of the form



  e is an encoding function whose values are strings in {0,1}*
  - The computation performed by T_u on this input string satisfies these two properties
    - T_u accepts $e(T)e(z)$ **iff** T accepts z
    - If T accepts z and produces output y, then T_u produces output $e(y)$

Universal Turing Machines

- We discuss a simple encoding function e, and then sketch one approach to constructing a universal TM
- What are the crucial features of the encoding function:
  - It is possible to decide algorithmically, for an arbitrary string $w \in \{0,1\}^*$, whether w is a legitimate value of e
  - A string w should represent at most one TM, or at most one string
  - There should be an algorithm for decoding strings of the form $e(T)$ or $e(z)$ and reconstructing the TM or string it represents

Universal Turing Machines: Encoding

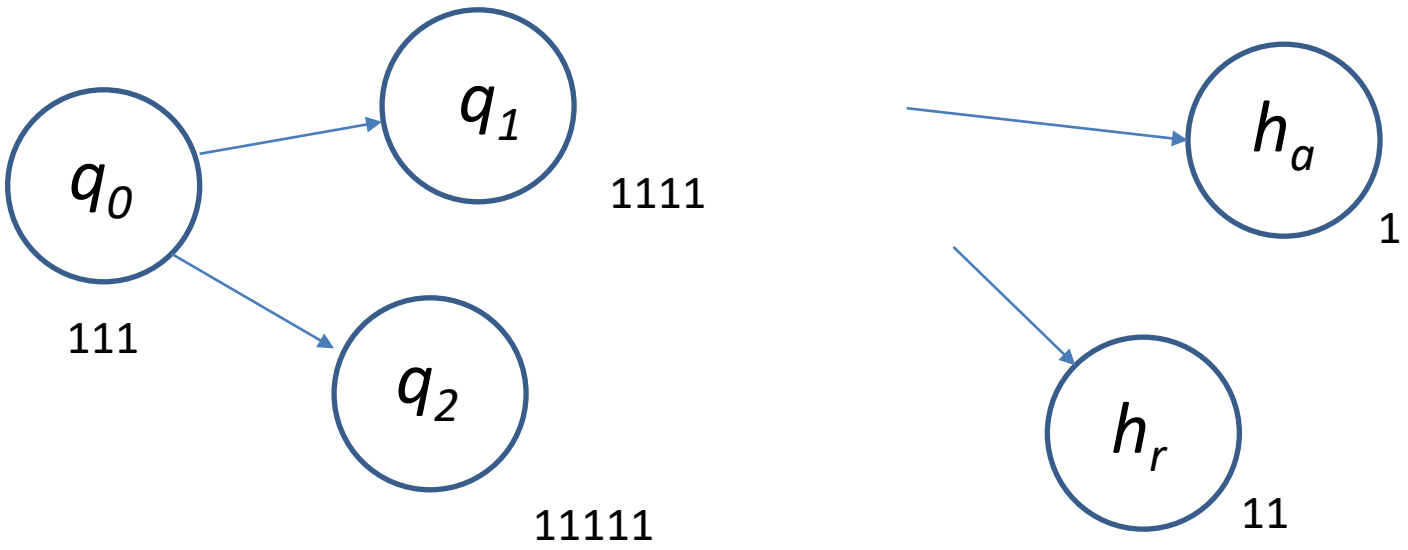
- State labels will be replaced by numbers, and we will base the encoding on these numbers
- We also enumerate all possible symbols including blank
- The idea of the encoding is to represent a TM as a set of moves and each move is associated with a 5-tuple of numbers
  - Each number is in unary representation followed by a 0

Universal Turing Machines

- **Definition**: If $T=(Q, \Sigma, \Gamma, q_0, \delta)$ is a TM and z is a string, define the strings $e(T)$ and $e(z)$ as follows
  - First assign numbers to each state, tape symbol, and tape head direction of T; $n(h_a) = 1$, $n(h_r) = 2$, and $n(q_0) = 3$
  - The other elements of Q get distinct numbers ≥ 4
  - $n(\mathrm{R}) = 1$, $n(\mathrm{L}) = 2$, $n(\mathrm{S}) = 3$

Right: 1
Left: 11
Stop: 111

Δ : 1
a: 11
b: 111
c: 1111



Universal Turing Machines

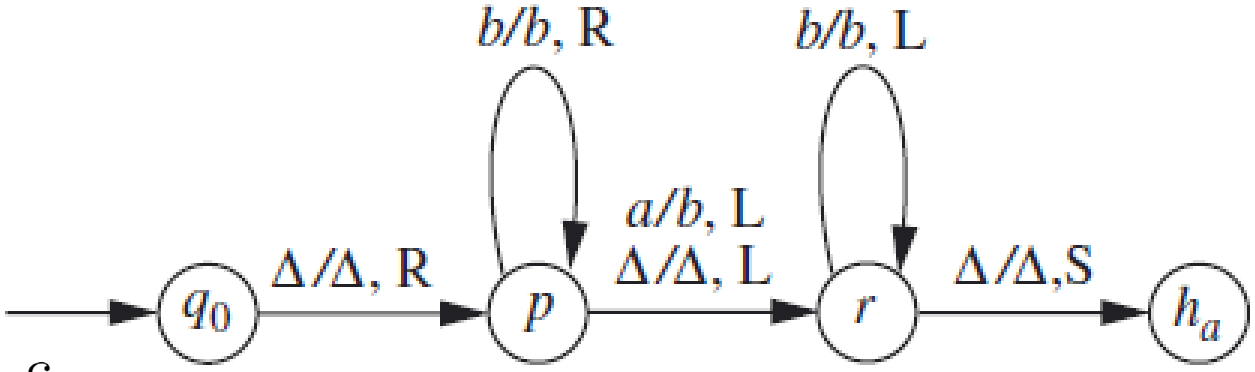
- For each move m of the form $\delta(p, \sigma) = (q, \tau, D)$,
  $e(m) = 1^{n(p)}01^{n(\sigma)}01^{n(q)}01^{n(\tau)}01^{n(D)}0$

- List the moves of T as $m_1, \ldots, m_k$ (the order is arbitrary), and let
  $e(T) = e(m_1)0e(m_2)0\ldots0e(m_k)0$

- If $z = z_1 z_2 \ldots z_j$ is a string over Γ^*, then
  $e(z) = 0\ 1^{n(z_1)}\ 0\ 1^{n(z2)}\ 0\ \ldots\ 0\ 1^{n(z_j)}0$

So, $e(T)\ e(z) = e(m_1)0e(m_2)0\ldots0e(m_k)0\ 0\ 1^{n(z_1)}\ 0\ 1^{n(z2)}\ 0\ \ldots\ 0\ 1^{n(z_j)}0$

Example of encoding



We define

- $n(h_a) = 1$, $n(q_0) = 3$, $n(p) = 4$, $n(r) = 5$
- $n(a) = 2$, and $n(b) = 3$
- $n(R) = 1$, $n(L) = 2$, $n(S) = 3$.

If m is the move with $\delta(q_0, \Delta) = (p, \Delta, R)$, then
$e(m) = 1^3 0 1 0 1^4 0 1 0 1 0 = 1110101111101010$.

1111

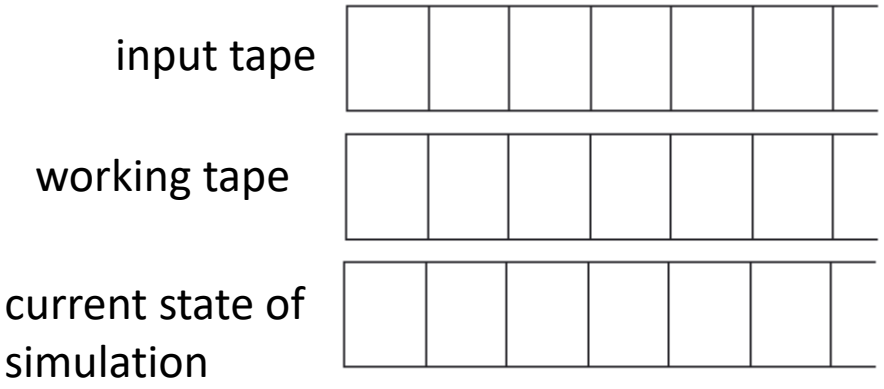
$e(T) = 1110101111010100 \quad 1111011101111011101000 \quad 1111011011111011101100$

$1111010111110101100 \quad 1111011101111011101100 \quad 1111010101011100$

- Theorem: Let $E = \{e(T) \mid T \text{ is a TM}\}$
  - Then for every $x \in \{0,1\}^*$, $x \in E$ if and only if :
    - x matches the regular expression $(11^*0)^50((11^*0)^50)^*$, so that it is a sequence of 5-tuples
    - No two substrings of x representing 5-tuples have the same first two parts (no move can appear twice)
    - None of the 5-tuples have first part 1 or 11 (no moves from halting states)
    - The last part of each 5-tuple must be 1, 11, or 111 (it must represent a direction)

- Those conditions don't guarantee that the string represents a TM that carries out a meaningful computation
  - But they do ensure that we can draw a transition diagram corresponding to the encoded TM

- Testing a string to determine whether it satisfies these conditions is straightforward, so we have verified that *e* satisfies the minimal requirements for such a function
- The simplest idea for a universal TM is to have 3 tapes: the first for the input, the second is the working tape, and the third remembers the state the input TM is currently simulated.



- **Homework: read the chapter about universal TM, and the details of simulation.**