Languages and Computation (COMP2049/AE2LAC)

Turing Machine

Dr. Tianxiang Cui

tianxiang.cui@nottingham.edu.cn

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:
  - A finite automaton cannot accept the language $L_1 = \{a^n b^n \mid n \geq 0\}$
  - A pushdown automaton cannot accept the language $L_2 = \{a^n b^n c^n \mid n \geq 0\}$
- We need something more powerful – a model of general-purpose computation

Turing Machine

- The abstract model we will study instead is the Turing Machine (TM)
  - It is not obtained by adding data structures onto a finite automaton
  - Rather, it predates the FA and PDA models
- A TM is a mathematical model of a general-purpose computer
- Can be considered as the generalization of a PDA that uses a tape instead of a stack
- Mainly used to study the notion of computation
  - What (exactly) can computers do (given sufficient time and memory) and what can they not do
- There are other notions of computation
  - λ-calculus, μ-recursive functions
  - They can all shown to be equivalent

Church–Turing Thesis

- A hypothesis about the nature of computable functions
- Every function which would naturally be regarded as "**computable**" **can be computed by a TM**
  - Any real-world computation can be translated into an equivalent computation involving a TM
- According to the Church-Turing thesis, TM is a general model of computation, potentially able to execute any algorithm

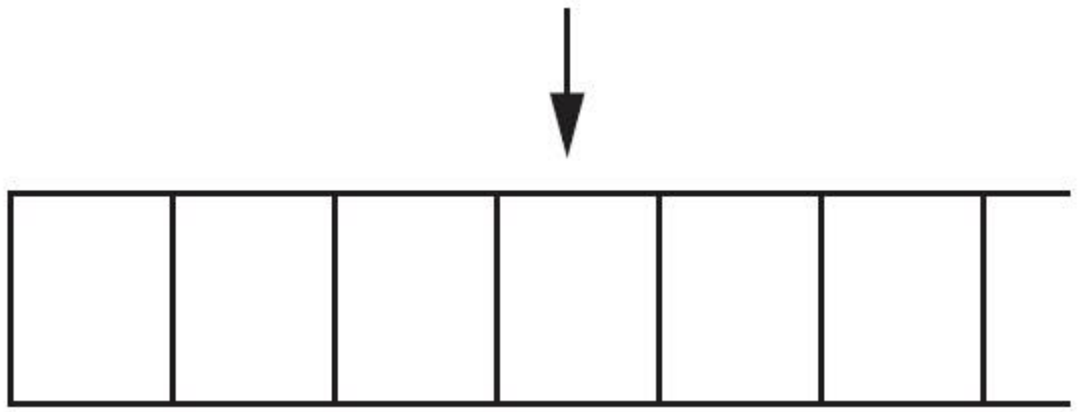
Human Computation

- To formulate the computational model, Turing considered a human being working with a pencil and paper
  - *The human computer could be assumed to operate under simple rules: First, the only things written on the paper are symbols from some fixed finite alphabet; second, each step taken by the computer depends only on the symbol he is currently examining and on his "state of mind" at the time; third, although his state of mind may change as a result of his examining different symbols, only a finite number of distinct states of mind are possible*
- 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

Turing Machine

- 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 of which holds one symbol
  - We will sometimes assign consecutive numbers to 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



Turing Machine

- In our version of a TM, a single move is determined by the **current state** (corresponding to the "state of mind" of the human computer) and the **current tape symbol** and has three parts:

1. **Changing** from the current state to another state
2. **Replacing** the symbol in the square by another
3. **Leaving** the tape head where it is, or **moving** it one square to the left, or **moving** it one square to the right

Turing Machine

- The tape may serve as:

1. The **input** device (the input string is assumed to be on the tape initially)
2. The **memory** available for use during the computation
3. The **output** device (the output, if it is relevant, is the string of symbols left on the tape at the end of the computation

- A Turing machine will have two **halt** states, one denoting **acceptance** and the other **rejection**
- If a TM decides to **accept** or **reject** the input string, it **stops**. But it might not decide to do this, and so it might continue **moving forever**
  - Different from FA or PDA

TM Formal Definition

- A Turing Machine is a 7-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, \Delta, F)$, where:

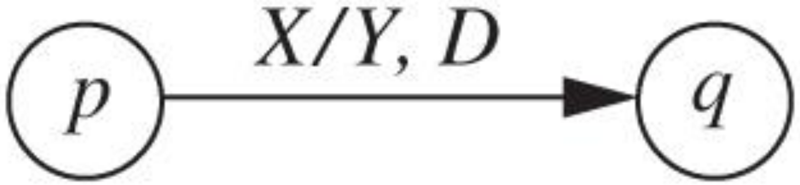
1. A finite set of states Q
2. A finite set of input alphabets Σ
3. A finite set of tape alphabets Γ ($\Sigma \subseteq \Gamma$)
4. A transition function δ: $Q \times \Gamma \rightarrow Q \times \Gamma \times \{L,R,S\}$
5. An initial state $q_0 \in Q$
6. A blank symbol Δ, $\Delta \in \Gamma$, $\Delta \notin \Sigma$
7. A set of accepting (or final) states $F \subseteq Q$

TM: Transition Function

- Typically, the transition function of a TM is of the form:

$$\delta(p, X) = (q, Y, D)$$

- p is the **current state**
- X is the **current tape symbol** pointed by tape head
- TM changes from state p to state q, after the move:
  - X is replaced with symbol Y
  - D indicates the direction of the move for the tape head
    - D = L, the tape head moves left by one position, D = R, the tape head moves right by one position, and D = S, the tape head doesn't move
- Can be represented by the following diagram (**read**/**write**, **move**)



TM Configuration

- We describe the current configuration of a TM by a single string

$$xqy$$

- q is the current state, x is the string of symbols to the left of the current square, y is either null or the string of symbols to the right of the current square (including the symbol in the current square), and everything after xy on the tape is blank
- If all symbols on and to the right of the current square are blanks, write:

$$xq\Delta$$

- We can describe the tape without mentioning the current state (s is the symbol in the current square)

$$x\underline{s}y$$

- Or:

$$x\underline{y} \ (x\underline{\Delta} \text{ if } y = \varepsilon)$$

Default Initial Configuration

- By default, if TM M has input alphabet $\sum$ and $x \in \sum^*$, the initial configuration corresponding to input x is given by

$$q_0 \Delta x$$

- In this configuration, M is in the initial state, the **tape head is in square 0**, that square is blank, and ***x* begins in square 1**. A TM starting in this configuration most often begins by moving its tape head one square right to begin reading the input
- But we can also start with other configurations
- Like PDA, we also have the "goes-to" relation $\vdash_M$ between two configurations for TM in order to trace a sequence of moves . $\vdash_M^n$ and $\vdash_M^*$ refer to n moves and zero or more moves, respectively

The Language of A TM

- Given an input string $x \in \Sigma^*$, is x accepted by a TM M?
- Initial condition:
  - The (**whole**) input string x is on M, followed by infinite blank symbols
- Final acceptance:
  - Accept x if M enters **final accepting state** and halts
  - If M halts and but **is not** in final accepting state, then **reject**
- $L(M) = \{x \in \Sigma^* \mid q_0\Delta x \vdash_M^* wqy \wedge q \in F\}$
  - Starting in the initial configuration corresponding to input x, M eventually halts in the final accepting state
- A language $L \subseteq \Sigma^*$ is accepted by a TM M if $L = L(M)$, where

$$L(M) = \{x \in \Sigma^* \mid x \text{ is accepted by } M\}$$

The Language of A TM

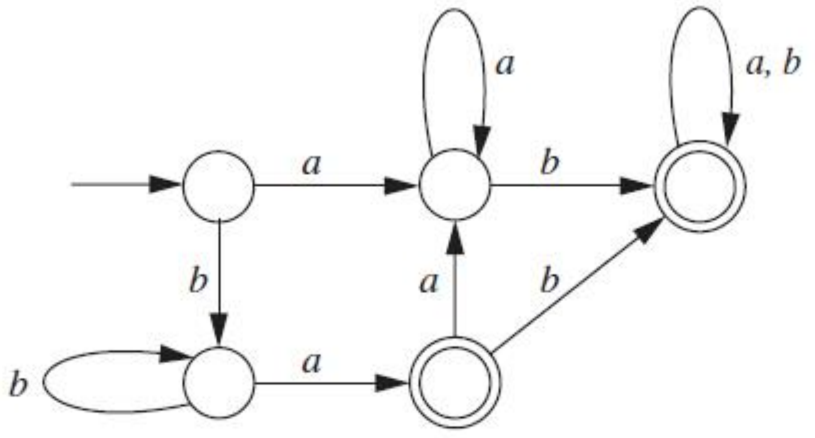
- Saying that L is accepted by M means that for any $x \in \sum^*$, $x \in L$ if and only if x is accepted by M
- This **does not** imply that if $x \notin L$, then x is rejected by M
- Two possibilities for a string x not in $L(M)$:

1. M rejects x
2. M never halts, or loops forever, on input x

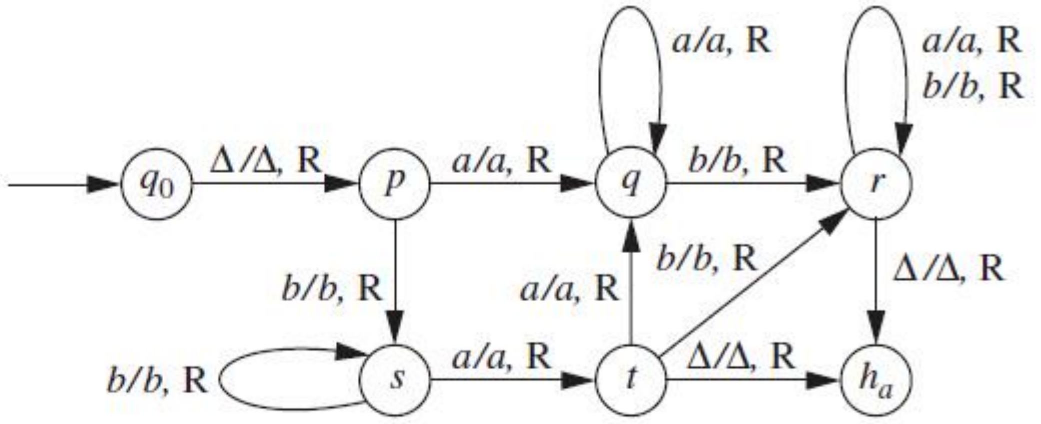
- A TM **may never** stop
  - This is unlike the machines we have encountered before

TM Example 1

- A DFA and a TM accepting the same language $L = \{a,b\}^*\{ab\}\{a,b\}^* \cup \{a,b\}^*\{ba\}$:
  - The language of all strings in $\{a,b\}^*$ that either contain the substring ab or end with ba



(a)



(b)

- Regular languages: TM just needs to make one pass, moving **right** each time, and never modifying the symbols on the tape

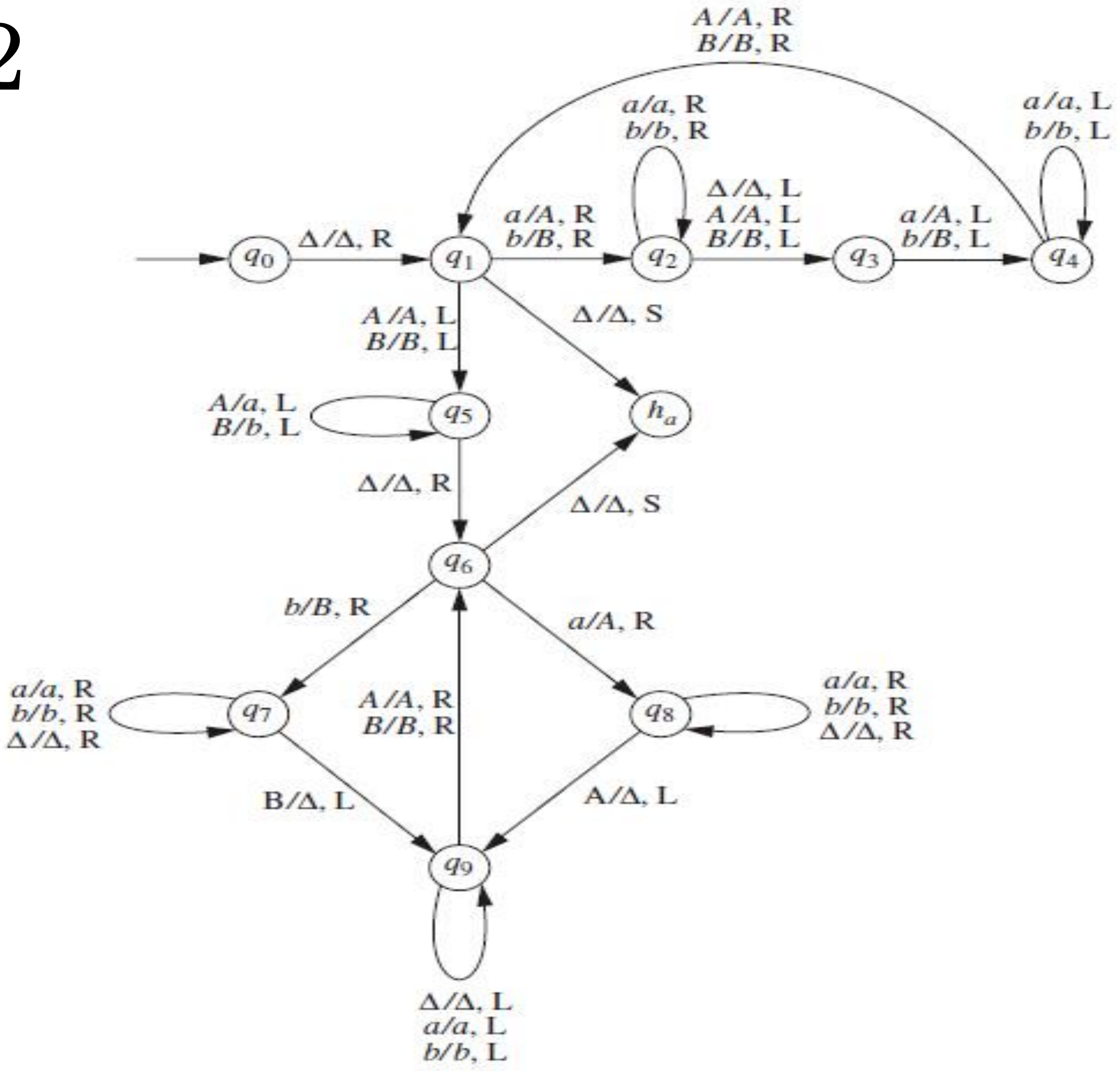
TM Example 2

- This TM accepting

  $\{xx \mid x \in \{a,b\}^*\}$

- Does it accept

1. *aba*
2. *ab*
3. *aa*

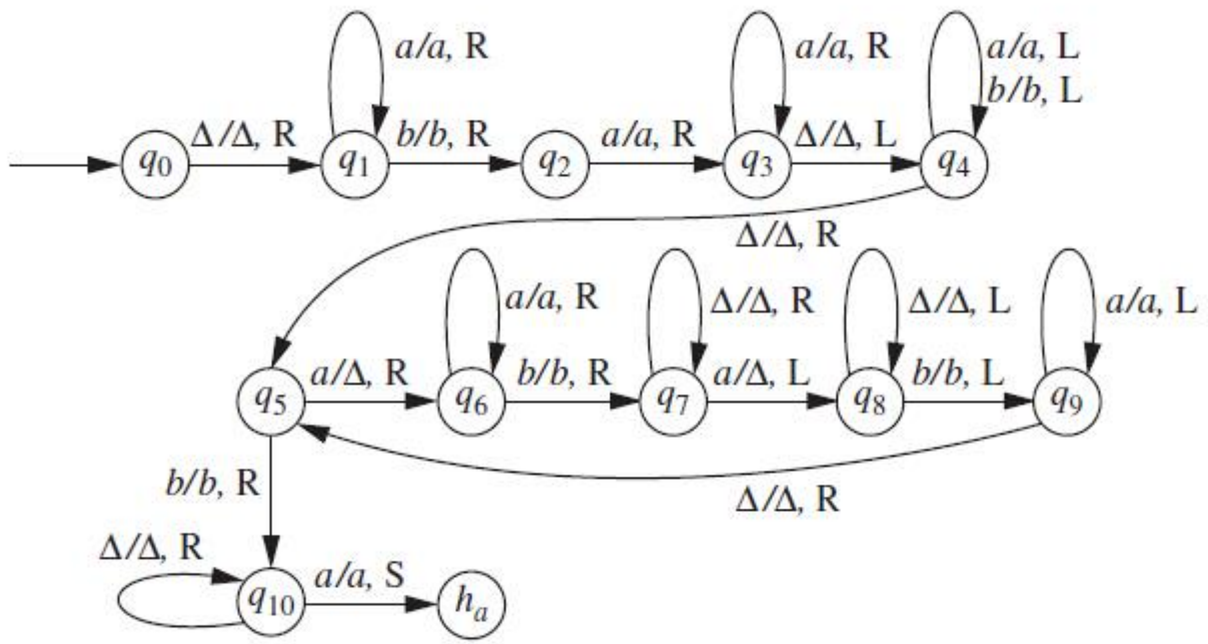


How Does It Work?

- The first half of the processing starts at the beginning and continue to move its tape head back and forth from one side to the other, it will replace input symbols by the corresponding uppercase letters to keep track of its progress. Once it arrives at the middle of the string—and if it realizes at this point that the length is odd, it can reject the string
- The second half of the processing starts at the beginning again and, for each lowercase symbol in the first half of the string, compares the lowercase symbol to the corresponding uppercase symbol in the second half. Here again we must keep track of our progress, and we do this by changing lowercase symbols to uppercase and simultaneously erasing the matching uppercase symbols

TM Example 3

- This TM accepting $\{a^i b a^j \mid 0 \le i < j\}$



- For the input string *aba*, TM is in an infinite loop. Looking for an *a* that would allow it to accept, but with no way of knowing that it will never find one

- But, the TM still works. Since it **does not** accept any string that **is not** in the language

TM Simulator

- There are many TM simulators on-line. Try this (or some other) example with one of those

  http://morphett.info/turing/turing.html#LoadMenu

TM Computing A Function

- A computer program whose purpose is to compute an output string for every legal input string can be interpreted as **computing a function** from one set of strings to another
- We'll consider TMs that compute partial functions on $(\Sigma^*)^k$, i.e., functions of k variables
- **Turing Machine Computing a Function**: For every input string x in the domain of the function, TM carries out a computation that ends with the output string f(x) on the tape (and TM in an accepting state)

TM Computing A Function

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

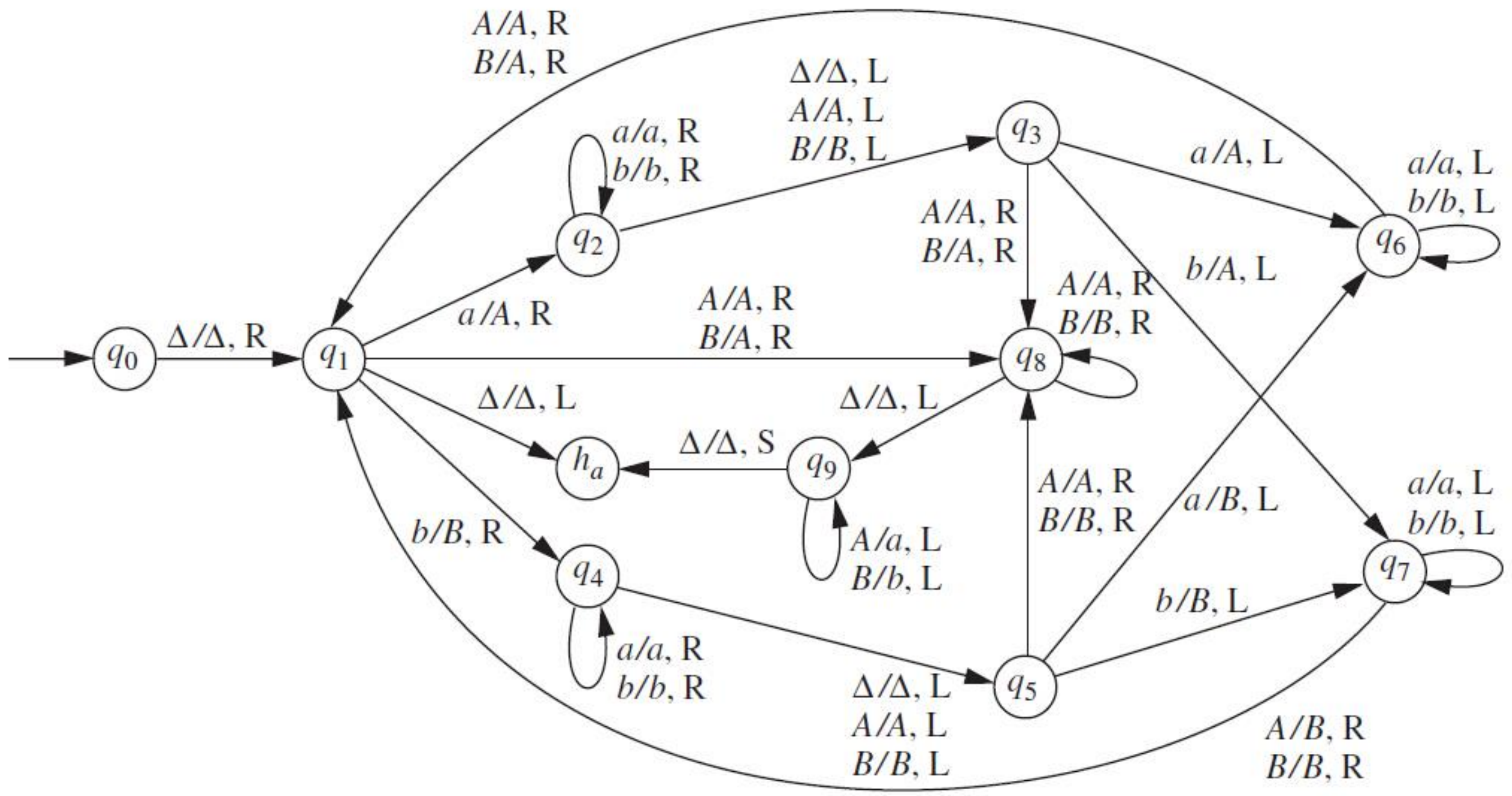
$$q_0 \, \Delta \, x_1 \, \Delta \, x_2 \, \Delta \ldots \Delta \, x_k \vdash_M^* h_a \, \Delta \, f(x_1, x_2, \ldots x_k)$$

$$h_a \in F$$

- And no other input that is a k-tuple of strings is accepted by M

TM Computing A Function: Example

- A Turing machine that computes the reverse of a string



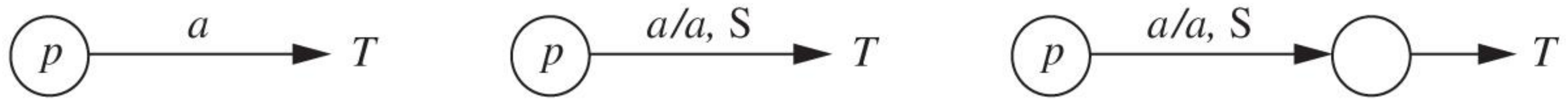
Combining Turing Machines

- 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

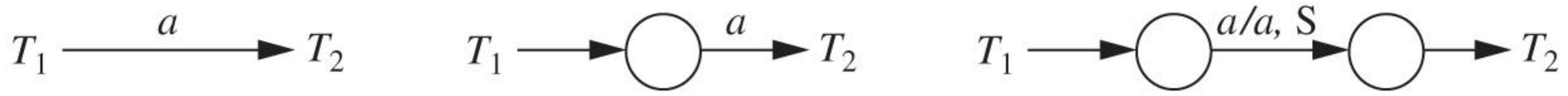
- In the simplest case, if T_1 and T_2 are TMs, we can consider the composition T_1T_2: "first execute T_1, then execute T_2 on the result"
  - The set of states of T_1T_2 is the union of the sets of states of T_1 and T_2
  - The initial state is the initial state of T_1
  - The transitions of T_1T_2 include all of those of T_2 and all of those of T_1 that don't go to the accepting state
  - A transition in T_1 that goes to the accepting state is replaced by a similar transition that goes to the start state of T_2
  - It is important that the output of T_1 be a valid input configuration for T_2

Combining Turing Machines

- We may use transition diagrams containing notations such as $T_1 \rightarrow T_2$, in order to avoid showing all the states explicitly
- We might use any of the following to mean "in state p, if the current symbol is a, then execute the TM T"



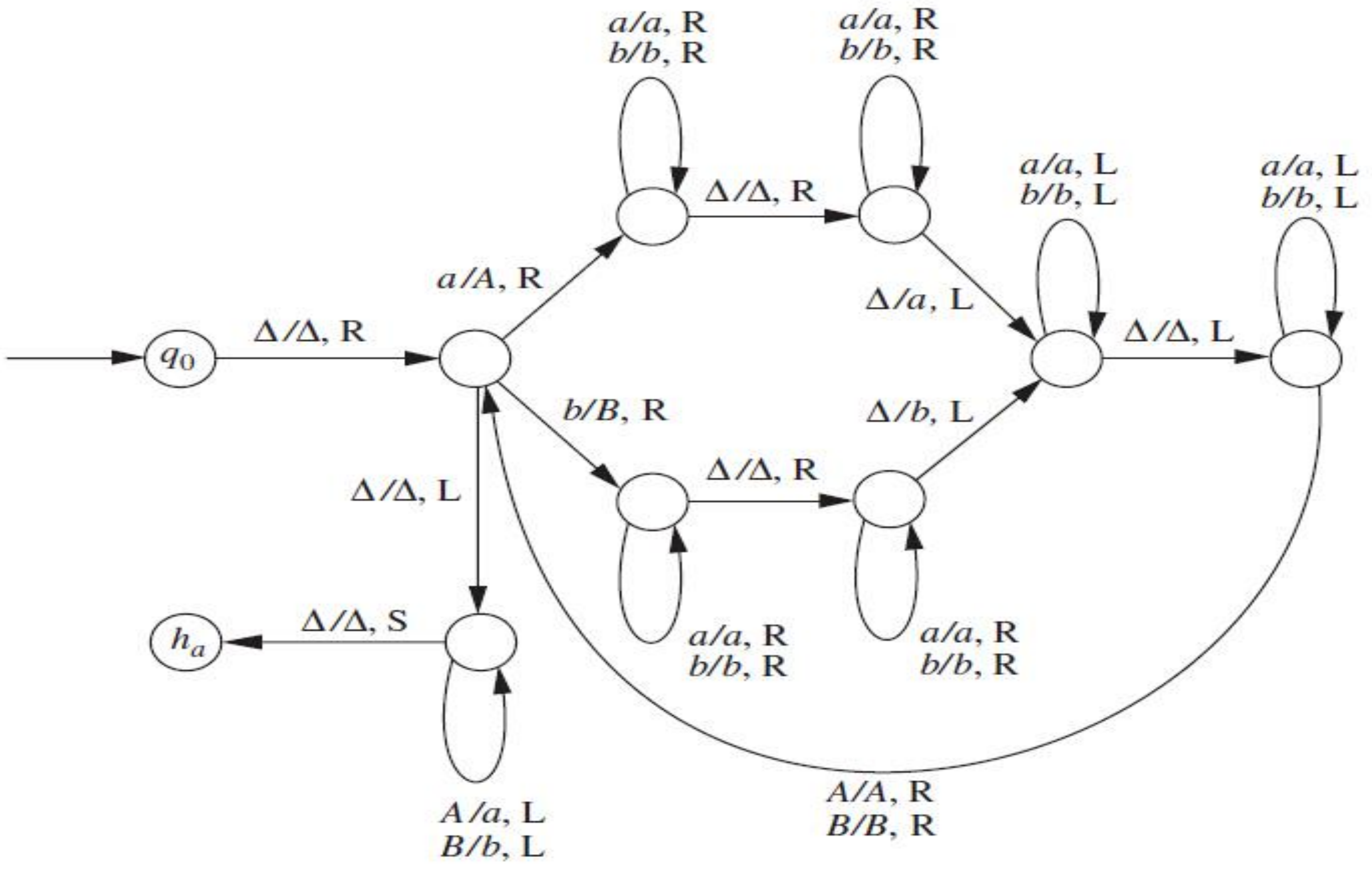
- Similarly we might use any of the following to mean "execute T_1, and if T_1 halts the accepting state with current symbol a, then execute T_2"



Combining TMs: Example

- The following TM is used to copy the input string *x*

$$q_0\Delta x \vdash \Delta q_1 x \vdash^* q_c x\Delta x$$



Combining TMs: Example

- We can composite several TMs to accept the Language of Palindromes
  - A palindrome is a sequence of characters which reads the same backward as forward – e.g., madam, abba
- We can form a composite TM, as:

$$Copy \rightarrow NB \rightarrow R \rightarrow PB \rightarrow Equal$$

- Copy the input string: $q_0\Delta abba \vdash^* q_c abba\Delta abba$
- Then find next blank: $q_c abba\Delta abba \vdash^* abba\Delta q_{NB} abba$
- Reverse the 2nd half of the string: $abba\Delta q_{NB} abba \vdash^* abba\Delta q_R abba$
- Find the previous blank: $abba\Delta q_R abba \vdash^* abba q_{PB}\Delta abba$
- Then finally check that if the two strings are equal, in this case accept, i.e., *abba* is a palindrome

Multitape Turing Machines

- If we have several types of data, it may be useful to have more than one "tape"
- Several tapes, with independent heads, are able to move all heads simultaneously in a single move
- This is useful, but we may have a different model of computation
- It will turn out that that, just as nondeterminism and ε-transitions do not increase the power of NFAs, allowing a Turing machine multiple tapes does not increase its power

Multitape Turing Machines

- We can show that for every multitape TM *M* there is a single-tape TM that **accepts exactly the same strings** as *M*, **rejects the same strings**, and **produces exactly the same output** for every input string it accepts
- Proof by using 1-tape TM to simulate a 2-tape TM
  - Textbook, page 245
- In fact, there are many different variants of TMs, e.g., TM with storage, Multi-track TM, etc. They are all equivalent

Church-Turing Thesis

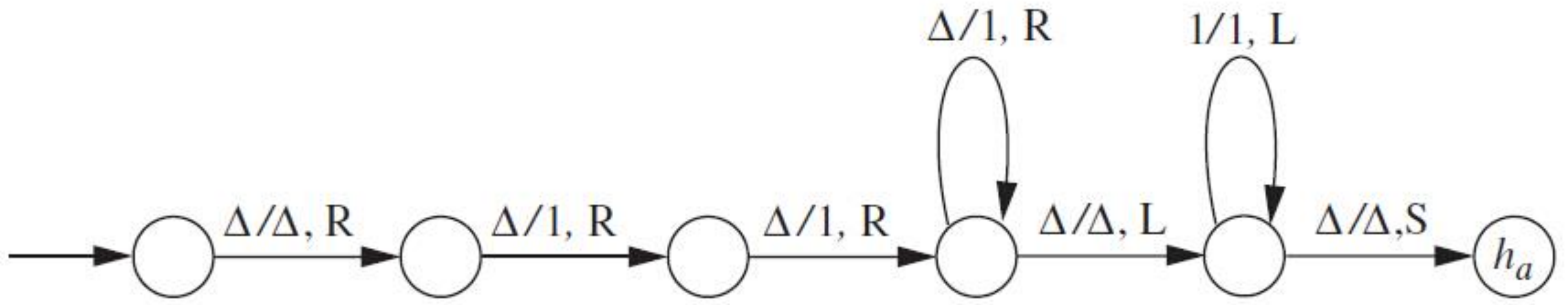
- TM is a general model of computation – **any algorithmic procedure** that can be carried out at all, by a human computer or a team of humans or an electronic computer, **can be carried out by a TM**
- This is not mathematically precise, but there is lots of evidence that it is true

Church-Turing Thesis: Evidence

- The nature of the model makes it seem that a TM can execute any algorithm a human can
- Various enhancements to a TM have been presented to make them operate more like a human, but the computing power is unchanged
- Other theoretical models of computation proposed have been shown to be equivalent to a TM
  - λ-calculus, μ-recursive functions
- **No one has ever suggested any kind of computation that cannot be implemented on a TM**
- Thus, 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
- The following NTM generates a string of two or more 1s



- The NTM essentially "makes a guess"
- Note: It is not practical for computing a function, since a TM should produce no more than one output for a given input

Nondeterministic Turing Machines

- The idea of a non-deterministic TM (NTM) can be useful (particularly as part of a composite machine)
- **Theorem**
- For every nondeterministic TM *M* there is an ordinary (deterministic) TM M_1 with $L(M_1) = L(M)$
- We can (in principle) replace every NTM with a (deterministic) TM. This stops us from needing guesses

Universal Turing Machines

- The TMs we have studied so far have been special-purpose computers capable of executing a single algorithm
- Just as a modern computer can store a program and execute any program stored in memory, so too can a "**Universal Turing Machine**"
- Turing (1936) anticipated "loading" different programs into a TM
    - The input string describes the program to run and any data the program needs to process

Universal Turing Machines

- **Definition**:
- A universal Turing machine is a Turing machine M_u that works as follows
  - It is assumed to receive an input string of the form $e(M)e(z)$, where M is an arbitrary TM, z is a string over the input alphabet of M, and e is an encoding function whose values are strings in $\{0,1\}^*$
  - The computation performed by M_u on this input string satisfies these two properties
    1. M_u accepts $e(M)e(z)$ if and only if M accepts z
    2. If M accepts z and produces output y, then M_u produces output $e(y)$

Universal Turing Machines: Encoding

- Let's have a look at a simple encoding function e
- State labels will be replaced by numbers, and we will base the encoding on these numbers
- In order to use number to encode symbols as well, we adopt this convention: there is an infinite set $S_1 = \{a_1, a_2, \ldots\}$ of symbols, including $a_1 = \Delta$, such that the tape alphabet of every TM M is a subset of S_1 , and assign number to the symbol $n(a_i) = i$
- The idea of the encoding is to represent a TM as a set of moves, and each move

$$\delta(p, a) = (q, b, D)$$

is associated with a 5-tuple of numbers that represent the five components $p, a, q, b, D,$ and each number is in unary followed by a **0**

Universal Turing Machines: Encoding

- **Definition**: If M is a TM and z is a string, define the strings $e(M)$ and $e(z)$ as:
- First assign numbers to each state, tape symbol, and tape head direction of M, and $n(h_a) = 1$, $n(h_r) = 2$, and $n(q_0) = 3$
- The other elements of Q get distinct numbers ≥ 4, and $n(R) = 1$, $n(L) = 2$, $n(S) = 3$
- For each move m of the form $\delta(p, a) = (q, b, D)$

$$e(m) = 1^{n(p)}\mathbf{0}1^{n(a)}\mathbf{0}1^{n(q)}\mathbf{0}1^{n(b)}\mathbf{0}1^{n(D)}\mathbf{0}$$

- List the moves of M as $m_1, \ldots, m_k$ (the order is arbitrary), and let

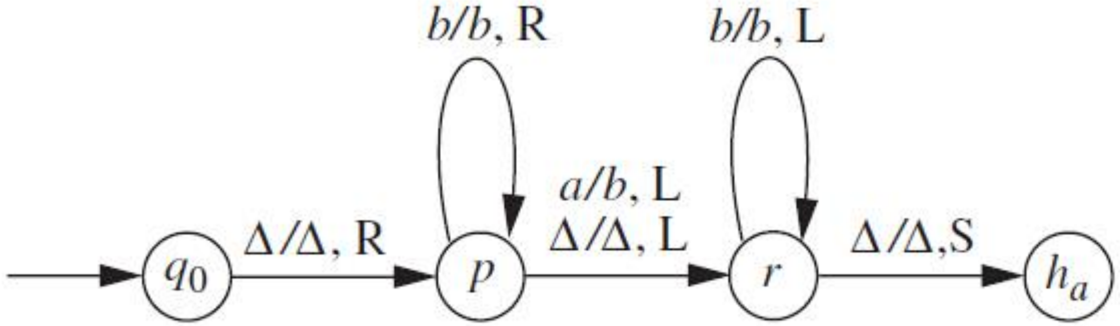
$$e(M) = e(m_1)\mathbf{0}e(m_2)\mathbf{0}\ldots\mathbf{0}e(m_k)\mathbf{0}$$

- If $z = z_1z_2 \ldots z_j$ is a string, then

$$e(z) = \mathbf{0}1^{n(z1)}\mathbf{0}1^{n(z2)}\mathbf{0}\ldots\mathbf{0}1^{n(zj)}\mathbf{0}$$

Universal Turing Machines: Example

- This TM transforms an input string of *a*'s and *b*'s by changing the leftmost *a*, if there is one, to *b*

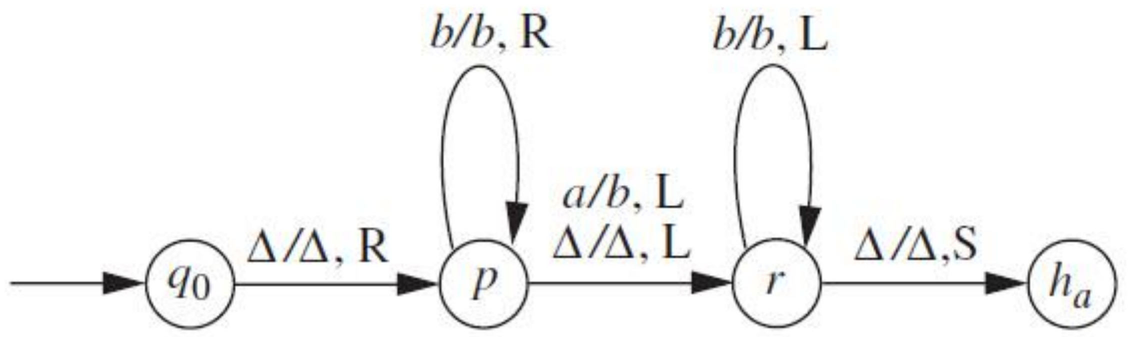


- We assume for simplicity that $n(a) = 2$ and $n(b) = 3$. By definition, $n(q_0) = 3$, and we let $n(p) = 4$ and $n(r) = 5$
- For the first move, m_1, it is determined by $\delta(q_0, \Delta) = (p, \Delta, R)$

$$e(m_1) = 1^{n(q0)}\mathbf{0}1^{n(\Delta)}\mathbf{0}1^{n(p)}\mathbf{0}1^{n(\Delta)}\mathbf{0}1^{n(R)}\mathbf{0}$$

$$e(m_1) = 111\mathbf{0}1\mathbf{0}1111\mathbf{0}1\mathbf{0}1\mathbf{0}$$

Universal Turing Machines: Example



- If we encode the moves in the order they appear in the diagram, left to right, we may have

$$e(M) = e(m_1)\mathbf{0}e(m_2)\mathbf{0}\ldots\mathbf{0}e(m_k)\mathbf{0}$$

$e(M)$ = 111010111101010**0** 11110111011110111010**0**
111101101111101110110**0** 111101011111010110**0**
11111011101111101110110**0** 1111101010101110**0**

Concluding Remarks

- TM – a general model of computation
- Church–Turing Thesis – any real-world computation can be translated into an equivalent computation involving a TM
- Language of a TM
- Use TM to Compute a function
- Combine TMs
- Multitape Turing Machines
- Nondeterministic Turing Machines
- Universal Turing Machines