Chapter 7

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 $\{xcx^r \mid x \in \{a, b\}^*\}$
  - A PDA cannot accept $AnBnCn = \{a^n b^n c^n \mid n \geq 0\}$ or $L = \{xcx \mid x \in \{a,b\}^*\}$

A General Model of Computation (cont'd.)

- A PDA-like machine with *two* stacks can accept *AnBnCn*
- An FA with a *queue* instead of a stack can accept *L*
- 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 (cont'd.)

- 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

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

A General Model of Computation (cont'd.)

- 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

A General Model of Computation (cont'd.)

- In our version of a Turing machine, 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
  - Changing from the current state to another state
  - Replacing the symbol in the square by another
  - Leaving the tape head where it is, moving it one square to the left, or moving it one square to the right
- The input string is assumed to be on the tape initially

A General Model of Computation (cont'd.)

- The tape provides the memory needed during computation and serves as the output device
- 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 and the other rejection. (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
  - This will turn out to be important

A General Model of Computation (cont'd.)

- Definition 7.1: 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

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

A General Model of Computation (cont'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
  - If the state q is either h_a or h_r we say that this move causes T to halt
    - Once it halts, it cannot move
  - We return to drawing transition diagrams similar to but more complicated than the diagrams for FAs

A General Model of Computation (cont'd.)

- The transition $\delta(p, X)=(q, y, D)$ will be represented by the following diagram
  - If the TM attempts to move the tape head to the left when it is on square 0, we will say that the TM halts in state h_r, leaving the tape head in square 0 and leaving the tape unchanged

A General Model of Computation (cont'd.)

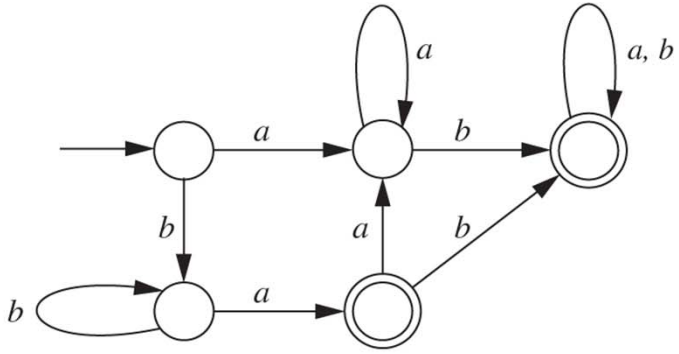
- Normally a TM begins with an input string starting in square 1 and all other squares (square 0 and all the ones following the input string) blank
- In any case, the set of nonblank squares on the tape must always be finite
- We describe the current *configuration* of a TM by a single string *xqy* where *q* is the current state, *x* is the string of symbols to the left of the current square, *y* is either null or starts in the current square, and everything after *xy* on the tape is blank

A General Model of Computation (cont'd.)

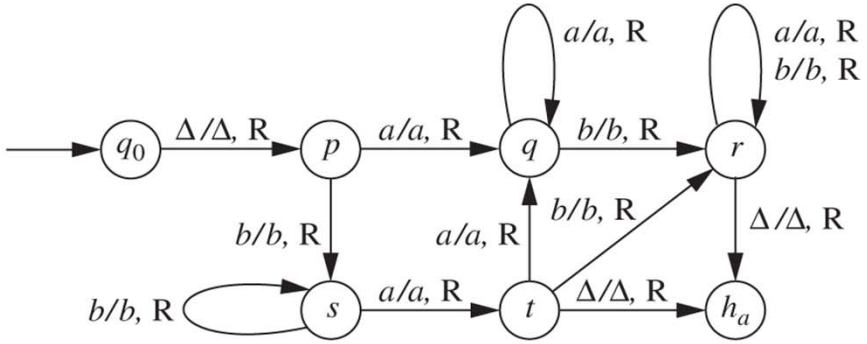
- We trace a sequence of moves by specifying the configuration at each step
- If q is a non-halting state and r is any state, we write $xqy \vdash_T zrw$ or $xqy \vdash_T^* zrw$ to mean that T moves from the first configuration to the second in one move, or in zero or more moves, respectively
- The initial configuration corresponding to input x is given by $q_0 \Delta x$

Turing Machines as Language Acceptors

- Definition 7.2: If $T = (Q, \Sigma, \Gamma, q_0, \delta)$ is a TM and $x \in \Sigma^*$, x is accepted by T if $q_0 \Delta x \vdash_T^* whay$ for some $w, y \in (\Gamma \cup \{\Delta\})^*$
- A language $L \subseteq \Sigma^*$ is accepted by T if
  $L = L(T) = \{x \in \Sigma^* \mid x \text{ is accepted by } T\}$
- The following transition diagrams show an FA and a TM that accept the same language



(a)



(b)

Turing Machines as Language Acceptors (cont'd.)

- If the language were not regular, the TM could not move its tape head to the right on every move.
- The (*b*) diagram on the previous slide does not show any of the moves to the reject state
  - They all have the same form, and there is one from each of the states *p*, *q*, and *s* (the nonhalting states other than q_0 that correspond to nonaccepting states in the FA), as shown below

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'll also 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

Turing Machines that Compute Partial Functions (cont'd.)

- Definition 7.9:
  - 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

Turing Machines that Compute Partial Functions (cont'd.)

- 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 Definition 7.9, except that the input alphabet is {1}, and the initial configuration looks like $q_0\Delta 1^{n_1}\Delta 1^{n_2}\Delta \ldots \Delta 1^{n_k}$

Combining Turing Machines

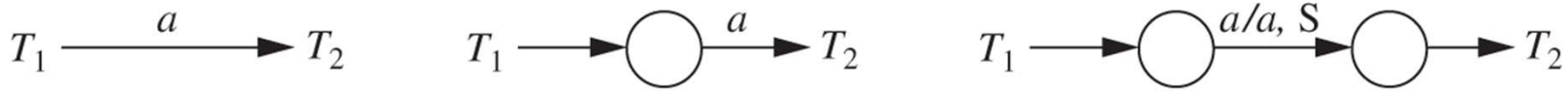
- Just as a large algorithm can be described as a number of subalgorithms working in combination, we can combine several Turing machines into a larger composite TM
- In the simplest case, if T_1 and T_2 are TMs, we can consider the composition T_1T_2: "first execute T_1, then execute T_2 on the result"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

Combining Turing Machines (cont'd.)

- 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
- It is important that the output of T_1 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 explicitly

Combining Turing Machines (cont'd.)

- We might use any of the above notations 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 in h_a with current symbol a, then execute T_2"



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 Turing machine
  - It will turn 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

Multitape Turing Machines (cont'd.)

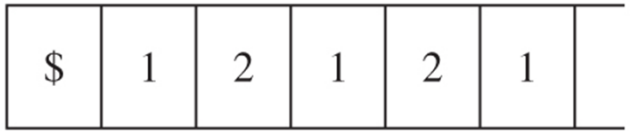
- We will show that for every multitape TM *T* there is a single-tape TM that accepts exactly the same strings as *T*, rejects the same strings, and produces exactly the same output for every input string it accepts
- To simplify the discussion we will only consider two-tape machines, but it is easy to see that the principles are the same if there are more than two

Multitape Turing Machines (cont'd.)

- 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 $(q_0, \underline{\Delta}\, x, \underline{\Delta})$

Multitape Turing Machines (cont'd.)

- Theorem 7.26: For every 2-tape TM $T = (Q, \Sigma, \Gamma, q_0, \delta)$ there is an ordinary 1-tape TM $T_1 = (Q_1, \Sigma, \Gamma_1, q_1, \delta_1)$, with $\Gamma \subseteq \Gamma_1$, such that:
  - $L(T) = L(T_1)$
  - For every $x \in \Sigma^*$, if $(q_0, \underline{\Delta}\, x, \underline{\Delta}) \vdash_T^* (h_a, y\underline{a}z, u\underline{b}v)$, then $q_1 \Delta x \vdash_{T_1}^* y h_a a z$
- Proof: by construction
  - The tape is divided into two "tracks", as suggested by this diagram:



Multitape Turing Machines (cont'd.)

- In square 0, there is a marker to make it easy to locate the beginning of the tape
  - It will also be helpful to have a marker at the other end of the nonblank portion of the tape
- Starting with the initial configuration $q_1 \Delta x$ with input $x = a_1 a_2 \ldots a_n$, T_1 places the \$ marker in square 0, inserts blanks between consecutive symbols of x and places the # marker after the last nonblank symbol, to produce the tape

  $\$\underline{\Delta}\Delta a_1 \Delta a_2 \Delta \ldots \Delta a_n \#$

Multitape Turing Machines (cont'd.)

- From this point on, the # is moved if necessary to mark the farthest right that T has moved on either of its tapes
- The only significant complication in T_1 simulating T is that it must be able to keep track of both of the tape heads
- We handle this by including in T_1's tape alphabet an extra copy σ' of every symbol σ (including Δ) that can appear on T's tape
- The primed symbol gives the location of the head

Multitape Turing Machines (cont'd.)

- By using extra states T_1 can "remember" the current state of T and also what the primed symbol on the first track is while it is looking for the primed symbol on the second track (or vice-versa)
- The steps T_1 makes to simulate a move of T are:
  - Move the tape head left to the \$, then right to a primed symbol σ'; remember σ and move back to \$
  - Move right to a primed symbol τ' on the second track; if the move that is now determined is $\delta(p, \sigma, \tau) = (q, \sigma_1, \tau_1, D_1, D_2)$ then reject if $q = h_r$, else change τ' to τ_1 and move in direction D_2 to the appropriate square

Multitape Turing Machines (cont'd.)

- The steps T_1 makes (cont'd,)
  - If in moving the tape head this way, the \$ is encountered, then reject, otherwise (moving the # if necessary) change the symbol in the new square on track 2 to the corresponding primed symbol and move back to the \$
  - Locate σ' on the first track again, change it to σ_1, and move the tape head in direction D_1 to the appropriate square on the first track
  - After allowing for either marker, as above, change the new symbol on the first track to the corresponding primed symbol

Multitape Turing Machines (cont'd.)

- As long as T has not halted, iterating these steps allows T_1 to simulate the moves of T correctly
- If T finally accepts, then T_1 must carry out these steps in order to end up in the right configuration:
  - Delete every square in the second track to the left of #
  - Delete both end-of-tape markers
  - Move the tape head to the primed symbol , change it to the corresponding unprimed symbol, and halt in h_a with the tape head on that square

Multitape Turing Machines (cont'd.)

- Corollary 7.27:
  - Every language that is accepted by a 2-tape TM can be accepted by an ordinary 1-tape TM, and every function that is computed by a 2-tape TM can be computed by an ordinary TM

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 isn't a statement that can be proved, but there is a lot of evidence for it

The Church-Turing Thesis (cont'd.)

- The nature of the model makes it seem that a TM can execute any algorithm a human can
- Enhancements to the TM have been shown not to increase its power
- Other theoretical models 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 7.31: For every nondeterministic TM 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
  - See book for details

Universal Turing Machines

- The TMs we have studied so far have been special-purpose computers capable of executing a single algorithm
- We can consider a “universal” Turing machine, which is capable of executing a program stored in its memory
  - It receives an input string that specifies both the algorithm it is to execute and the input that is to be provided to the algorithm

Universal Turing Machines (cont'd.)

- Definition 7.32: 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 $e(T)\ e(z)$, where T is an arbitrary TM, z is a string over the input alphabet of T, and 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 $e(T)e(z)$ if and only if T accepts z
    - If T accepts z and produces output y, then T_u produces output $e(y)$

Universal Turing Machines (cont'd.)

- We discuss a simple encoding function e, and then sketch one approach to constructing a universal TM
- The encoding function has several crucial features:
  - 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 (cont'd.)

- 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 $S = \{a_1, a_2, \ldots\}$ of symbols, including $a_1 = \Delta$, such that the tape alphabet of every TM T is a subset of S
- 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 followed by a 0

Universal Turing Machines (cont'd.)

- Definition 7.33: 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, and $n(\mathrm{R}) = 1$, $n(\mathrm{L}) = 2$, $n(\mathrm{S}) = 3$
  - 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 $e(T) = e(m_1)0e(m_2)0\ldots0e(m_k)0$

Universal Turing Machines (cont'd.)

- Definition 7.33: (cont'd.)
  - If $z = z_1 z_2 \ldots z_j$ is a string, then $e(z) = 0\ 1^{n(z_1)}\ 0\ 1^{n(z2)}\ 0\ \ldots\ 0\ 1^{n(z_j)}0$
- Theorem 7.36: 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 $(11^*0)^5 0((11^*0)^5 0)^*$, so that it is a sequence of 5-tuples
    - No two substrings of x representing 5-tuples have the same first two parts
    - None of the 5-tuples have first part 1 or 11
    - The last part of each 5-tuple must be 1, 11, or 111

Universal Turing Machines (cont'd.)

- 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
- Details of the simulation are in the book