On Turing Machines

Eliezer A. Albacea
Institute of Computer Science
University of the Philippines Los Baños
4031 College, Laguna
eaalbacea@uplb.edu.ph

ABSTRACT

In this paper, we define Turing machines and give examples of how Turing machines may be constructed. Also illustrated is how Turing Machines compute functions. We also discussed the Halting problem and the Church-Turing Thesis.

Keywords

Turing machine, Turing-computable function, recursive language, recursively enumerable language, Church-Turing Thesis.

1. DEFINITION OF A TURING MACHINE

Definition: A Turing Machine (TM) is denoted by M = (Q, Σ, Γ, δ, q_0, B, F) where Q is the set of states, Γ is the finite set of allowable tape symbols, B is the blank symbol ∈ Γ, Σ is the set of input symbols, δ is the next move function which is a mapping from Q × Γ to Q × Γ × {L, R, S}, q_0 is the initial state and F is the set of final (halting) states (F ⊆ Q). See Figure 1 for an illustration of a Turing machine.

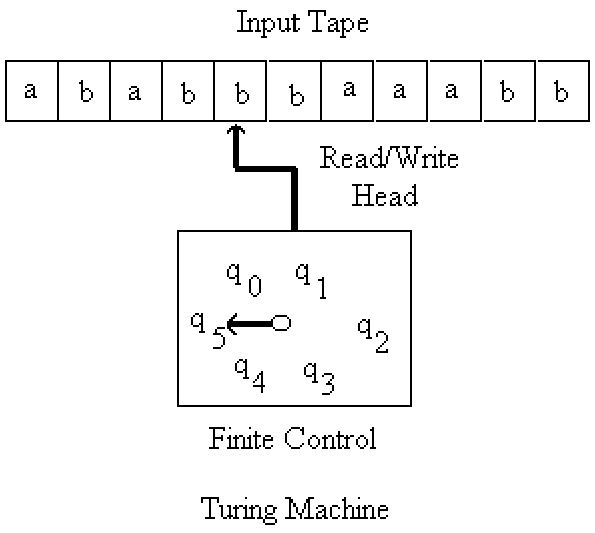


Figure 1 A Turing Machine.

The TM machine will initially be in state q0 and the read/write head will be pointing at the first symbol of the input/output tape. The δ function, δ(q,T) = (q',T',D) defines the state of computation after the TM has read the tape symbol T and it is currently in state q. In this case (q',T',D), which says that the new state is q', it writes the T' in place of T and the read/write head is move in D direction (either L = left, R = right, or S = stationary).

The input tape of the TM is infinite in both directions with blanks (B) on the tape for parts not included in the initial input.

The δ may alternatively be presented in transition diagram. The transition diagram for δ(q,T) = (q',T',D) is:

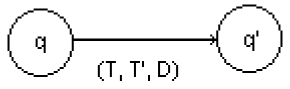


Figure 2 Transition diagram for δ(q,T) = (q',T',D).

2. EXAMPLES OF TURING MACHINES

Example: Consider the TM M = (Q, Σ, Γ, δ, q_0, B, {f}) where Q = {q_0, f}, Σ = {a} and Γ = {a, B} and δ is given by the table:

State	Symbol	
	a	B
q_0	(q_0,B,R)	(f,B,S)

This TM erases nonblank symbols in the input tape.

Initially, the TM is in state q_0 and the read/write head is pointing at the first symbol in the input tape. For example, the input tape contains the initial input BabbabaB. In reality, it should be …BBabbabaBB… but we simply shortened this to BabbabaB. Hence, we are at state q_0 and the read/write head is pointing at a. What applies therefore is the transition function

$$\delta(q_0,a) = (q_0,B,R).$$

This will bring the TM to state q_0, replaces the letter a (input being pointed to at the moment) with a B and moves the read/write head one input to the right (R). If the direction of head movement is L then it moves to the left and if it is S, then it simply stays where it is. Hence, it will now be in state q_0 and reading the input letter b. Then, it simply checks the transition table for the next move and the process is repeated until it reaches the final state in which case it halts recognizing what is in the input. Or, it may encounter an undefined entry in the transition table, in which case it also halts and not recognizing what is in the input.

In transition diagram, the TM above can be shown as:

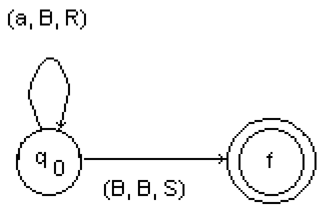


Figure 3

Example: Consider the TM M = (Q, Σ, Γ, δ, q_0, B, {f}) where Q = {q_0, f}, Σ = {a} and Γ = {a, B} and δ is given by the table:

State	Symbol	
	a	B
q_0	(q_0,B,L)	(q_0,B,L)

This TM hangs or this TM machine does not stop executing because it will always find a B in the tape. Note that the tape is infinite and it contains B. In programming parlance, we say that the program is in "infinite loop."

In transition diagram, the TM above can be shown as:

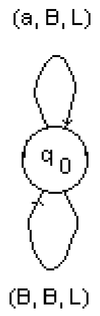


Figure 4

Example: Consider the TM M = (Q, Σ, Γ, δ, q_0, B, {f}) where Q = {q_0, f}, Σ = {a,b} and Γ = {a, b, B} and δ is given by the table:

State	Symbol		
	a	b	B
q_0	(q_0,b,R)	(q_0,a,R)	(f,B,S)

This TM replaces all occurrence of a by a b and all occurrence of a b by an a.

In transition diagram, the TM above can be shown as:

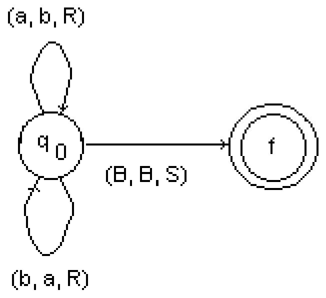


Figure 5

Example: Consider the TM M = (Q, Σ, Γ, δ, q_0, B, {f}) where Q = {q_0, q_1 ,f}, Σ = {a,b} and Γ = {a, b, B} and δ is given by the table:

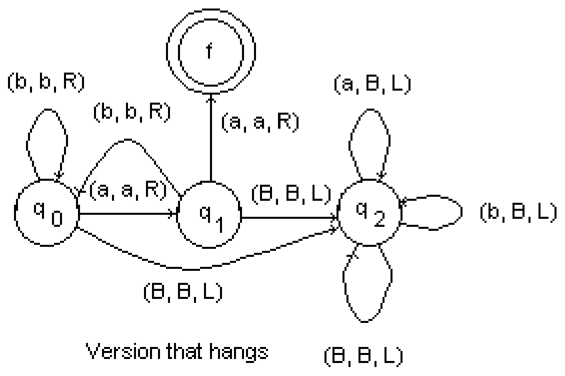
State	Symbol		
	a	b	B
q_0	(q_1,a,R)	(q_0,b,R)	(q_2,B,L)
q_1	(f,a,R)	(q_0,b,R)	(q_2,B,L)
q_2	(q_2,B,L)	(q_2,B,L)	(q_2,B,L)

This TM searches for the substring aa. If it finds aa, then it goes to the final state. Otherwise, it simply erases the tape and hangs.

We note that a string may contain the substring aa or not. There is no problem with the case where aa is found in the input since the TM can go to a final state. But if the string does not contain the substring aa, then we can either make the TM hang as what was done above or we force the TM to stop. The TM stops when the δ is undefined. For example, we can make the TM above stops when we encounter a B in the input tape.

State	Symbol		
	a	b	B
q_0	(q_1,a,R)	(q_0,b,R)	
q_1	(f,a,R)	(q_0,b,R)	

In transition diagrams, the TM's above can be shown as:



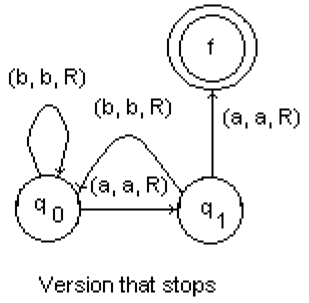


Figure 6

Example: The Turing Machine for recognizing the language L = aba*b is given by TM M = (Q, Σ, Γ, δ, q_0, B, {f}) where Q = {q_0, q_1, q_2,f}, Σ = {a,b} and Γ = {a, b, B} and δ is given by the table:

State	Symbol		
	a	b	B
q_0	(q_1,a,R)		
q_1		(q_2,b,R)	
q_2	(q_2,a,R)	(f,b,R)	

.

In transition diagram, the TM above can be shown as:

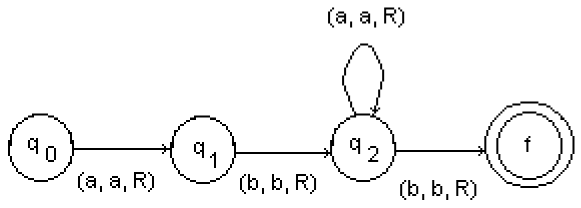


Figure 7

3. INSTANTANEOUS DESCRIPTION

Definition: The instantaneous description (ID) of a TM is denoted by $\alpha_1 q \alpha_2$ where $\alpha_1\alpha_2 \in \Gamma^*$ and q is the current state ($q \in Q$).

When $\alpha 2 = \in$ then the TM is said to be in a halted description.

Definition: Let M = (Q, Σ, Γ, δ, q_0, B, F) and let $\alpha_1 q_i \alpha_2$ and $\alpha_3 q_{i+1} \alpha_4$ be instantaneous descriptions. Then $\alpha_1 q_i \alpha_2 \mid\!- \alpha_3 q_{i+1} \alpha_4$ iff for some $a \in \Sigma$, $A \in \Gamma$, $\delta(q_i, a) = (q_{i+1}, A, R)$ and $\alpha_3 = \alpha_1 A$, $\alpha_2 = a\alpha_4$.

The TM is said to yield the second configuration (ID) in one step.

Definition: For any TM M, |-* is the reflexive, transitive closure of |-. We say that ID c_1 yields ID c_2 if $c_1 \mid\!-^* c_2$. A computation by M is a sequence of ID's $c_0, c_1, c_2, \ldots, c_n$ for some $n \geq 0$ such that $c_0 \mid\!- c_1 \mid\!- c_2 \mid\!- \ldots \mid\!- c_n$. We say that the computation is of length n or has n steps.

Example: Consider the TM M for erasing nonblank symbol. If M is started with ID $q_0$aaaaB, its computation would be represented as follows:

q_0aaaaB |- Bq_0aaaB
|- BBq_0aaB
|- $BBBq_0aB$
|- $BBBBq_0B$
|- BBBBfB

Definition: A TM M = (Q, Σ, Γ, δ, q_0, B, F) is said to accept L if for any w ∈ L there is a computation q_0w |-* uf for some u ∈ Γ* and f ∈ F. If such a TM M exists, then L is said to be a Turing-acceptable language.

Example: Consider the TM M = (Q, Σ, Γ, δ, q_0, B, {f}) where Q = {q_0, q_1 ,f}, Σ = {a,b} and Γ = {a, b, B} and δ is given by the table:

State	Symbol		
	a	b	B
q_0	(q_1,a,R)	(q_0,b,R)	
q_1	(f,a,R)	(q_0,b,R)	

In transition diagram, this table is equivalent to the following:

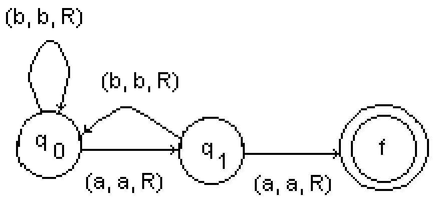


Figure 8

The TM machine accepts the language L = (b*ab*)*aa(b*ab*)* since there is a computation for every string in the language. For example, the computation for the string bbaaba is:

$q_0bbaabaB$ |- $bq_0baabaB$
|- bbq_0aabaB
|- $bbaq_1abaB$
|- bbaafbaB

4. COMPUTING WITH TURING MACHINES

Definition: Let Σ_1 and Σ_2 be alphabets of languages L_1 and L_2, respectively. Let f be a function from Σ_1 to Σ_2. A TM M = (Q, Σ, Γ, δ, q_0, B, F) is said to compute f if $\Sigma_1, \Sigma_2 \subseteq \Sigma$ and for any $w \in \Sigma_1$,

if f(w) = u then q_0w |-* uf,

where f ∈ F. If such a TM M exists, then f is said to be a Turing-computable function.

The notion of a Turing-computable function from strings to strings can be extended in several ways:

One is the Turing-computable function from string to string. It can be extended to consider functions of any number of arguments, including 0. Let f be a function from $(\Sigma_1^*)k$ to Σ_2^*, where k ≥ 0, and let M be a TM (Q, Σ, Γ, δ, q_0, B, F), where $\Sigma_1, \Sigma_2 \subseteq \Sigma$. For any $w_1, w_2, \ldots w_k \in \Sigma_1^*$, if $f(w_1, w_2, \ldots, w_k) = u$, then

$q_0w_1Bw_2Bw_3B \ldots Bw_k$ |-* uf.

Again, we say that M computes f and that f is Turing-computable.

Example: Substitute Function. Let $\Sigma_1 = \Sigma_2 = \{0,1\}$ and let f(w) = z, where $w \in \Sigma_1$ and $z \in \Sigma_2$, be defined as follows:

For any $w \in \Sigma_1$. f(w) = z, where z is the result of replacing each occurrence of 0 in w by 1, and vice versa.

Thus, an input

010101B

will produce the tape configuration

101010B.

Rough Algorithm: The TM scans forward through its input, changing 0's to 1's and vice versa, until a blank symbol is found.

Formally, the TM can be described as follows: M = ({q_0,q_1}, {0,1}, {0,1,B}, δ, q_0, B, {q_1}), where δ is defined as:

States	Symbols		
	0	1	B
q_0	$(q_0,1,R)$	$(q_0,0,R)$	(q_1,B,R)

In transiiton diagram, this TM can be shown as :

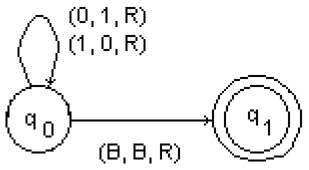


Figure 9

Example: Show the computation for the input string 0101001.

$q_00101001B$ |- $1q_0101001B$
|- $10q_001001B$
|- $101q_01001B$
|- $1010q_0001B$
|- $10101q_001B$
|- $101011q_01B$
|- $1010110q_0B$
|- $1010110Bq_1$

The Turing-computable function may be extended to consider functions from natural numbers (N) to natural numbers (N). Let I be some fixed symbol other than the blank symbol, then the natural number n may be represented by the string I^n. A function f: N → N is said to be computed by a TM M if M computes the function f': $\{I\}^* \rightarrow \{I\}^*$, where $f'(I^n) = I^{f(n)}$ for each $n \in N$. In general, a TM M computes a function f: $N^k \rightarrow N$ if it computes the function f': $(\{I\}^*)^k \rightarrow \{I\}^*$, where $f'(I^{n1}, I^{n2}, \ldots, I^{nk}) = I^{f(n1,n2,\ldots,nk)}$ for any n1, n2, …, nk ≥ 0.

Example: Successor Function. Let f be the successor function: f(n) = n+1 for each $n \in N$. A TM for this is $M = (\{q_0,q_1\}, \{0\}, \{0,B\}, \delta, q_0, B, \{q_1\})$ where δ is defined as:

States	Symbols	
	0	B
q_0	$(q_0,0,R)$	$(q_1,0,R)$

In transition diagram this can be written as:

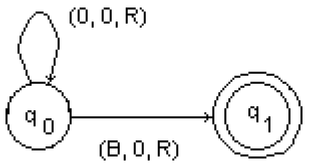


Figure 10

Thus with an initial input tape configuration of:

0000B

will produce a final tape configuration of

00000B.

Example: Show the computation for the input string 0000.

q_00000B |- $0q_0000B$
|- $00q_000B$
|- $000q_00B$
|- $0000q_0B$
|- $00000q_1$

Example: Proper Subtraction. The proper subtraction function f(m,n) is defined as:

f(m,n) = m-n, when m ≥ n,

f(m,n) = 0, when m < n.

Rough Algorithm: The objective of the TM machine is to transform the initial input tape:

0^m10^n

to

0^{m-n}, if $m \geq n$

all B's or blanks if $m < n$.

The TM repeatedly replaces its leading 0 by a B, then searches right for a 1 followed by a 0 and change this 0 to 1. Next, M moves left until it encounters a B and repeats the cycle. The repetition ends if

1. searching right for a 0, the TM encounters a B;
2. beginning the cycle, M cannot find a 0 to change to a B, because the first m 0's already have been changed. Then $m \geq n$, so $f(m,n) = 0$. The TM replaces all remaining 1's and 0's y B's.

Formally, the TM can be described as follows: $M = (\{q_0, q_1, \ldots, q_5, f\}, \{0, 1\}, \{0, 1, B\}, \delta, q_0, B, \{f\})$, where δ is defined as:

States	Symbols		
	0	1	B
q_0	(q_1,B,R)	(q_5,B,R)	
q_1	$(q_1,0,R)$	$(q_2,1,R)$	
q_2	$(q_3,1,L)$	$(q_2,1,R)$	(q_4,B,L)
q_3	$(q_3,0,L)$	$(q_3,1,L)$	(q_0,B,R)
q_4	$(q_4,0,L)$	(q_4,B,L)	$(f,0,R)$
q_5	(q_5,B,R)	(q_5,B,R)	(f,B,R)
F			

In transition diagram notation, this can be written as:

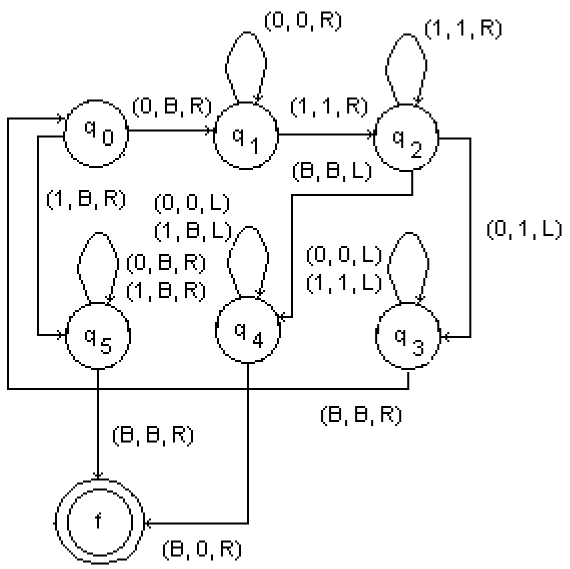


Figure 11

Example: Show the computation for the input 000100.

$q_0000100B$ |- $Bq_100100B$
|- $B0q_10100B$
|- $B00q_1100B$
|- $B001q_200B$
|- $B00q_3110B$
|- $B0q_30110B$
|- $Bq_300110B$
|- $q_3B00110B$
|- $Bq_000110B$
|- BBq_10110B
|- $BB0q_1110B$
|- $BB01q_210B$
|- $BB011q_20B$
|- $BB01q_311B$
|- $BB0q_3111B$
|- BBq_30111B
|- $Bq_3B0111B$
|- BBq_00111B
|- $BBBq_1111B$
|- $BBB1q_211B$
|- $BBB11q_21B$
|- $BBB111q_2B$
|- $BBB11q_41B$
|- $BBB1q_41BB$

$\vdash$ BBB$q_4$1BBB

$\vdash$ BBq_4BBBBB

$\vdash$ BB0fBBBB

Example: Addition. The addition function f(m,n) is defined as:

f(m,n) = m+n,

when m,n ≥ 0.

Rough Algorithm: The objective of the TM machine is to transform the initial input tape:

0^m10^n

to

0^{m+n}

.

The algorithm looks for the first occurrence of the a 1, replace this with a 0 and then continue this time looking for a B. Once a blank is found, moves one square left and replace it with a blank.

Formally, the TM can be described as follows: M = ({q_0, q_1, q_2, f), {0, 1}, {0, 1, B}, δ, q_0, B, {f}), where δ is defined as:

States	Symbols		
	0	1	B
q_0	(q_0,0,R)	(q_1,0,R)	(f,B,S)
q_1	(q_1,0,R)		(q_2,B,L)
q_2	(f,B,S)		
f			

In transition diagram, the above TM is equivalent to:

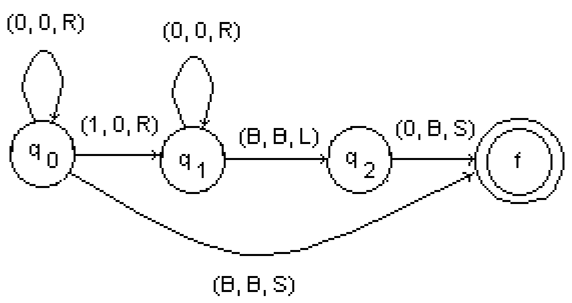


Figure 12

Example: Show the computation for the input 000100.

$q_0$000100B $\vdash$ 0$q_0$00100B

$\vdash$ 00$q_0$0100B

$\vdash$ 000$q_0$100B

$\vdash$ 0000$q_1$00B

$\vdash$ 00000$q_1$0B

$\vdash$ 000000q_1B

$\vdash$ 00000$q_2$0B

$\vdash$ 00000fBB

5. VARIANTS OF TURING MACHINES

Two-Way Infinite Tape TM

As in the original one-way infinite tape model, a machine in this model may be denoted by M = (Q, Σ, Γ, δ, q_0, B, F). The relation |- which relates two ID's is defined as for the one-way infinite tape, except that if δ(q,x) = (p,Y,L), then qxα |- pBYα and if δ(q,x) = (p,B,R), then qxα |- pα (in the original model, the B would appear to the left of p).

Multitape Turing Machines (MT-TM)

A MT-TM consists of a finite control with k-tape heads and k-tapes; each tape is infinite in both directions. In a single move, depending on the state of the finite control and the symbol scanned by each of the tape heads, the machine can

1. change state;
2. print a new symbol on each of the cells scanned by its tape head;
3. move each of its tape heads, independently, one cell to the left or right, or keep it stationary.

Initially, the input appears on the first tape and the other tapes are blank.

Consider a three-tape TM ($k = 3$). The case $k > 3$ can be defined similarly. Formally, a three-tape TM can be denoted by $M = (Q, \Sigma, \Gamma, \delta, q0, B, F)$ where $Q = [p,q,r]$ with p the state of the first tape, q the second tape and r the third tape, $\Sigma = [a,B,B]$, $\Gamma = [X,Y,Z]$ and δ is defined by $\delta([p,q,r],[X,Y,Z]) = ([p_i,q_i,r_i], [X_i,Y_i,Z_i],[A,B,C])$, [A,B,C] are the moves of the first, second, and third tape heads, respectively.

Multidimensional Turing Machines (MD-TM)

A MD-TM is a device that has the usual finite control, but the tape consists of k-dimensional array of cells infinite in all 2k directions, for some fixed k. Depending on the state and symbol scanned, the device changes state, prints a new symbol and moves the tape head in one of the 2k directions, either positively or negatively, along one of the k axes. Initially, the input is along one axis, and the head is at the left end of the input.

Multihead Turing Machine

A k-head TM has some fixed number k, read/write heads. The heads are numbered from 1 to k, and a move of the TM depends on the state and on the symbol scanned by each head. In one move, the heads may each move independently left, right or stationary.

Nondeterministic Turing Machines

A nondeterministic TM is denoted by $M = (Q, \Sigma, \Gamma, \Delta, q_0, B, F)$ where Δ the next move function is a mapping from $Q \times \Gamma$ to a subset of $Q \times \Gamma \times \{L, R, S\}$. The instantaneous descriptions (ID's) and the relations |- and |-* are defined as in the deterministic case. The only difference is that |- need not be single-valued.

Nondeterministic TM's are as powerful as the deterministic case (meaning they are equivalent). However, there are problems whose natural solutions are the nondeterministic case.

6. RECURSIVE AND RECURSIVELY ENUMERABLE LANGUAGES

Definition: We say that M **semidecides** L if for every string w the following is true: $w \in L$ if and only if M halts on input w, otherwise it does not halt. A language L is **recursively enumerable** if and only if there is a Turing Machine M that semidecides L.

Example: Let $L = \{w \in \{a,b\}^* | \text{ w contains an a}\}$. L is semidecided by the Turing Machine $= (\{q_0,q_1\}, \{0\}, \{0,B\}, \delta, q_0, B, \{q_1\})$ where δ is defined as:

States	Symbols		
	a	b	B
q_0	(q_1,a,S)	(q_0,b,R)	(q_0,B,R)

This Turing Machine scans the input one by one until an a is encountered in which case it halts. Otherwise it endlessly continuous to go the right or it does not halt.

Since the TM may not halt, we can say that the TM machine is not an algorithm for the problem being solved.

Definition: We say that M **decides** L if for every string w, M halts on input w whether w is in L or not. A language L is **recursive** if and only if there is a Turing Machine M that decides L.

Let ⓿ and ❶ be two symbols not in Σ. Then a language $L \subseteq \Sigma^*$ is recursive if and only if a TM that compute the function $g: \Sigma^* \to \{$⓿, ❶$\}$ can be constructed, where for each $w \in \Sigma^*$,

g(w) = ❶, if $w \in L$

g(w) = ⓿, if $w \notin L$.

If g is computed by a TM M, then M is said to decide L, or to be a decision procedure for L.

The symbol ❶ stands for "yes" or w is in L and ⓿ stands for "no" or w is not in L.

Since the TM decides a language, it can be reasonably thought of as an **algorithm** that performs correctly and reliably some computational task.

Example: Let $\Sigma = \{a\}$ and $L = \{w \in \Sigma \mid |w| \text{ is even}\}$. Then the following TM $M = (Q, \Sigma, \Gamma, \delta, q_0, \#, F)$ where $Q = \{q_0, q_1, \ldots, q_6\}$, $\Sigma = \{a\}$, $\Gamma = \{a,$ ❶, ⓿, $\#\}$, $f \in F$ and δ is given by:

States	Symbols			
	a	#	❶	⓿
q_0		(q_1,#,L)		
q_1	(q_2,#,S)	(q_4,#,R)		
q_2		(q_3,#,L)		
q_3	(q_0,#,S)	(q_6,#,R)		
q_4		(q_5,❶,S)		
q_5			(f,❶,R)	(f,⓿,R)
q_6		(q_5,⓿,S)		

The TM produces the following computations:

q_0#a^n# |-* # ❶#f, if n is even and

q_0#a^n# |-* # ⓿#f, if n is odd.

7. THE HALTING PROBLEM

Definition: Problems with no algorithm exists are called **undecidable** or **uncomputable** while problems with algorithms are called **decidable** or **computable**.

With this definition, we can say that the problem of recognizing recursive languages are decidable since an algorithm can be formulated for the problem. However, the problem of recognizing recursively enumerable languages may be decidable or not because if it is decidable, then the set of recursively enumerable languages is equivalent to the set of recursive languages. We show in this section that the set of recursively enumerable languages is not equivalent to recursive languages, i.e., there are recursively enumerable languages that are not recursive. Recognizing this set of recursively enumerable languages that are not recursive are what we call undecidable.

Now, we consider our first undecidable problem, the problem of testing whether a string w is accepted by a TM M. We define this as:

Member(M,w) = {Mw | TM M accepts input string w}.

Theorem: Member(M,w) is recursively enumerable.

Proof: We can construct a TM U that recognizes Member(M,w) as follows: U on input Mw (M is an encoding of the TM M and w is the input string):

1. Simulate M on input w
2. If M enters an accept state, U accepts. Otherwise, if M enters a reject state, U rejects.

Note that U loops on input Mw if M loops on w - thus it is possible for U not to stop.

This problem is called the **Halting Problem** because if there is only a way by which we can determine that M will not halt on w, then we can force U to stop and reject.

The TM U as describe is an example of the Universal TM which was used by Alan Turing in its proof of the undecidability of the Halting Problem. The machine is called universal because it is able to simulate any other TM from the description of the machine.

Theorem: Recognizing Member(M,w) is undecidable.

Proof: We assume that Member(M,w) is decidable. Being decidable, we can construct a TM H that decides Member(M,w), i.e., on input Mw, where M is a TM and w is a string, H halts and accepts if M accepts w. Furthermore, H halts and rejects if M fails to accept w. In summary, we say

$$H = \begin{cases} \text{accept, M accepts w} \\ \text{reject, M does not accept w} \end{cases}$$

Next, we construct another Turing Machine U that calls H. This new TM U calls H to determine what M does when the input to M is its own description M rather than w. The behavior of U is that it does the opposite of what H does. That is, it rejects if M accepts and accepts if M does not accept. U's operation on input M can be outlined as follows:

1. Run H on input MM
2. Output the opposite of what H outputs; if H accepts, reject and if H rejects, accept.

We can therefore write this as follows:

$$D = \begin{cases} \text{accept, if M does not accept M} \\ \text{reject, if M accepts M} \end{cases}$$

Now, we examine what happens when we run D on its own description. In this case we get:

$$D = \begin{cases} \text{accept, if D does not accept D} \\ \text{reject, if D accepts D} \end{cases}$$

Clearly, no matter what D does it is forced to do the opposite. Clearly this is a contradiction since D accepts when D does not accept. Hence, there is no such TM D and similarly no such TM H.

The existence of Member(W,w), a language that is recursively enumerable but not recursive, proved the following theorem.

Theorem: The class of recursive languages is a strict subset of the class of recursively enumerable languages.

A related problem, also called by some authors as the halting problem, is the problem of recognizing:

Halt(M,w) = {Mw | TM M halts on input string w}.

This problem is also undecidable and our proof is a reduction from Member(M,w).

Theorem: Reconizing Halt(M,w) is undecidable.

Proof: Assume that a TM H decides Halt(M,w). We can construct a TM U to decide Member(M,w), where U operates on input Mw (M is an encoding of the TM M and w is the input string):

1. Run TM H on input Mw
2. If H rejects, reject.
3. If H accepts, simulate M on w until it halts. If M accepted w, accept; if M rejects w, reject.

Clearly if H decides Halt(M,w), then U decides Member(M,w). But since Member(M,w) is undecidable, so is Halt(M,w).

8. CHURCH-TURING THESIS

Earlier we have illustrated that Turing machines can be designed to accept more complicated languages and arithmetic functions. Because Turing machines can carry out computations, we view Turing machines as a formal equivalent of the intuitive notion of an algorithm. That is, anything cannot be considered an algorithm unless a Turing machine can be constructed for it.

However, the use of a Turing machine is not only the approach to computations. There are other models of computations and these models of computations are:

1. Godel-Herbrand-Kleene (1936) – General recursive functions defined by means of an equation calculus (Kleene, 1952).
2. Church (1936) - λ-definable functions (Church, 1936).
3. Godel-Kleene (1936) - μ-recursive functions and partial recursive functions (Kleene, 1936).
4. Turing (1936) – Functions computable by finite state machines known as the Turing Machines (Turing 1936).
5. Post (1943) – Functions defined from canonical deduction systems (Post, 1943).
6. Markov (1951) – Functions given by certain algorithms over a finite alphabet (Markov, 1954).
7. Shepherdson-Sturgis (1963) – Unlimited Register Machine (URM)-computable functions (Shepherdson and Sturgis, 1963).

The **Church-Turing Thesis** states that each of the above proposals for a characterization of the notion of effective computability gives rise to the same class of functions, the class we call the Turing-computable functions. Thus, all the models of computations above are equivalent to each other and also equivalent to the Turing machine approach. Also, the intuitively and informally defined class of effectively computable functions coincides with the class of Turing-computable functions. This means that we can extend our concept of an algorithm as not only limited to those where a Turing machine can be constructed but also to those computable by other models of computations above.

This thesis is not a theorem (which is susceptible to mathematical proof); it has the status of a claim or belief that may be substantiated by evidence. We cannot hope to prove this thesis because we do not have an exhaustive list of models of computations. In fact, there might be somebody who can overthrow this thesis by showing the existence of a model of computation that can carry out computation that cannot be carried out by a Turing machine. However, this is unlikely because there are so many evidences that show that the thesis holds. Some of these evidences are:

1. The fundamental result itself is one evidence. Many independent proposals for a precise formulation of the intuitive idea have led to the same class of functions, which we have called Turing-computable functions.
2. A vast collection of effectively computable functions has been shown explicitly to belong to Turing-computatable functions.
3. An implementation of a program for a Turing-computable function is clearly an example of an algorithm.
4. No one has ever found yet a function that would be accepted as computable in the informal sense, that does not belong to Turing-computable function.

On the basis of these evidences and of their experiences, most mathematicians are led to accept the Church-Turing Thesis.

In order to illustrate the flavor of other models of computations, we discuss unrestricted grammars, μ-recursive functions and unlimited register machines. These three models of computations were shown to be equivalent to the Turing machine approach.

9. REFERENCES

[1] Church, A. An unsolvable problem of elementary number theory, Am. J. Math. 58, 1936, 345-363.

[2] Cohen, D.I.A Introduction to Computer Theory, John Wiley and Sons, Philippine Reprint, 1993.

[3] Cutland, N.J. Computability: An Introduction to recursive function theory, Cambridge University Press, 1980.

[4] Davis, MD, Sigal, R, and Weyuker, EJ. Computability, Complexity, and Languages, Academic Press, 1994.

[5] Hopcroft, JE and Ullman, JD. Introduction to Automata Theory, Languages, and Computations, Addison-Wesley, 1979.

[6] Kelly, D. Automata and Formal Languages: An Introduction, Prentice-Hall, 1995.

[7] Kleene, S.C. General recursive functions of natural numbers, Mathematiche Annalen, 112, 1936, 727-742.

[8] Kleene, S.C. Introduction in Metamathematics, Van Nostrand, Princeton, 1952.

[9] Lewis, HR and Papadimitriou CH. Elements of the Theory of Computations (1st Edition), Prentice Hall, 1981.

[10] Lewis, HR and Papadimitriou CH. Elements of the Theory of Computations (2nd Edition), Prentice Hall, 1998.

[11] Markov, A.A. The Theory of Algorithms. English translation National Science Foundation, 1954.

[12] Martin, J.C. Introduction to Languages and Theory of Computation, McGraw-Hill, 1991.

[13] Post, E. Formal reductions of the general combinatorial decision problem, Am. J. Math 65, 1943, 197-215.

[14] Shepherdson, J.C. and Sturgis, H.E. Computatbility of recursive functions, J. Assoc. Comput. Machinery 10, 1963, 217-255.

[15] Sipser, M. Introduction to the Theory of Computation, PWS Publishing Co., 1997.

[16] Turing. A.M. On computable numbers, with application to the Entscheidungs problem, Proc. London Math. Soc. 42 & 43, 1936, 230-265 (vol 42), 544-546 (vol 43).