

TM Computability

We will use TMs not just as language recognizers, but in many ways. But first some conventions.

1. We will surround the input string by a blank ($\#$) on each end and write the result on the leftmost cells of the tape.
2. The tape head is positioned initially at the cell containing the blank marking the right end of the input sequence.
3. The finite control initially starts in the initial state $(S, \#W\#)$.

Slide Lecture 14 –414

First consider the Turing Machine Operation
as a string to string mapping.

Let Σ_0 and Σ_1 be alphabets not containing $\#$.

Let f be a function, $f : \Sigma_0^* \rightarrow \Sigma_1^*$.

A $TM = (K, \Sigma, \delta, s)$ computes f if

$$\Sigma_0, \Sigma_1 \subseteq \Sigma$$

and for all $w \in \Sigma_0^*$ if $f(w) = u$ then :

$$(s, \#w\#) \vdash_M^* (h, \#u\#)$$

Such functions are called *Turing Computable Functions*.

Slide Lecture 14 –415

Example:

Let $\Sigma_0 = \Sigma_1 = \{a, b\}$ and let $f : \Sigma_0^* \rightarrow \Sigma_1^*$
 be defined as follows:
 For each $w \in \Sigma_0^*$, $f(w) = w'$, where w' maps $a \rightarrow b$ and $b \rightarrow a$.

We construct a Turing Machine that scans input right to left
 changing a's to b's (and b's to a's) until the leftmost blank
 symbol is scanned.

Slide Lecture 14 –416

Turing Machines can also be viewed as “Decision Procedures.”

Let Σ_0 be an alphabet, $\# \notin \Sigma_0$.
 Let \ddot{Y} and \ddot{N} be special symbols.

Language L is *Turing Decidable* IFF the function

$$X_L : \Sigma_0^* \rightarrow \{\ddot{Y}, \ddot{N}\}$$

is Turing Computable, where
 for each string $W \in \Sigma_0^*$

$$X_L(W) = \begin{cases} \ddot{Y} & \text{if } W \in L \\ \ddot{N} & \text{if } W \notin L \end{cases}$$

Slide Lecture 14 –417

Turing Machines can also be viewed as “Language Acceptors.”

Let Σ_0 be an alphabet, $\# \notin \Sigma_0$.

A TM denoted M accepts a string $W \in L, L \subseteq \Sigma_0^*$ if M halts on an input sequence W in an accepting configuration.

For $W \notin L$ the machine’s behavior is undefined.

Slide Lecture 14 –418

Turing Machine Computations

1. Decidable languages
2. Language Acceptors
3. String to String
4. Natural number to Natural number

Slide Lecture 14 –419

Enhancements to Turing Machines:

Using 2 read-write tapes = tape infinite in both directions.

	-1	0	1		
--	----	---	---	--	--

Input can be reconfigured as :

0	1			
-1				

Slide Lecture 14 –420

Does a TM with 2 tracks have the same power as a TM with 1 track?

TM's with $K \geq 2$ tracks (or tapes) fail to alter the computability of the TM.

We simulate a second track at some distance K to the right on the first track. We can dynamically move the contents of the second simulated tape further to the right if needed.

Since 1 tape can simulate 2 (or more) tapes, this does not alter the computability of the TM.

Slide Lecture 14 –421

We often use multiple tapes to simplify a computation.

Example of a 3-tape machine that accepts $L = \{a^n b^n c^n \mid n \geq 0\}$

OUR MACHINES:

#	W	#		
#				
#				

Slide Lecture 14 –422

The initial configuration is: $\{s, \#W\#, \#, \#\}$

The machine operates as follows:

1. Scan W from right to left. Write c 's to tape 3 and write b 's to tape 2.
2. Move all 3 tapes heads to the leftmost cells.
($q, \#a..a\#, \#b..b\#, \#c..#\$)
3. Scan the 3 tapes *Synchronously*...

Slide Lecture 14 –423

The same problem can be easily encoded on a 1 tape machine:

1. If less than 3 *c*'s appear on the tape, check for termination cases *aabbc* and *abc*.
2. Going right to left, erase 3 *c*'s.
3. Going right to left, write 2 *c*'s over rightmost *b*'s.
4. Write a *b* over rightmost *a*.
5. Repeat.

Slide Lecture 14 –424

Consider *#aaabbbccc#*

1. *#aaabbb#####* Erase 3 *c*'s.
2. *#aaabcc#* Write 2 *c*'s over *b*'s.
3. *#aabcc#* Write *b* over rightmost *a*.

Slide Lecture 14 –425

Other machines equivalent to TM's:
Automata with two pushdown stacks!

If we think of two stacks as a Turing machine tape,
then we have the ability to

1. Move in either direction and
2. Rewrite any symbol.

Post Machines are another analog of the TM.
Its operations treat the input as lists
and uses flow charts to model computation.

Slide Lecture 14 –426

Nondeterministic Turing Machines

For any **Nondeterministic TM** there exists an equivalent
Deterministic TM that simultaneously pursues
all possible nondeterministic computations.

Proof by construction:

1. Record M's initial configuration on the second tape of a 2-tape TM.
2. For each configuration perform each possible move of M and record the $c \geq 0$ new configurations on the alternate tape. (This is a bread-first search of the space of possible configurations).
3. If a halting configuration is detected, then halt.
If $c = 0$, M hangs; Else repeat step 2.

Slide Lecture 14 –427

Universal Turing Machines: Interpretive Computers.

The *Universal Turing Machine* (denoted by μ) is a TM which can interpret (i.e., “simulate”) the operation of any other TM.

This is analogous to the construction of a general purpose stored program computer capable of interpreting (executing) any program of a specified form.

Slide Lecture 14 –428

1. Initially μ 's tape contains an encoded description of a TM, ΔT , and some input X.

X	\$	ΔT	#	
---	----	------------	---	--

2. μ will simulate T on input X and compute the function $T(X)$, and accept X iff T accepts X.

Slide Lecture 14 –429

The key to demonstrate the existence of such a μ lies in the encoding, ΔT .

We assume that the finite set of states for any TM is a subset of the fixed, countably infinite set.

$$K_\infty = q_1, \dots, q_\infty.$$

Likewise we define the set of all possible alphabet symbols to be:

$$\Sigma_\infty = a_1, \dots, a_\infty.$$

From this we pick a finite alphabet.

Slide Lecture 14 –430

We will use a 3-tape TM for μ , and define the functions of each tape as follows:

Tape 1: Represents T's read/write tape.

Tape 2: Holds ΔT the encoded description of T.

ΔT is a collection of "blocks" where the i^{th} block will contain all moves from the i^{th} state of T (i.e., state q_i).

Tape 3: An encoding of T's current state.

Slide Lecture 14 –431

The Universal Turing machine matches
the symbol under the tape head (Tape 1)
and the current state (Tape 3)
against the state-input pairs on Tape 2.
It then updates the current state (Tape 3)
and rewrites the input or moves the head (on Tape 1).

Slide Lecture 14 –432

Example:
 $L = \{W \in \{a_1, a_2\}^* \mid W \text{ contains at least 1 } a_1\}.$

Define T:

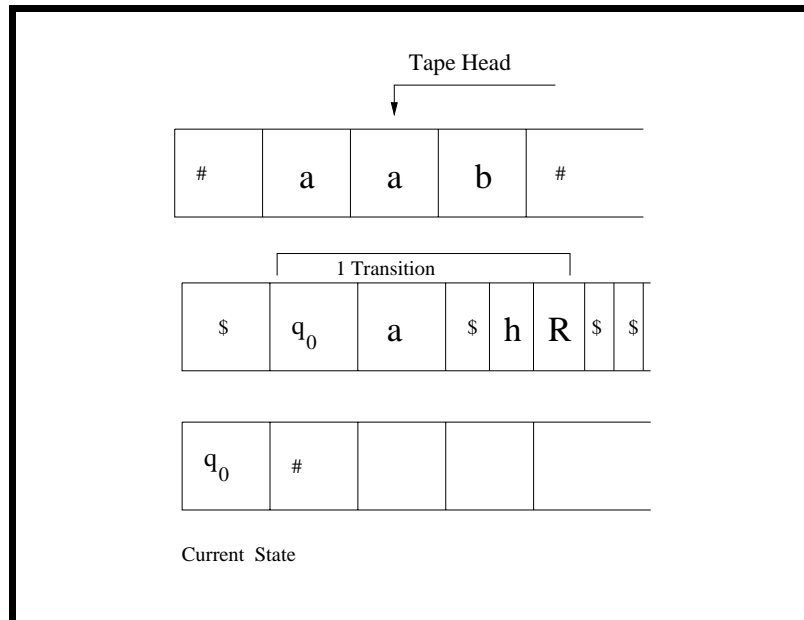
$$\Sigma_T = a_1, a_2, \#.$$

$$K_T = q_0,$$

$$S = q_0$$

$$\Delta_T = ((q_0, a_1), (h, R)), ((q_0, a_2), (q_0, L)), ((q_0, \#), (q_0, L)).$$

Slide Lecture 14 –433



Slide Lecture 14 –434

The Simulation:

1. Given the current state and current symbol being scanned, scan Tape 2 for a move.
2. Given the move, update the current state on the Tape 3 and either move the Tape 1 head left or right, or write a symbol as indicated by the move.
3. If "h" is written on Tape 3, halt, otherwise goto step 1.

Slide Lecture 14 –435

A language that is accepted by a Turing machine is said to be *Recursively Enumerable* (R.E. – not regular expression).

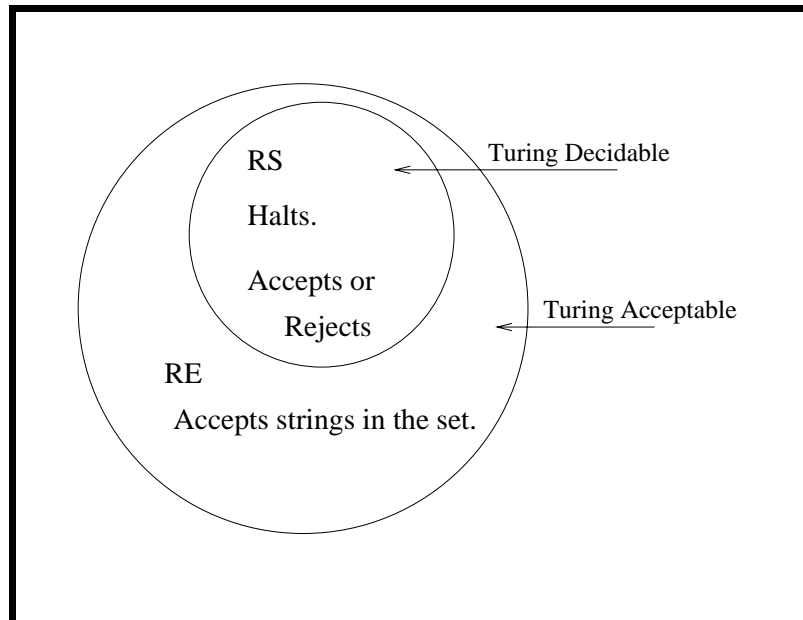
A subset of R.E. sets is called the *Recursive Set* (R.S.).

A recursive set corresponds to a *Turing Decidable* language.

Every *Turing Decidable* language is also *Turing Acceptable*.

The reverse is not true.

Slide Lecture 14 –436



Slide Lecture 14 –437

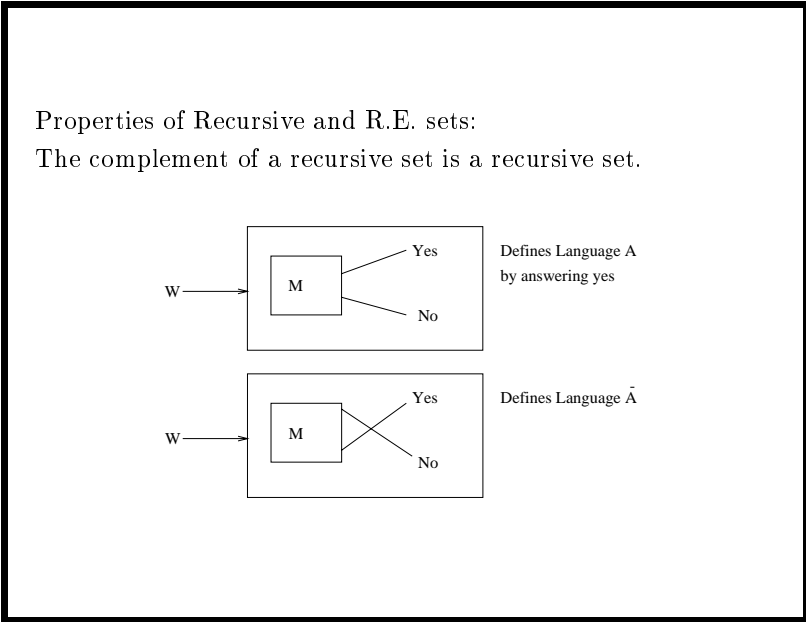
Every CFL is a recursive set.

TM's processing recursively enumerable languages act as *Acceptors*.

TM's processing recursive sets can be used as *Deciders*, or as *decision procedures*.

Theoretical constructions, such as P and NP, are based on the use of Turing Machines as decision procedures.

Slide Lecture 14 –438



Slide Lecture 14 –439

Denote the complement of A by \overline{A} .

B is the complement of A if

$$A \cup B = \Sigma^*$$

$$A \cap B = \emptyset$$

For instance if $\Sigma = 1, 2, 3, 4, 5, 6, 7, 8, 9, 0$

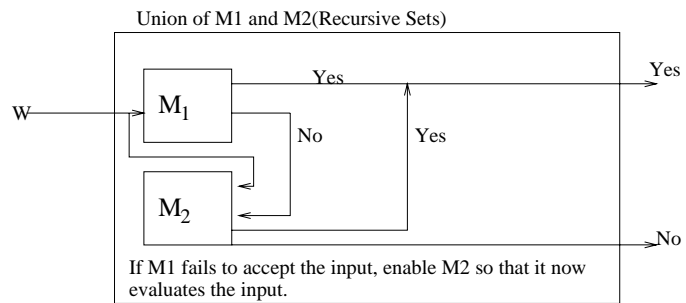
then $\Sigma^* =$ All positive integers.

If $A = N \in \Sigma^* \mid N \leq 200$

then $\overline{A} = N \in \Sigma^* \mid N > 200$

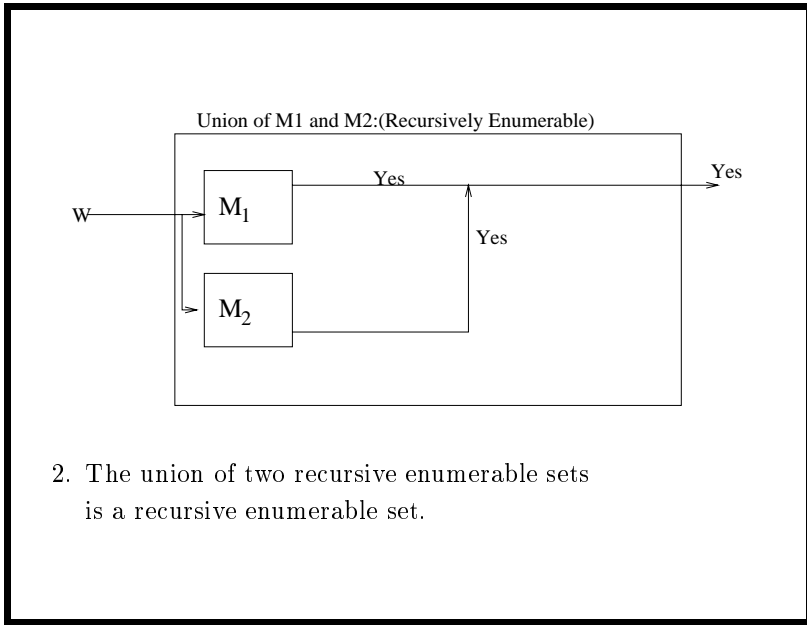
Slide Lecture 14 -440

Theorems:

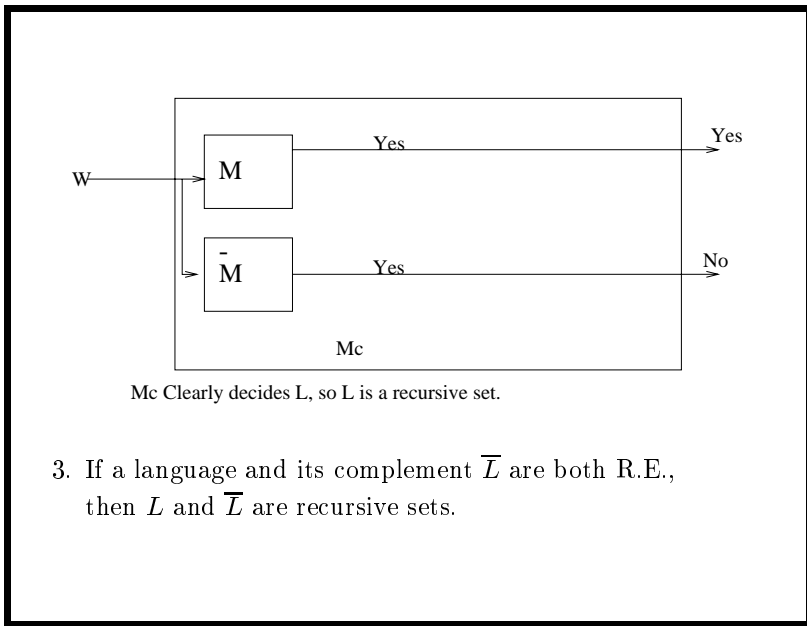


1. The union of two recursive sets is a recursive set.

Slide Lecture 14 -441



Slide Lecture 14 -442



Slide Lecture 14 -443

Summary:

Let L and \bar{L} be complement languages.

One of the following must be true.

1. Both L and \bar{L} are recursive sets.
2. Neither L nor \bar{L} are recursively enumerable.
3. Either L or \bar{L} is recursively enumerable, but not recursive, and the other is not recursively enumerable.