Lecture 15: Computing with Turing Machines

Hereafter, we shall assume that the initial configuration of M on input w is $(s, \triangleright\underline{\sqcup}w)$.

Let $M = (K, \Sigma, \delta, s, H)$ be a Turing Machine such that

$$H = \{y, n\}$$

where y means "yes", n means "no".

- $\Sigma_0 \subseteq \Sigma - \{\sqcup, \triangleright\}$ is the *input alphabet* of M.
- Let $L \subseteq \Sigma_0^*$ be a language.
- On input w:
  - If M halts at y, we say M **accepts** w.
  - If M halts at n, we say M **rejects** w.
  - Note that it is possible that M never halts on w.

We say M **decides** L if for any string $w \in \Sigma_0^*$, the following is true:

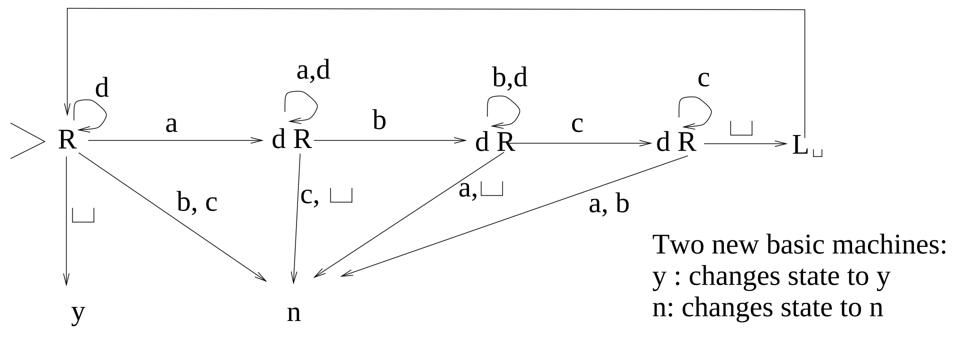
- if $w \in L$, then M accepts w (i.e. M halts at state y).
- if $w \notin L$, then M rejects w (i.e. M halts at state n).

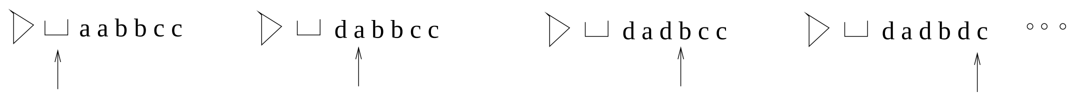
This means, M *halts on all input* and it correctly halts at either y or n depending on whether or not the input w is in L.

Definition:
A language L is said to be **recursive** (also known as Turing-decidable) if and only if there is a TM that **decides** it.

Example: Prove that the following language is recursive.

$$L = \{a^n b^n c^n : n \geq 0\}$$





An informal description:

1. Start from the left end of the string.
2. Move to the right in search of an a; if found, replace it by a d; if b or c is found instead, *reject*.
3. Move further right in search of a b; if found, replace it by a d; if not found, *reject*.
4. Move further right in search of a c; if found, replace it by a d; if not found, *reject*.
5. If all input has been replaced by d's, *accept*.
6. Return to the left end of the input, go to 2.

Examples of recursive languages

Example: Prove that the following language is recursive:

$$L = \{a^i b^j c^k : i \times j = k \text{ and } i, j, k \geq 1\}$$

An informal description:

1. Scan the input from left to right to be sure that it is a member of $a^+b^+c^+$; otherwise *reject* it.
2. Return the head to the left-hand end of the tape.
3. Cross off an a and scan to the right until a b occurs. Then, move back and forth between the b's and the c's, each time crossing off one b and one c until all b's are gone.
4. Restore the crossed off b's and repeat 3 if there is another a to cross off. if all a's are crossed off, check whether all c's are also crossed off. If yes, *accept*; otherwise, *reject*.

"Rules" for describing a TM informally (for this course):

1. You can use all basic machines discussed in class as black boxes (e.g., copying machine, shifting machine, ...).
2. Describe what the TM does on the tape in each step, while making sure that all the "control" information needed for executing these steps can be encoded in the state of the TM, i.e., there is a constant amount of such information.

We can think of the process of converting the input to the output by a TM as evaluating a function.

Definition: Let $M = (K, \Sigma, \delta, s, \{h\})$ be a TM, and let $\Sigma_0 = \Sigma - \{\sqcup, \triangleright\}$. Suppose M halts on input $w \in \Sigma_0^*$, and $(s, \triangleright\underline{\sqcup}w) \vdash_M^* (h, \triangleright\underline{\sqcup}y)$ for some $y \in \Sigma_0^*$. Then y is called the **output** of M on w, and is denoted $M(w)$.

$M(\cdot)$ is a function from Σ_0^* to Σ_0^* by the above definition.

Definition: A function $f : \Sigma_0^* \rightarrow \Sigma_0^*$ is **recursive** if there exists a TM M such that $M(w) = f(w)$ for all $w \in \Sigma_0^*$. Such an M is said to **compute** f.

Example: The **successor** function. See Example 4.2.3 in the textbook on page 197.

Recursively Enumerable Languages

Let $M = (K, \Sigma, \delta, s, H)$ be a Turing Machine
where $H = \{h\}$.

Let $\Sigma_0 \subseteq \Sigma - \{\sqcup, \rhd\}$ be an alphabet.

Let $L \subseteq \Sigma_0^*$ be a language.

We say M **semidecides** L if for any string $w \in \Sigma_0^*$, the following is true:

- if $w \in L$, then M halts on input w
- if $w \notin L$, then M does not halt (denoted as $M(w) = \nearrow$).

The above definition can be equivalently written as:
– $w \in L$ if and only if $M(w) \neq \nearrow$.

Turing Machines that semidecides languages

Given a string w and a Turing machine M that semidecides a language L. If we run w on M, it may lead to a halt right away, or we may have to wait for a long time. Even if we wait for seven billion execution steps, if w has not been accepted, it still might be accepted eventually. Or it might be in a loop and we shall never get an answer.

Definition:

A language L is said to be **recursively enumerable** (a.k.a. Turing-acceptable) if and only if there is a TM that **semidecides** L.

Note that this is a valid definition, although it does not give you a way to determine the membership of a string.

Examples

Example: Show that the following language is recursively enumerable:

$$L = \{w \in \{a, b\}^* : w \text{ contains at least one } a\}$$

$$\begin{tikzcd} R \arrow[loop right, "\overline{a}"] \end{tikzcd}$$

This machine simply scans right until an a is encountered, and then halts (recall: the basic TM R simply moves right one position and halts). If w does not contain a, M goes on forever onto the blanks. Thus, the above TM semidecides L.

This language is also recursive since the TM below decides it.

$$\begin{tikzcd} R \arrow[loop above, "\sigma \neq a \text{ and} \sigma \neq \sqcup"] \arrow[rr, "\sqcup"] \arrow[dr, "a"'] & & n \\ & y & \end{tikzcd}$$

Note: All the languages you have seen so far in this course are recursive. Examples of languages that are not recursive or not recursively enumerable are complicated. We'll see some later.

- TMs that decide languages are algorithms.
- TMs that semidecide languages are NOT algorithms.

Theorem 1 *If L is recursive, then L is recursively enumerable.*

Proof:
Idea: Since L is recursive, there is a machine M that *decides* L. Make the rejecting state of M (i.e., the state n) a non-halting state and make the machine loops forever on that state.

- Let $M = (K, \Sigma, \delta, s, \{y, n\})$ decides L.
- We can construct a TM that semidecides L as follows: $M' = (K, \Sigma, \delta', s, \{y\})$ where δ' is :

  $$\delta'(q, \sigma) = \delta(q, \sigma) \text{ for all } q \neq n \text{ and } \sigma \in \Sigma$$

  $$\delta'(n, \sigma) = (n, \sigma) \text{ for all } \sigma \in \Sigma$$

  That is, the machine indefinitely stays at the state n and writes back σ without halting.

  – If $w \in L$,

    $\Rightarrow M$ will halt in state y.
    $\Rightarrow M'$ will also halt in state y.

  – If $w \notin L$,

    $\Rightarrow M$ will halt in state n.
    $\Rightarrow M'$ will loop forever in state n. (i.e., n is no longer a halting state)

Question:
If L is recursively enumerable, is L always recursive? No (see examples later).

Theorem 2 *If L is context-free, then L is recursive.*

Sketch:
The objective is to show that there is a TM that *decides* L.

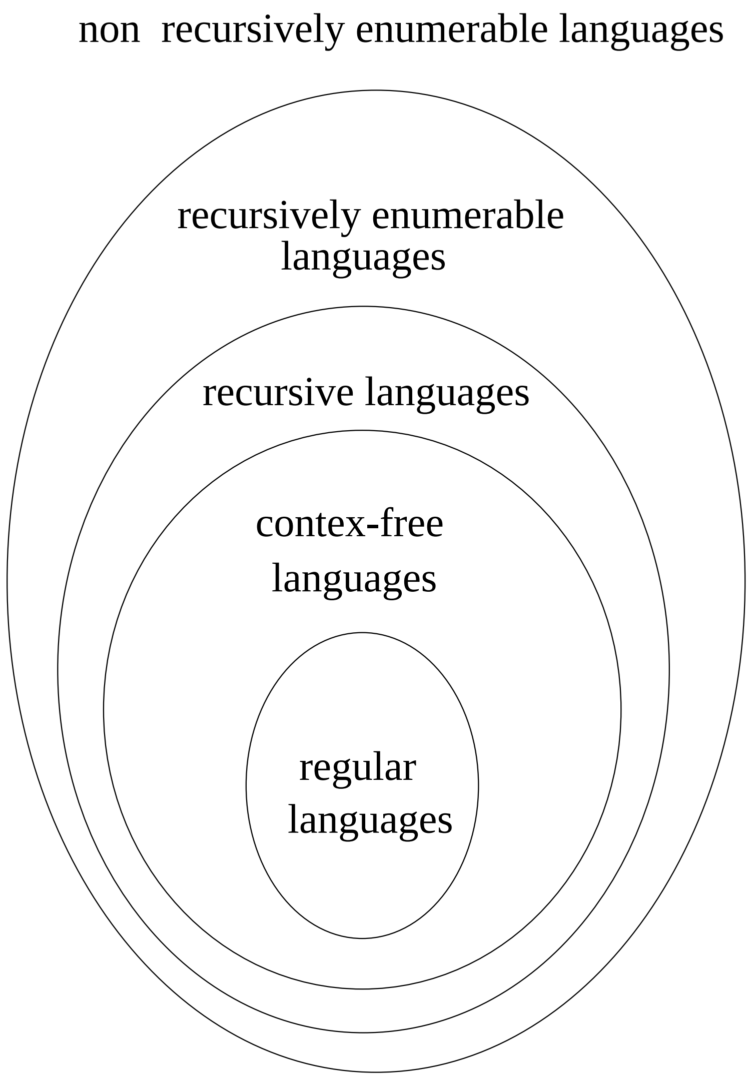
- First, we state without proof the following theorem:
  Every CFG can be converted to the Chomsky normal form. Every rule of a CFG in Chomsky Normal form is of the form either $X \rightarrow YZ$ or $X \rightarrow \sigma$, where X, Y, Z are nonterminals, and σ is a terminal. In addition, $S \rightarrow e$ is permitted only if $e \in L$.

- Since L is context-free, there exists a CFG that generates L. Convert the CFG into Chomsky Normal Form. Then, any string w of length n would have derivations of at most $2n - 1$ steps (without using $S \rightarrow e$ would take exactly $2n - 1$ steps).

- If n=0, the TM lists all derivations of 1 step; if $n > 0$, it lists all derivations of at most $2n - 1$ steps. If any of these derivations yield w, *accept*; if not, *reject*.

Example of CNF
$S \rightarrow aSb|e$ can be converted to
$S \rightarrow AX|e$
$X \rightarrow SB$
$A \rightarrow a$
$B \rightarrow b$

Hierarchy of languages


non recursively enumerable languages
recursively enumerable languages
recursive languages
contex-free languages
regular languages

Theorem 3 *If L is recursive, then $\overline{L}$ is recursive (i.e., the set of recursive languages is closed under complementation).*

(Other closure properties will be discussed later after we learn about multi-tape TMs and nondeterministic TMs)

Proof:

- Let $M = (K, \Sigma, \delta, s, \{y, n\})$ be the TM that decides L.
- We can construct a TM M' that decides L' by reversing the roles of states y and n in M. That is:
  $M' = (K, \Sigma, \delta', s, \{y, n\})$

$$\delta'(q,\sigma) = \begin{cases} (n,\gamma) & \text{if } \delta(q,\sigma) = (y,\gamma) \\ (y,\gamma) & \text{if } \delta(q,\sigma) = (n,\gamma) \\ (p,\gamma) & \text{if } \delta(q,\sigma) = (p,\gamma) \text{ and } p \neq y, p \neq n \end{cases}$$

Question:

Is the complement of a recursively enumerable language always recursively enumerable? No (example later).