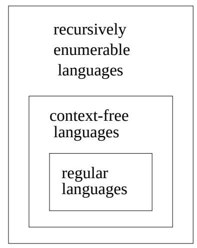
CPS 140 - Mathematical Foundations of CS
Dr. S. Rodger
Section: Recursively Enumerable Languages (handout)

Definition: A language L is *recursively enumerable* if there exists a TM M such that L=L(M).

if $w \in L$? if $w \notin L$?



Definition: A language L is *recursive* if there exists a TM M such that L=L(M) and M halts on every w$\in \Sigma^+$.

Enumeration procedure for recursive languages

To enumerate all w$\in \Sigma^+$ in a recursive language L:

- Let M be a TM that recognizes L, L = L(M).
- Construct 2-tape TM M'
  Tape 1 will enumerate the strings in Σ^+
  Tape 2 will enumerate the strings in L.
  - On tape 1 generate the next string v in Σ^+
  - simulate M on v
    if M accepts v, then write v on tape 2.

Enumeration procedure for recursively enumerable languages

To enumerate all w$\in \Sigma^+$ in a recursively enumerable language L:

Repeat forever

- Generate next string (Suppose k strings have been generated: $w_1, w_2, ..., w_k$)
- Run M for one step on w_k
  Run M for two steps on w_{k-1}.
  ...
  Run M for k steps on w_1.
  If any of the strings are accepted then write them to tape 2.

Theorem For any nonempty Σ, there exist languages that are not recursively enumerable.

Proof:

- A language is a subset of Σ^*.

  The set of all languages over Σ is

Theorem There exists a recursively enumerable language L such that $\bar{L}$ is not recursively enumerable.

Proof:

- Let $\Sigma = \{a\}$

  Enumerate all TM's over Σ:

	a	aa	aaa	aaaa	aaaaa	...
L(M_1)	0	1	1	0	1	...
L(M_2)	1	0	1	0	1	...
L(M_3)	0	0	1	1	0	...
L(M_4)	1	1	0	1	1	...
L(M_5)	0	0	0	1	0	...
...						

The next two theorems in conjunction with the previous theorem will show that there are some languages that are recursively enumerable, but not recursive.

Theorem If languages L and $\bar{L}$ are both RE, then L is recursive.

Proof:

- There exists an M_1 such that M_1 can enumerate all elements in L.

  There exists an M_2 such that M_2 can enumerate all elements in $\bar{L}$.

  To determine if a string w is in L or not in L perform the following algorithm:

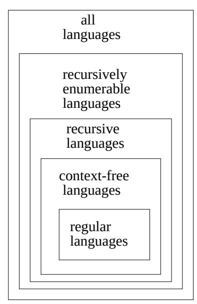
Theorem: If L is recursive, then $\bar{L}$ is recursive.

Proof:

- L is recursive, then there exists a TM M such that M can determine if w is in L or w is not in L. M outputs a 1 if a string w is in L, and outputs a 0 if a string w is not in L.

  Construct TM M' that does the following. M' first simulates TM M. If TM M halts with a 1, then M' erases the 1 and writes a 0. If TM M halts with a 0, then M' erases the 0 and writes a 1.

Hierarchy of Languages:



Definition A grammar G=(V,T,R,S) is *unrestricted* if all productions are of the form

$$u \rightarrow v$$

where u ∈(V∪T)$^+$ and v ∈(V∪T)*

Example:

Let G=({S,A,X},{a,b},R,S), R=

S → bAaaX
bAa → abA
AX → ϵ

Example Find an unrestricted grammar G s.t. L(G)=$\{a^n b^n c^n | n > 0\}$

G=(V,T,R,S)

V={S,A,B,D,E,X}

T={a,b,c}

R=

1) S $\rightarrow$ AX	7) Db $\rightarrow$ bD
2) A $\rightarrow$ aAbc	8) DX $\rightarrow$ EXc
3) A $\rightarrow$ aBbc	9) BX $\rightarrow$ ϵ
4) Bb $\rightarrow$ bB	10) cE $\rightarrow$ Ec
5) Bc $\rightarrow$ D	11) bE $\rightarrow$ Eb
6) Dc $\rightarrow$ cD	12) aE $\rightarrow$ aB

There are some rules missing in the grammar.

To derive string aaabbbccc, use productions 1,2 and 3 to generate a string that has the correct number of a's b's and c's. The a's will all be together, but the b's and c's will be intertwined.

$$\text{S} \Rightarrow \text{AX} \Rightarrow \text{aAbcX} \Rightarrow \text{aaAbcbcX} \Rightarrow \text{aaaBbcbcbcX}$$

Theorem If G is an unrestricted grammar, then L(G) is recursively enumerable.

Proof:

- List all strings that can be derived in one step.

  List all strings that can be derived in two steps.

Theorem If L is recursively enumerable, then there exists an unrestricted grammar G such that L=L(G).

Proof:

- L is recursively enumerable.
  $\Rightarrow$ there exists a TM M such that L(M)=L.
  M = $(K, \Sigma, , , \delta, q_0, B, F)$
  $q_0 w \overset{*}{\vdash} x_1 q_f x_2$ for some $q_f \in$F, $x_1, x_2 \in , ^*$

Construct an unrestricted grammar G s.t. L(G)=L(M).

$S \overset{*}{\Rightarrow} w$

Three steps

1. $S \overset{*}{\Rightarrow} B \ldots B\#xq_f yB \ldots B$
   with x,y∈ , * for every possible combination
2. $B \ldots B\#xq_f yB \ldots B \overset{*}{\Rightarrow} B \ldots B\#q_0 wB \ldots B$
3. $B \ldots B\#q_0 wB \ldots B \overset{*}{\Rightarrow} w$

Definition A grammar G is *context-sensitive* if all productions are of the form

$$x \to y$$

where $x, y \in (V \cup T)^+$ and $|x| < |y|$

Definition L is context-sensitive (CSL) if there exists a context-sensitive grammar G such that L=L(G) or L=L(G) ∪ $\{\epsilon\}$.

Theorem For every CSL L not including ϵ, ∃ an LBA M s.t. L=L(M).

Theorem If L is accepted by an LBA M, then ∃ CSG G s.t. L(M)=L(G).

Theorem Every context-sensitive language L is recursive.

Theorem There exists a recursive language that is not CSL.