October 27

\* Neither FSM nor PDA can be regarded as truly general models for computers

They cannot recognize some simple languages {a^n b^n c^n: n >= 0}.

Turing machines are more general than FSM and PDA

Turing machines are very similar in their workings to FSM and PDA

Although primitive, attempts to strengthen TM do not give any additional

capabilities

- many tapes

- fancier memory (RAM)

Turing machines form a stable and maximal class of computational devices

in terms of the types of computations they can perform

\* Turing Machines

A TM is an automata with a one-way infinite input tape

+--+--+--+---+--+--+--+

|a1|a2|a3|...|an| | |... INFINITE

+--+--+--+---+--+--+--+

\\_\_\_^\_\_\_\_\_\_\_\_\_\_/ ^ ^

| | | |

| | BLANKS...

| +--------------- INPUT

| READ/

| WRITE

+--------------------+

| FINITE CONTROL |

| |

| <--------> |

+--------------------+

O O

TM M = (K, Sigma, delta, s, H)

K: finite set of states

Sigma: alpha containing "blank symbol" B and "left end symbol" $

s: s in K is the initial state

H: H <= K is the set of halting states

delta: delta:(K-H)xSigma -> KxSigmax{<-, ->, S}) is the

transition function such that

a) forall q in K-H, if delta(q,$) = (p,b,m) then b neq <-

b) forall q in K-H and a in Sigma,

if delta(q,a) = (p,b,m) then b neq $

In one step, M can read a symbol, and write a new one in its place,

change state, and move the read/write head

(note this is different from the book, where M either writes a new

symbol or moves, but not both).

delta(k,a) = (p,b,m) means "From state s scanning "a", replace "a" with

"b", change to state p, and move m.

Note that when "$" is seen we never overwrite the left end symbol but

simply recognize it and move right.

Note that delta is not defined on states in H - when the machine reaches

a halting state, the operation stops.

A "configuration" of a TM M = (K, Sigma, delta, s, H) is a member of

K x (Sigma\* union $) x (Sigma\*(Sigma - {B}) union {e})

Configurations represent current state, tape up to and including head,

tape to the right of the head)

All configurations start with left end symbol

No configurations end with a blank unless the blank is being scanned.

A configuration whose state is in H is a "halted configuration".

Simplified notation: (q, wa, u) can be written as (q, w\_au)

(\_a is underline a).

\* Example

M = ({q0, q1, q2, q3, q4}, {0,1}, delta, q0, {q4})

Scanning

State| 0 1 X Y B

-----+---------------------------------------------

q0 | (q1,X,R) - - (q3,Y,R) -

q1 | (q1,0,R) (q2,Y,L) - (q1,Y,R) -

q2 | (q2,0,L) - (q0,X,R) (q2,Y,L) -

q3 | - - - (q3,Y,R) (q4,B,R)

q4 | - - - \_ -

This machine ends up in state q4 whenever the input is

of the form 0^i 1^i, i>= 1.

(q0, $\_000111) |- (q1, $X\_00111) |- (q1, $X0\_0111) |- (q1, $X00\_111) |-

(q2, $X0\_0Y11) |- (q2, $X\_00Y11) |- (q2, $\_X00Y11) |- (q0, $X\_00Y11) |-

(q1, $XX\_0Y11) |- (q1, $XX0\_Y11) |- (q1, $XX0Y\_11) |- (q2, $XX0\_YY1) |-

(q2, $XX\_0YY1) |- (q2, $X\_X0YY1) |- (q0, $XX\_0YY1) |- (q1, $XXX\_YY1) |-

(q1, $XXXY\_Y1) |- (q1, $XXXYY\_1) |- (q2, $XXXY\_YY) |- (q2, $XXX\_YYY) |-

(q2, $XX\_XYYY) |- (q0, $XXX\_YYY) |- (q3, $XXXY\_YY) |- (q3, $XXXYY\_Y) |-

(q3, $XXXYYY\_B) |- (q4, $XXXYYY\_B)

The symbol |- means "yields in one step".

A "computation" by M is a sequence of configurations C0, C1, ..., Cn, for

some n >= 0 s.t. C0 |-M C1 |-M C2 |-M ... |-M Cn.

The computation is of "length" n or equivalently has n "steps", and we can

say C0 |-nM Cn.

\* Acceptance by a TM

Let M be a TM such that H = {y,n}. These halting states correspond to

"yes" and "no". Any halting configuration containing y is an "accepting

configuration", and any halting configuration containing n is a "rejecting

configuration".

We say M "accepts" input w if (s, $w) yields an accepting configuration.

M "rejects" input w is (s, $w) yields a rejecting configuration.

M "decides" a language L if for every w in L M accepts w and for every

w not in L M rejects w.

If a TM decides a language L then L is "recursive".

For our example, let q4 be the "y" state. Then the input string

000111 is accepted by M, and M decides {0^i 1^i: i>=1}.

Note that if M does not accept some w, we may not be able to determine

this...

M may reject by never halting.... We may not be able to determine this by

inspecting M, w, and/or simulating M or w. Perhaps if M ran longer - it

would accept?

Recursively Enumerable Languages (r.e.)

= {L: there exists M s.t. L = L(M)}

M "semidecides" L if for any string w in Sigma\* w in L iff M halts

on input w.

Recursive Languages (nice ones)

= {L: there xists M s.t. L = L(M) and further, for all w in Sigma\*,

M halts on input w, i.e., there exists alpha1, alpha2 s.t. either

iw |-\* alpha1 k\_accept alph2, or iw |-\* alpha1 k\_reject alpha2}

If L is recursively enumerable we can determine if w in L, but not

necessarily if w not in L. Thus we do not have an algorithm for

determining whether or not w in L.

If L is recursive then there exists an algorithm (TM) for testing membership

in L.

Example

L = {a^n b^n c^n: n >= 1} is not context-free, but it is recursive.

Go back and forth "marking" "a"s, "b"s, and "c"s one character at a time.

A represents marked "a"

B represents marked "b"

C represents marked "c"

At any time, the tape will contain something like

AAAaaa...BBBbbb...CCCccc...

delta(k0,a) = (k1, A, R) Mark single "a" by changing it to A

delta(k1,a) = (k1,a,R)

delta(k1,B) = (k1,B,R) Find next "b" while moving right

delta(s1,b) = (s2,B,R) Mark the "b" by changing it to B

delta(k2,b) = (k2,b,R)

delta(s2,C) = (s2,C,R) Find next "c" while moving right

delta(k2,c) = (k3,C,L) Make the "c"

delta(k3,X) = (k3,X,L) For X in {C,B,b,a}...go back to first unmarked "a"

delta(k3,A) = (k0,A,R) We moved one cell too far to the left. Move one

cell right; start at k0 to begin another round.

Show graph notation

+--+ a/A,R +--+

|k0|------->|k1| etc.

+--+ +--+

Lack of transition means error state

How does the computation end?

Most errors are handled by lack of transitions.

Thus, if "c" is found while looking for a "b" (in state k1) it means that

there are too few "b"s and the machine rejects the nput by having no

applicable transition.

To make the machine recursive, add transitions for each error case

that lead to a reject state.

How do we accept the strings?

delta(k0,B) = (k\_check,B,R) Last "a" has been marked so a "b" is found in k0

where k\_check is a state responsible for verifying that all "b"s and "c"s

have been marked.

If not - go to k\_reject

If so - go to k\_accept

k\_check transitions left as an exercise.

\* Relationships between classes of languages

Regular < CFL < CSL < Recursive < Recursively Enumerable

Note that {a^n b^n c^n: n>=1} is in Recursive - CFL.