CHAPTER 2: CONTEXT-FREE LANGUAGES

CONTEXT-FREE GRAMMARS ("CFG's")

Definition & Examples

CHOMSKY NORMAL FORM

PUSHDOWN AUTOMATA

Non-deterministic $\neq$ Deterministic

EQUIVALENCE BETWEEN

CONTEXT-FREE GRAMMARS

AND

NON-DETERMINISTIC PUSHDOWN AUTOMATA

PROOF

PUMPING LEMMA

TO PROVE SOME LANGUAGES ARE NOT CONTEXT-FREE

EXAMPLE

$$E \to E + T \mid T$$
$$T \to T \times F \mid F$$
$$F \to (E) \mid a$$

$$E \to E + T$$
$$E \to T$$

VARIABLES (= "NON TERMINALS")

TERMINALS (SYMBOLS FROM ALPHABET)

RULES (= "PRODUCTIONS")

$$E ::= E + T$$

START VARIABLE. Often "S"

Assume: First rule uses start symbol

$$E \rightarrow E + T \mid T$$
$$T \rightarrow T \times F \mid F$$
$$F \rightarrow (E) \mid a$$

DERIVATION

$$\underline{E} \Rightarrow \underline{E}+T \Rightarrow \underline{T}+T \Rightarrow \underline{F}+T \Rightarrow a+\underline{T}$$
$$\Rightarrow a+\underline{F} \Rightarrow a+(\underline{E}) \Rightarrow a+(\underline{T})$$
$$\Rightarrow a+(\underline{T}\times F) \Rightarrow a+(\underline{F}\times F)$$
$$\Rightarrow a+(a \times \underline{F}) = a+(a\times a)$$

WE WRITE:

$$E \overset{*}{\Longrightarrow} a+(a\times a)$$
$$E \overset{*}{\Longrightarrow} a+(E) \Longrightarrow a+(T) \overset{*}{\Longrightarrow} a+(a\times a)$$

LEFT-MOST DERIVATION:

Always choose Left-most Variable

$$\ldots \Longrightarrow \underline{F}+T \Longrightarrow \underline{a}+T \Longrightarrow \ldots a+(a\times a)$$

RIGHT-MOST DERIVATION:

Always choose Right-most Variable

$$\Longrightarrow F+\underline{T} \Longrightarrow F+\underline{F} \Longrightarrow \ldots a+(a\times a)$$

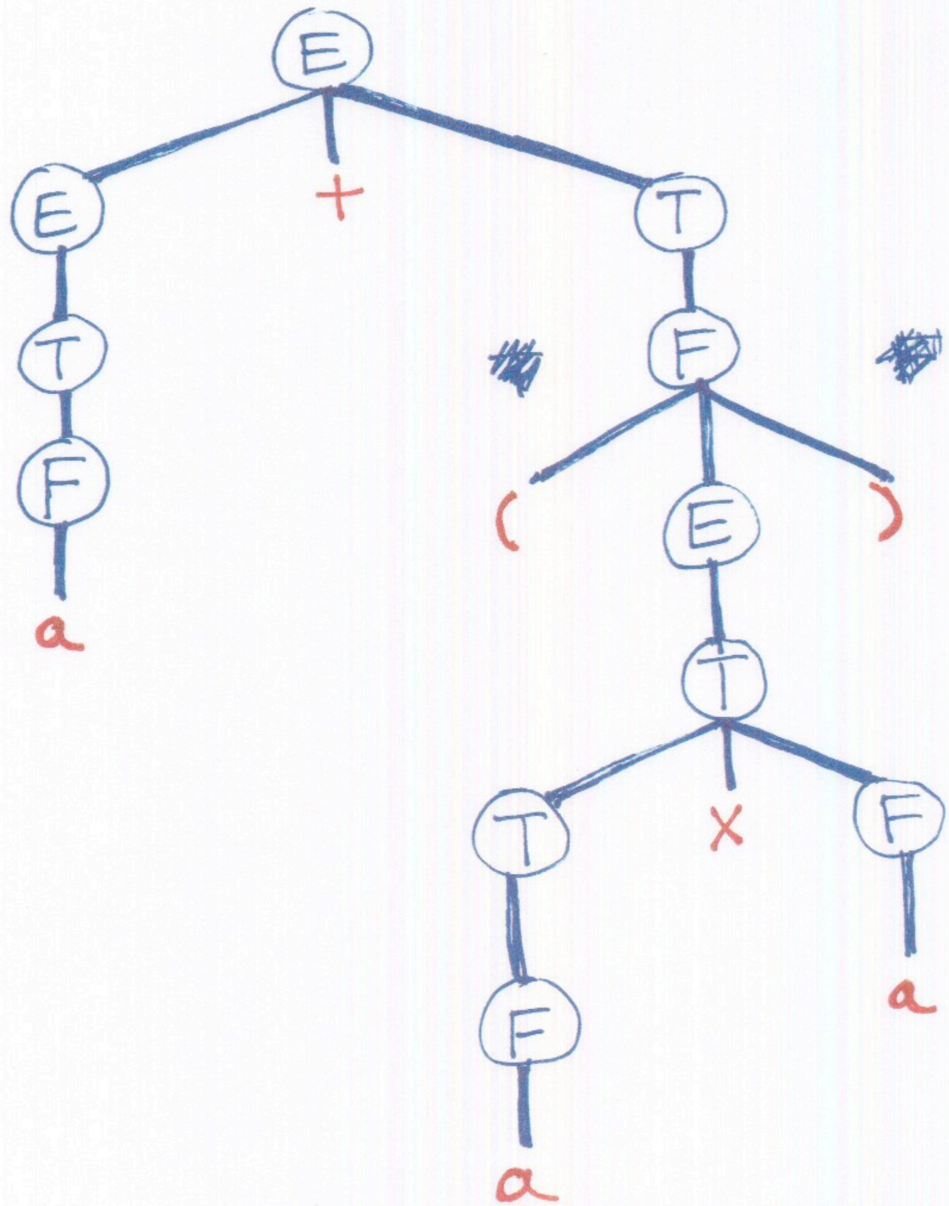
PARSE TREES

$E \to E + T \mid T$

$T \to T \times F \mid F$

$F \to (E) \mid a$

$E \stackrel{*}{\Rightarrow} a + (a \times a)$



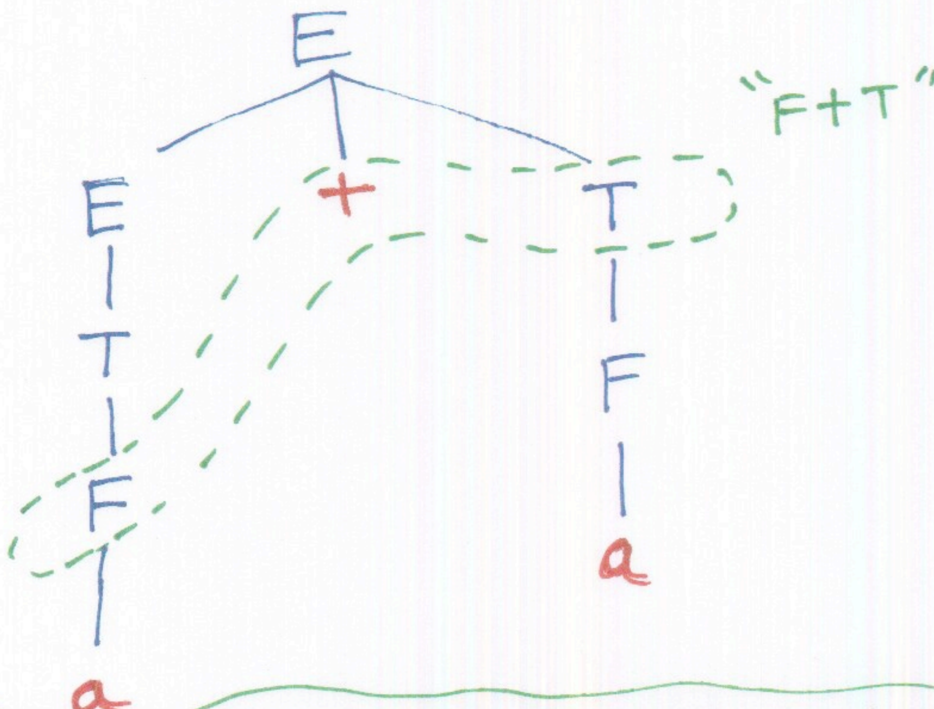
YOU GET THE SAME RESULT:

$$
\boxed{\begin{array}{l} E \to E+T \mid T \\ T \to T \times F \mid F \\ F \to (E) \mid a \end{array}}
$$

$$\ldots \Rightarrow \underline{F}+T \Rightarrow a+\underline{T} \Rightarrow a+\underline{F} \Rightarrow a+a$$

$$\ldots \Rightarrow F+\underline{T} \Rightarrow F+\underline{F} \Rightarrow \underline{F}+a \Rightarrow a+a$$

PARSE TREE:



The parse tree abstracts away the actual order in which the rules are used. It only remembers <u>which</u> rules were used.

DEFINITION

A "CONTEXT-FREE GRAMMAR" ("CFG")

$G = (V, \Sigma, R, S)$

V = The set of VARIABLES (i.e., NON-TERMINALS)

Σ = The set of TERMINALS

NOTE: $V \cap \Sigma = \{\}$.

Both are finite sets.

R = The set of RULES (i.e., PRODUCTIONS)

S = Start Variable. $S \in V$.

DEFINITION

The Language of a grammar is

$$\{ w \mid w \in \Sigma^* \text{ and } S \overset{*}{\Rightarrow} w \}$$

DEFINITION

A "CONTEXT-FREE LANGUAGE" IS A LANGUAGE GENERATED BY A CONTEXT-FREE GRAMMAR.

6

EXAMPLE CFG

$$S \rightarrow (S) \mid SS \mid \epsilon$$

Language = { ε, (), ()(), (()()), ((()())()(()())), ((())), (()()()()()), ... }

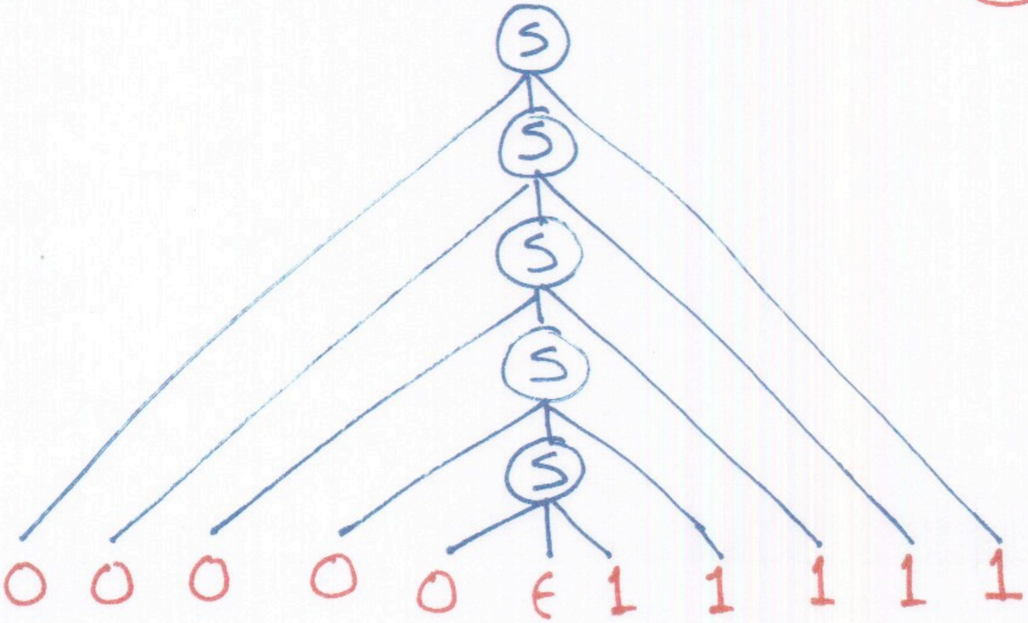
EXAMPLE

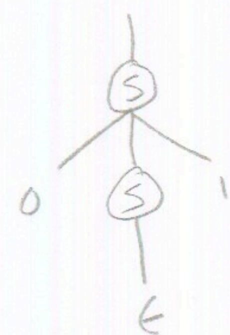
$$\{ 0^n 1^n \mid n \geq 0 \}$$

We used the PUMPING LEMMA to show this is NOT REGULAR.
So:
REG $\subset$ CFL
[PROPER SUBSET]

$$S \rightarrow \epsilon$$

$$S \rightarrow 0S1$$

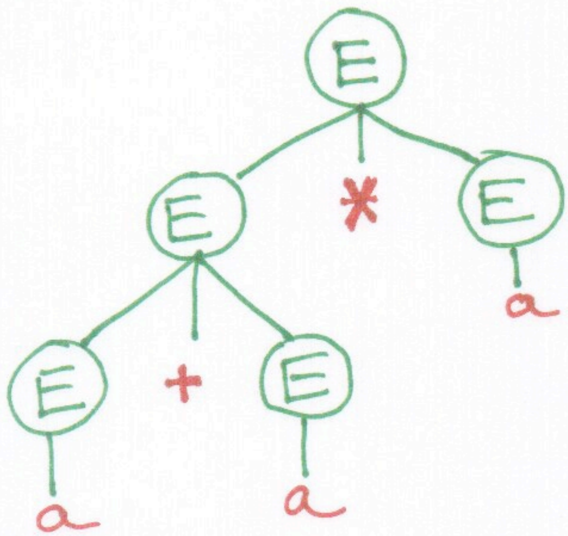


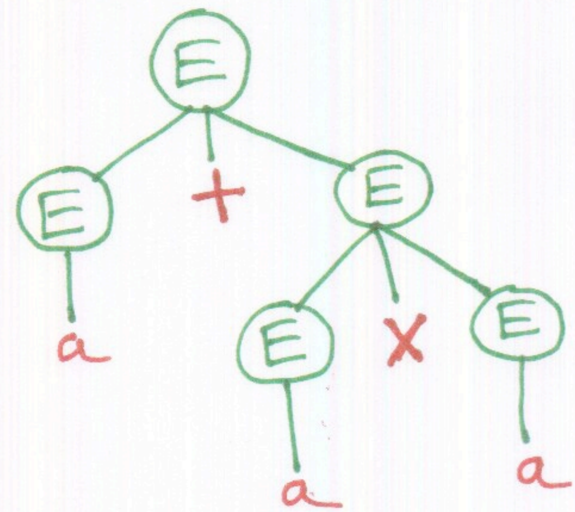


7

$$
\begin{aligned}
E &\to E + E \\
&\to E \times E \\
&\to (E) \\
&\to a
\end{aligned}
$$

$a + a \times a$





- FOR EACH PARSE TREE, THERE IS EXACTLY ONE LEFT-MOST DERIVATION. (AND ONE RIGHT-MOST DERIVATION.)

AMBIGUOUS STRINGS

- MORE THAN ONE PARSE TREE.
- FUNDAMENTALLY DIFFERENT WAYS TO DERIVE THIS STRING.

AMBIGUOUS GRAMMAR

A grammar is "AMBIGUOUS" if some string can be derived in more than one way.
I.e., Some string has MULTIBLE PARSE TREES.

AMBIGUOUS GRAMMAR

$$
\begin{aligned}
E &\rightarrow E + E \\
&\rightarrow E \times E \\
&\rightarrow (E) \\
&\rightarrow a
\end{aligned}
$$

EQUIVALENT UNAMBIGUOUS GRAMMAR

$$
\begin{aligned}
E &\rightarrow E + T \\
&\rightarrow T \\
T &\rightarrow T \times F \\
&\rightarrow F \\
F &\rightarrow (E) \\
&\rightarrow a
\end{aligned}
$$

8.1

AMBIGUOUS STRINGS

MULTIPLE LEFT-MOST DERIVATIONS
≡ MULTIPLE PARSE TREES.

AMBIGUOUS GRAMMAR

IF THERE EXISTS SOME STRING
THAT CAN BE DERIVED AMBIGUOUSLY.

If you have an ambiguous grammar, you might be able to find an equivalent grammar that is unambiguous.

AMBIGUOUS LANGUAGE

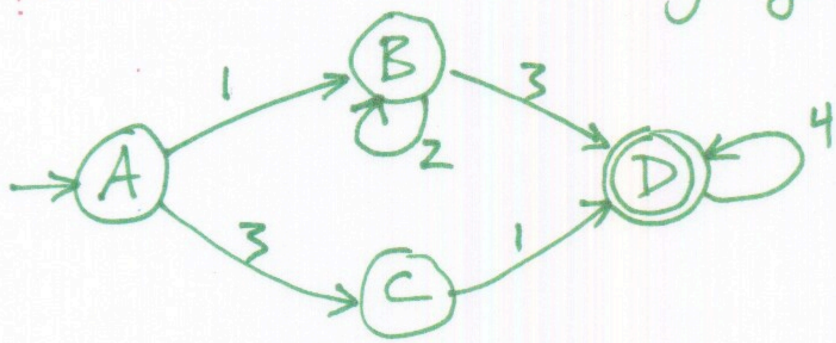
ALL GRAMMARS FOR THE LANGUAGE
ARE AMBIGUOUS.
THERE IS NO UNAMBIGUOUS GRAMMAR
THAT ACCEPTS THE LANGUAGE
THE LANGUAGE IS "INHERENTLY AMBIGUOUS."

8.12

EVERY REGULAR LANGUAGE IS CONTEXT-FREE.

PROOF:

Given a DFA for the Language, Construct a grammar that generates the same Language.



- Make a variable for each state.
- Make the variable for the starting state the starting variable
- Make a rule for each edge
- Add an EPSILON Rule for each accept state.

$A \rightarrow 1B$

$A \rightarrow 3C$

$B \rightarrow 2B$

$B \rightarrow 3D$

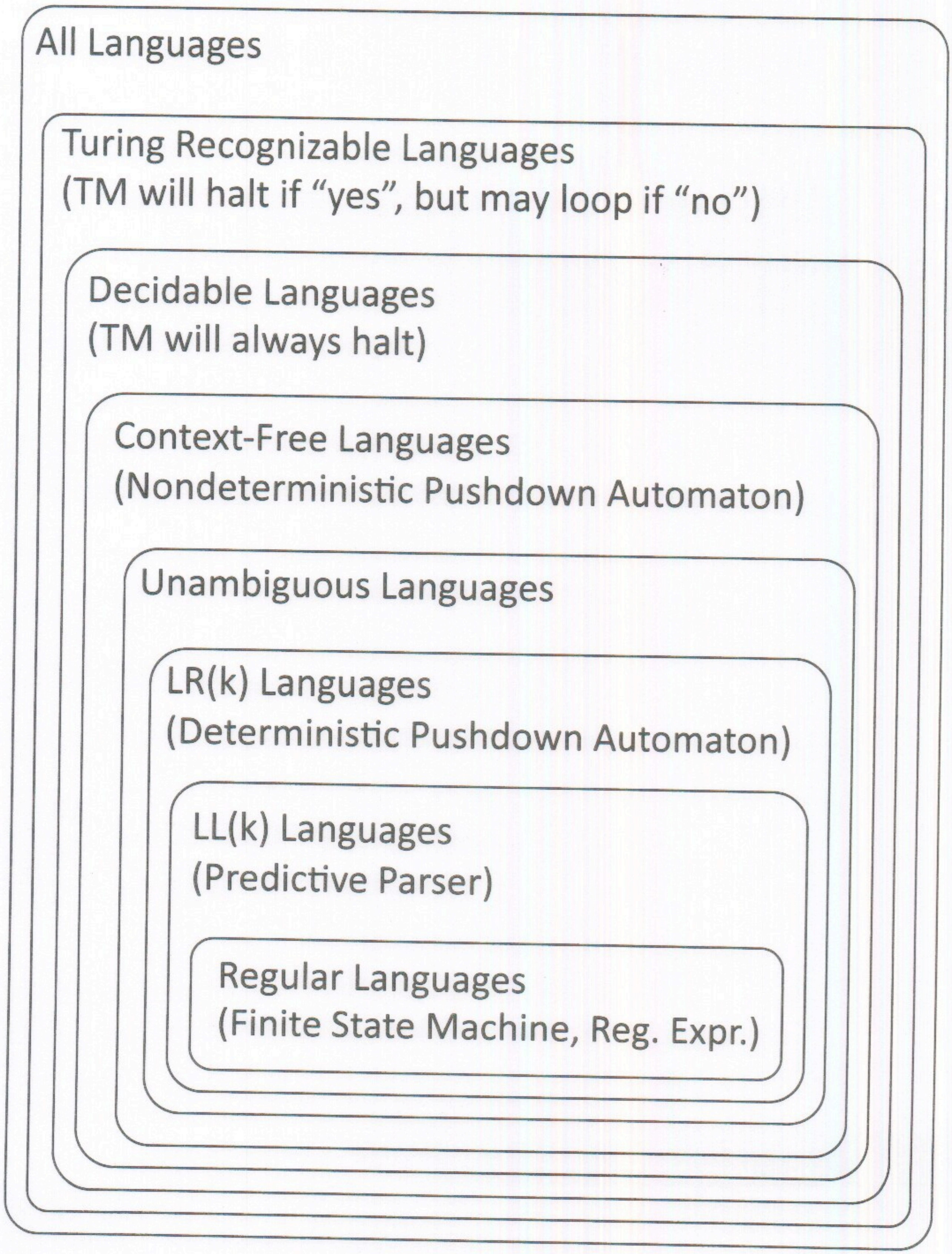
$C \rightarrow 1D$

$D \rightarrow 4D$

$D \rightarrow \epsilon$

9

THE LANGUAGE ONION:



10

CHOMSKY NORMAL FORM

EVERY RULE IN THE GRAMMAR HAS THE FORM

$$A \to BC$$

EXACTLY TWO VARIABLES; CAN'T BE "S"

OR

$$A \to a$$

OR A TERMINAL SYMBOL.

WE CAN ALSO HAVE

$$S \to \epsilon$$

THEOREM

EVERY CONTEXT-FREE LANGUAGE CAN BE GENERATED BY A GRAMMAR IN CHOMSKY NORMAL FORM.

Two grammars are "EQUIVALENT" if they generate the same language.

FOR EVERY CFG THERE IS AN EQUIVALENT CHOMSKY NORMAL FORM.

11

ALGORITHM TO CONVERT ANY CFG INTO CHOMSKY NORMAL FORM

STEP 1

MAKE SURE START SYMBOL DOES NOT APPEAR ON RIGHTHAND SIDE.

STEP 2

REMOVE RULES LIKE $A \rightarrow \epsilon$

STEP 3

GET RID OF ALL UNIT RULES $A \rightarrow B$

STEP 4

GET RID OF RULES WITH MORE THAN 2 SYMBOLS ON RIGHTHAND SIDE

$A \rightarrow BCDE$

$A \rightarrow Bcde$

STEP 5

MAKE SURE

$A \rightarrow BC$ ← ONLY 2. MUST BE VARIABLES.

$A \rightarrow a$ ← ONLY 1. MUST BE A TERMINAL SYMBOL.

11.1

PROOF

GIVEN A CFG G, SHOW HOW TO CONVERT IT TO CHOMSKY NORMAL FORM.

STEP 1: MAKE SURE START SYMBOL DOESN'T APPEAR ON RIGHTHAND SIDE. ADD NEW START SYMBOL.

EXAMPLE

$S \rightarrow ASA \mid aB$

$A \rightarrow B \mid S$

$B \rightarrow b \mid \epsilon$

$S_0 \rightarrow S$

$S \rightarrow ASA \mid aB$

$A \rightarrow B \mid S$

$B \rightarrow b \mid \epsilon$

Add new start symbol

12

STEP 2:

Can't have $A \rightarrow \epsilon$. (unless A is start Variable)

Remove such rules.

$$B \rightarrow BC\boxed{A}CB\boxed{A}B$$

$$\Downarrow$$

$$\left.\begin{array}{l} B \rightarrow BCACBAB \\ B \rightarrow BC\,|\,CB\boxed{A}B \\ B \rightarrow BC\boxed{A}CB\,|\,B \\ B \rightarrow BC\,|\,CB\,|\,B \end{array}\right\}$$ Add these new rules.

$S_0 \rightarrow S$

$S \rightarrow ASA \mid aB$

$A \rightarrow B \mid S$

$B \rightarrow b \mid \epsilon$

Remove $B \rightarrow \epsilon$

$S_0 \rightarrow S$

$S \rightarrow ASA \mid aB \mid a$

$A \rightarrow B \mid S \mid \epsilon$

$B \rightarrow b$

Remove $A \rightarrow \epsilon$

$S_0 \rightarrow S$

$S \rightarrow ASA \mid aB \mid a \mid SA \mid AS \mid S$

$A \rightarrow B \mid S$

$B \rightarrow b$

13

STEP 3:

GET RID OF ALL "UNIT RULES"

$A \rightarrow B$

GIVEN:	ADD:
$B \rightarrow xyz$	$A \rightarrow xyz$

FROM BEFORE:

$S_0 \rightarrow S$

$S \rightarrow ASA \mid aB \mid a \mid SA \mid AS \mid \underline{\underline{S}}$

$A \rightarrow B \mid S$

$B \rightarrow b$

Remove $S \rightarrow S$ (do nothing)

$S_0 \rightarrow S$

$S \rightarrow ASA \mid aB \mid a \mid SA \mid AS$

$A \rightarrow B \mid S$

$B \rightarrow b$

Remove $S_0 \rightarrow S$

$S_0 \rightarrow ASA \mid aB \mid a \mid SA \mid AS$

$S \rightarrow ASA \mid aB \mid a \mid SA \mid AS$

$A \rightarrow B \mid S$

$B \rightarrow b$

14

FROM BEFORE:

$S_0 \rightarrow ASA \mid aB \mid a \mid SA \mid AS$

$S \rightarrow ASA \mid aB \mid a \mid SA \mid AS$

$A \rightarrow \underline{B} \mid S$

$B \rightarrow b$

REMOVE $A \rightarrow B$:

$S_0 \rightarrow ASA \mid aB \mid a \mid SA \mid AS$

$S \rightarrow ASA \mid aB \mid a \mid SA \mid AS$

$A \rightarrow b \mid \underline{S}$

$B \rightarrow b$

REMOVE $A \rightarrow S$:

$S_0 \rightarrow ASA \mid aB \mid a \mid SA \mid AS$

$S \rightarrow ASA \mid aB \mid a \mid SA \mid AS$

$A \rightarrow b \mid ASA \mid aB \mid a \mid SA \mid AS$

$B \rightarrow b$

15

STEP 4:

Replace

$$A \to BCDE$$

with

$$A \to BA_1$$
$$A_1 \to CA_2$$
$$A_2 \to DE$$

Introduce new variables to make sure every righthand side is not longer than 2.

FROM BEFORE:

$$S_0 \to ASA \mid aB \mid a \mid SA \mid AS$$
$$S \to ASA \mid aB \mid a \mid SA \mid AS$$
$$A \to b \mid ASA \mid aB \mid a \mid SA \mid AS$$
$$B \to b$$

SHORTEN $S_0 \to ASA$ and $S \to ASA$ and $A \to ASA$ By introducing A_1:

$$S_0 \to AA_1 \mid aB \mid a \mid SA \mid AS$$
$$S \to AA_1 \mid aB \mid a \mid SA \mid AS$$
$$A \to b \mid AA_1 \mid aB \mid a \mid SA \mid AS$$
$$A_1 \to SA$$
$$B \to b$$

16

Step 5:

Replace $A \rightarrow bC$

with: $A \rightarrow A_1 C$

$A_1 \rightarrow b$

FROM BEFORE:

$$S_0 \rightarrow AA_1 \mid aB \mid a \mid SA \mid AS$$
$$S \rightarrow AA_1 \mid aB \mid a \mid SA \mid AS$$
$$A \rightarrow b \mid AA_1 \mid aB \mid a \mid SA \mid AS$$
$$A_1 \rightarrow SA$$
$$B \rightarrow b$$

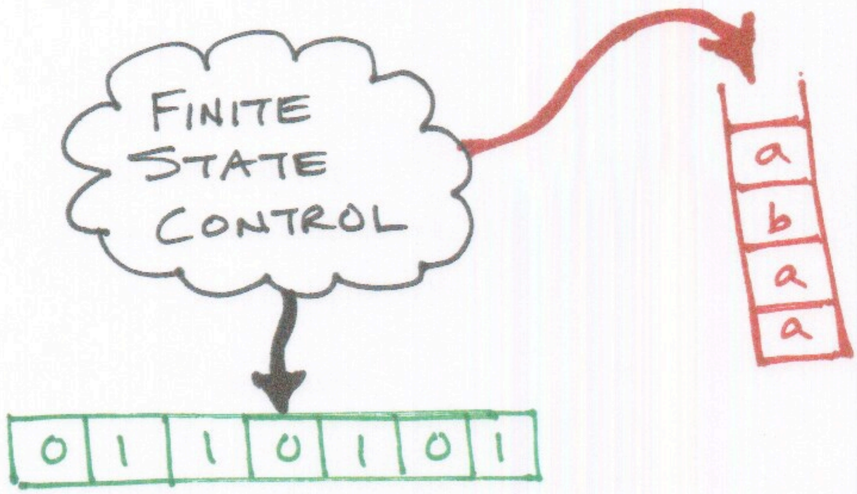
INTRODUCE $A_2 \rightarrow a$

$$S_0 \rightarrow AA_1 \mid A_2B \mid a \mid SA \mid AS$$
$$S \rightarrow AA_1 \mid A_2B \mid a \mid SA \mid AS$$
$$A \rightarrow b \mid AA_1 \mid A_2B \mid a \mid SA \mid AS$$
$$A_1 \rightarrow SA$$
$$B \rightarrow b$$
$$A_2 \rightarrow a$$

THIS GRAMMAR IS NOW IN CHOMSKY NORMAL FORM!

17

PUSHDOWN AUTOMATA



INPUT STRING:

SAME AS F.S.M.

(CANNOT BACK UP)

STACK

OPERATIONS:

READ+POP / IGNORE

PUSH / IGNORE

STACK ALPHABET

Γ GAMMA,

MAY BE DIFFERENT FROM INPUT ALPHABET, Σ

STATE TRANSITIONS:

MAY DEPEND ON STACK TOP.

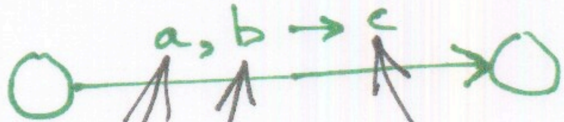
MAY PUSH ONTO THE STACK.

NON-DETERMINISTIC!

18

FINITE STATE MACHINE

PUSHDOWN AUTOMATON



Input symbol

Symbol on top of the stack. This symbol is popped.

This symbol is pushed onto the stack.

May be "ε"

"ε" means the stack is neither read nor popped

"ε" means nothing is pushed.

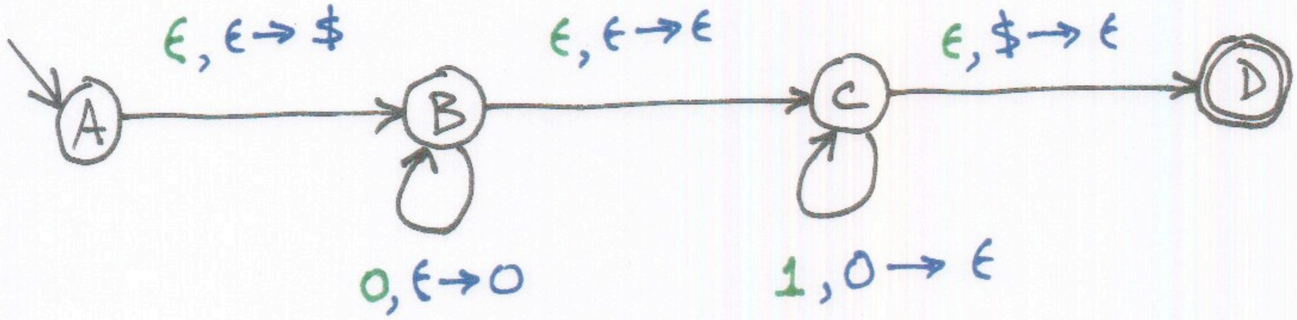
NONDETERMINISM

$$\{0^n 1^n \mid n \geq 0\}$$

$\Sigma = \{0, 1\}$ input alphabet

$\Gamma = \{\$, 0\}$ stack alphabet

To detect bottom of stack.
(We could use other symbols, but these seem clear enough.)



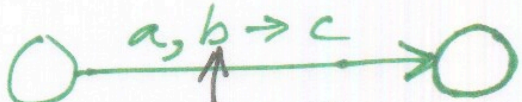
WHEN IS A STRING ACCEPTED?

- BEGIN IN THE START STATE.
- END IN AN ACCEPT STATE.
- CONSUME ALL THE INPUT SYMBOLS.

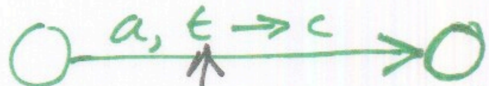
(OKAY TO LEAVE STUFF ON THE STACK.)

- THERE IS A PATH THRU THE FINITE STATE CONTROL.

NOTE: IT IS NOT POSSIBLE TO POP AN EMPTY STACK.



Read a "b" and pop it.



Don't look at the stack

- NON-DETERMINISTIC:

You just have to find one path to a ACCEPT state.

21

FORMAL DEFINITION

$$(Q, \Sigma, \Gamma, \delta, q_0, F)$$

Q = Set of states

Σ = Input alphabet

Γ = Stack alphabet

$$\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$$
$$\Gamma_\epsilon = \Gamma \cup \{\epsilon\}$$

$$\delta: Q \times \Sigma_\epsilon \times \Gamma_\epsilon \rightarrow \mathcal{P}(Q \times \Gamma_\epsilon)$$

q_0: Starting State $\quad q_0 \in Q$

F: Accept States $\quad F \subseteq Q$

22

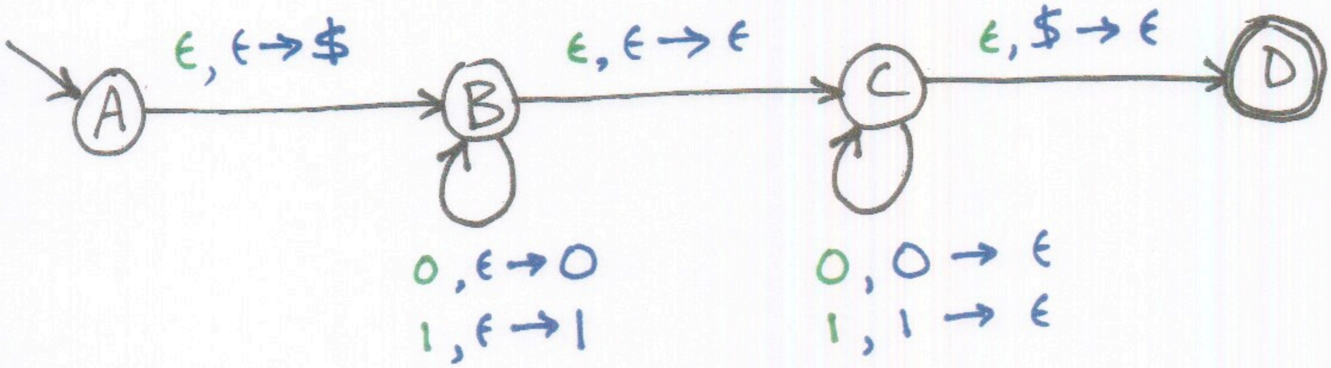
EXAMPLE

Palindrome

MADAM IM ADAM
WAS IT A CAT I SAW
NO LEMON, NO MELON

$$\{ w \mid w \text{ is a palindrome AND } w \in \{0,1\}^* \}$$

$$\boxed{\begin{aligned} S &\to 0S0 \\ &\to 1S1 \\ &\to \epsilon \end{aligned}}$$



GRAMMAR DESIGN CHALLENGE

$L = \{ w \mid w \in \{0,1\}^*$ and the number of 0's equals the number of 1's $\}$

APPROACH: TRY TO THINK OF "MEANINGS" FOR THE NON-TERMINALS.

S = EQUAL # of 0's and 1's.

A = ONE MORE "1" THAN "0"'s

B = ONE MORE "0" THAN "1"'s

SOLUTION:

$S \rightarrow 0A \mid 1B \mid \epsilon$

$A \rightarrow 1S \mid 0AA$

$B \rightarrow 0S \mid 1BB$

24

SOLUTION #2:

$$S \rightarrow SAB \mid \epsilon$$
$$A \rightarrow 0S1 \mid \epsilon$$
$$B \rightarrow 1S0 \mid \epsilon$$

NOTE:

$$C \rightarrow Cx \mid \epsilon$$

GENERATES: x^*

$\{\epsilon, x, xx, xxx, xxxx, \ldots\}$

$$S \rightarrow SAB \mid \epsilon$$

GENERATES:

$$AB\,AB\,AB\,AB \ldots$$

EVERY A CAN GO TO ϵ.
EVERY B CAN GO TO ϵ.
$\Rightarrow$ ANY STRING OF A's AND B's!

$$\underbrace{01}_{A}\ \underbrace{01}_{A}\ \underbrace{01}_{A}\ \underbrace{01}_{A} \qquad \underbrace{0\ \underbrace{\boxed{0\,0\,0\,1\,1\,1}}_{S}\ 1}_{A}$$

ARE TWO GRAMMARS EQUIVALENT?

"EQUIVALENT" = GENERATE THE SAME LANGUAGE

UNDECIDABLE!

CANNOT WRITE A COMPUTER PROGRAM.

[PROGRAM MAY NOT HALT!]

APPROACH:

- GENERATE EVERY STRING IN TURN. (INFINITELY MANY).
- TEST EACH STRING.
  - ACCEPTED BY GRAMMAR #1?
    - FIND A PARSE TREE.
    - (THIS IS DECIDABLE.)
  - ACCEPTED BY GRAMMAR #2?
- FIND A COUNTER-EXAMPLE?
  - HALT; PRINT "NOT EQUIVALENT"
- OTHERWISE, KEEP LOOKING.
  - MAY NOT HALT... OR MAY...?
- NO WAY TO KNOW WHEN TO STOP LOOKING!!!

26

NON-CONTEXT-FREE GRAMMARS

$L = \{ 0^N 1^N 0^N \mid N \geq 0 \}$

$\underbrace{00000}_{5}\underbrace{11111}_{5}\underbrace{00000}_{5}$

IS THIS LANGUAGE CONTEXT-FREE?

NO!

(Use the PUMPING LEMMA FOR CFG'S TO PROVE IT.)

CHOMSKY HIERARCHY

TYPE-3 LANGUAGES

REGULAR

TYPE-2 LANGUAGES

CONTEXT-FREE $\quad A \rightarrow \gamma$

TERMINALS AND NON-TERMINALS

TYPE-1 LANGUAGES

CONTEXT-SENSITIVE

$\alpha A \beta \rightarrow \alpha \gamma \beta$

TYPE-0 LANGUAGES

RECURSIVELY-ENUMERABLE
TURING-ENUMERABLE.
TURING-RECOGNIZABLE

ex: $ABx \rightarrow ADEy$

27

$$L = \{ 1^N 2^N 3^N \mid N \geq 1 \}$$

APPROACH:

EACH "A" will turn into a "1".
EACH "B" will turn into a "2".
EACH "C" will turn into a "3".

$$S \stackrel{*}{\Rightarrow} \ldots \quad AAABBBCCC \stackrel{*}{\Rightarrow} \ldots 111\ 222\ 333$$

INITIALLY THE STRING WILL START WITH "1"'S

111 BBB CCC

Will never use "A" actually.

RULES TO FINISH IT UP:

$1B \rightarrow 12$
$2B \rightarrow 22$
$2C \rightarrow 23$
$3C \rightarrow 33$

1 1 1 B B B C C C
1 1 1 2 B B C C C
1 1 1 2 2 B C C C
1 1 1 2 2 2 C C C
1 1 1 2 2 2 3 C C
1 1 1 2 2 2 3 3 C
1 1 1 2 2 2 3 3 3

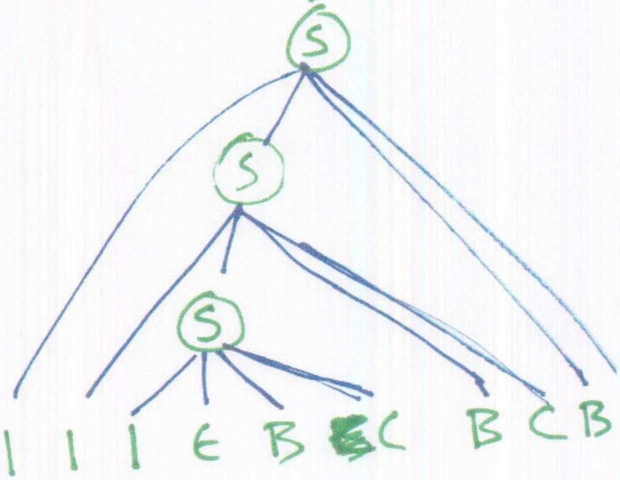
NOTE
... CB ....
CAN NEVER BE REDUCED.
"C" CAN ONLY TURN INTO A "3".
AND "3B" CAN NEVER BE REDUCED.
$\Rightarrow$ B'S MUST PRECEED "C"S.

28

STEP 1: GENERATE THE CORRECT NUMBER OF 1'S, B'S, and C'S (JUST NOT THE RIGHT ORDER.).

$S \rightarrow 1SBC$

$S \rightarrow \epsilon$



1111 BC BC BC BC

STEP 2: GET THE "B"S IN FRONT OF THE "C"S.

$CB \rightarrow BC$

STEP 3:

REDUCE "B" TO "2" AND "C" TO "3."

(Rules shown above.)

CONTEXT-SENSITIVE GRAMMAR

$L = \{ 1^N 2^N 3^N \mid N \geq 1 \}$

$S \rightarrow 1SBC$

$S \rightarrow \epsilon$

FROM BEFORE

$CB \rightarrow HB$

$HB \rightarrow HC$

$HC \rightarrow BC$

REVISED

$1B \rightarrow 12$

$2B \rightarrow 22$

$2C \rightarrow 23$

$3C \rightarrow 33$

FROM BEFORE

30

ARE CONTEXT-FREE LANGUAGES CLOSED UNDER UNION?

GRAMMAR 1:

$S_1 \rightarrow$ ... lots of rules...

GRAMMAR 2:

$S_2 \rightarrow$ ... lots of rules...

UNION:

$S \rightarrow S_1 \mid S_2$

$S_1 \rightarrow \ldots$

$S_2 \rightarrow \ldots$

YES!

ARE CONTEXT-FREE LANGUAGES CLOSED UNDER CONCATENATION?

$S \rightarrow S_1 \, S_2$

$S_1 \rightarrow \ldots$

$S_2 \rightarrow \ldots$

YES!

ARE CONTEXT-FREE LANGUAGES CLOSED UNDER INTERSECTION?

CONSIDER $L_1 = \{0^N 1^N 2^i\}$

IT IS A "CFL":

$S \rightarrow AB$
$A \rightarrow 0A1 \mid \epsilon$
$B \rightarrow 2B \mid \epsilon$

CONSIDER $L_2 = \{0^K 1^N 2^N\}$

IT IS A "CFL":

$S \rightarrow AB$
$A \rightarrow 0A \mid \epsilon$
$B \rightarrow 1B2 \mid \epsilon$

CONSIDER $L = L_1 \cap L_2$

$\{0^N 1^N 2^N\}$

THIS IS <u>NOT</u> A CONTEXT-FREE LANGUAGE.

NO!

NOT IN GENERAL, BUT FOR SOME LANGUAG[ES] THE INTERSECTION MAY ALSO BE CONTEXT-FREE.

32

ARE CONTEXT-FREE LANGUAGES CLOSED UNDER COMPLEMENT?

THESE ARE SETS.

DEMORGAN'S LAWS APPLY.

$$A \cap B = \overline{\bar{A} \cup \bar{B}}$$

ASSUME: CLOSED UNDER COMPLEMENT.

THEN RIGHT-HAND SIDE IS CLOSED.

THEN LEFT-HAND SIDE IS A C.F.L.

CONTRADICTION.

No! (NOT IN GENERAL, BUT YES FOR SOME LANGUAGES

EXAMPLE

$L = \{ww \mid w \in \{0,1\}^*\}$

PALINDROMES $= \{ww^R \mid w \ldots$

The first half of the string is equal to the second half.

L is not context-free.

HOWEVER

$\bar{L}$ IS CONTEXT-FREE.

33

THEOREM

A LANGUAGE IS CONTEXT-FREE IFF SOME PUSHDOWN AUTOMATON RECOGNIZES IT.

PROOF

PART 1:

GIVEN A CFG, SHOW HOW TO CONSTRUCT A PUSHDOWN AUTOMATON THAT RECOGNIZES IT.

PART 2:

GIVEN A PUSHDOWN AUTOMATON, SHOW HOW TO CONSTRUCT A CONTEXT-FREE GRAMMAR THAT RECOGNIZES THE SAME STRINGS.

34

GIVEN: A GRAMMAR

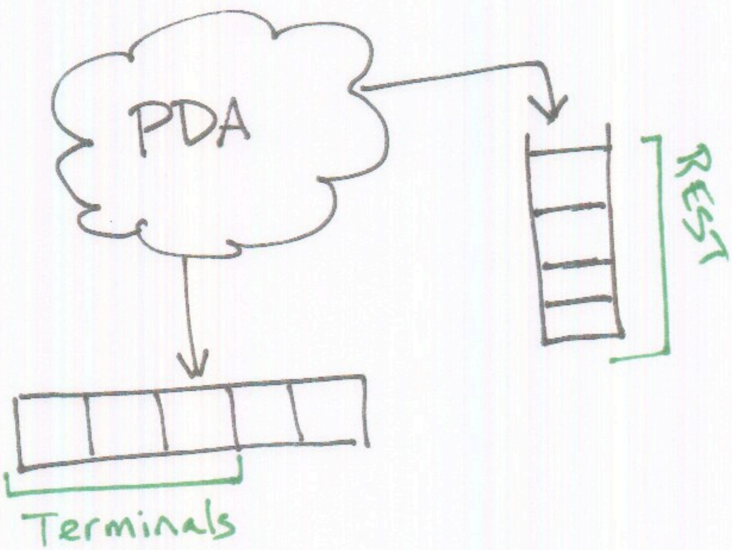
$$S \rightarrow BS \mid A$$
$$A \rightarrow 0A \mid \epsilon$$
$$B \rightarrow BB1 \mid 2$$

FIND: (OR BUILD) A P.D.A.

PROOF, PART 1

CONSIDER A DERIVATION: (LEFT-MOST)

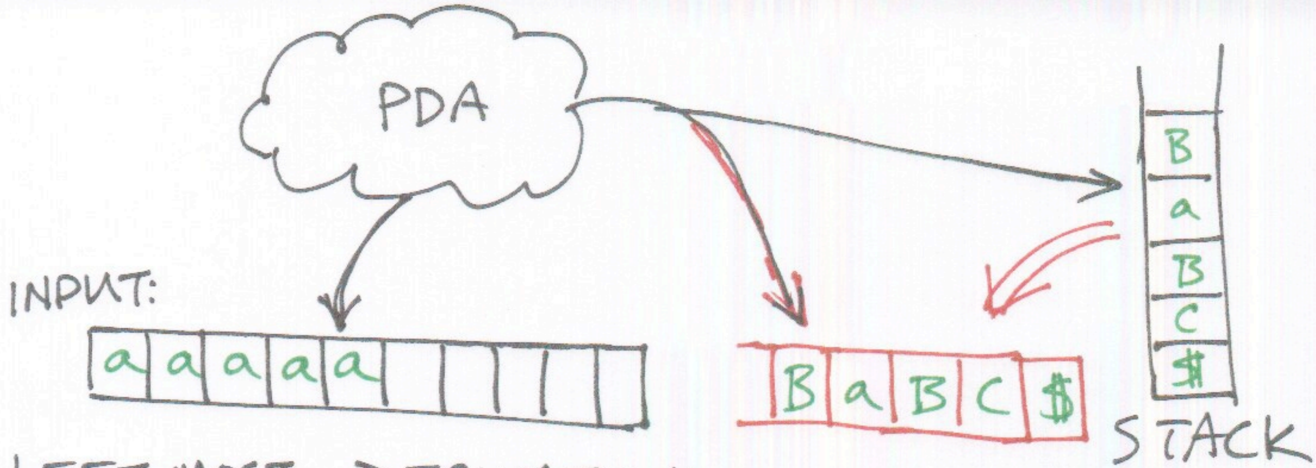
$$
\begin{aligned}
& S \\
\Rightarrow\ & BS \\
\Rightarrow\ & BB1S \\
\Rightarrow\ & 2B1S \\
\Rightarrow\ & 221S \\
\Rightarrow\ & 221A \\
\Rightarrow\ & 221\,\epsilon
\end{aligned}
$$



GENERAL FORM:

$$\underbrace{a\,a\,a\,a\,a}_{\text{Terminals}}\ \underbrace{B\,a\,B\,B\,a\,C\,a}_{\text{Rest (both)}}$$

35



LEFT-MOST DERIVATION:

$$S \stackrel{*}{\Rightarrow} \ldots \quad aaaaa\boxed{BaBC} \Rightarrow \ldots$$

AT EACH STEP, EXPAND LEFT-MOST NON-TERMINAL.

RULE:

$$B \rightarrow ASAxBA$$

$$\ldots \Rightarrow aaaaa\boxed{ASAxBAaBC}$$

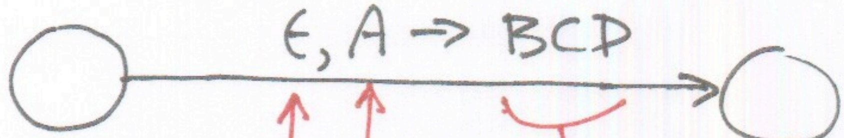
SO:

- MATCH STACK-TOP TO A RULE
- POP STACK
- PUSH RIGHT-HAND SIDE OF RULE ONTO STACK.

36

RULE: $A \rightarrow BCD$

Add this to PDA



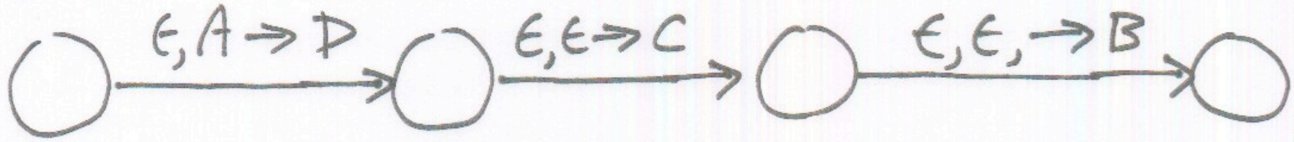
Right-hand side.

Match Top and Pop.

INPUT IS NOT ADVANCED.

NOTE:

TO PUSH MULTIPLE ITEMS, YOU'LL NEED TO ADD SOME EXTRA STATES.



WHICH RULE TO USE?

P.D.A.'S ARE <u>NON</u>-<u>DETERMINISTIC</u>!

(TRY THEM ALL IN PARALLEL.)

(JUST CHOOSE THE "RIGHT" RULE.)

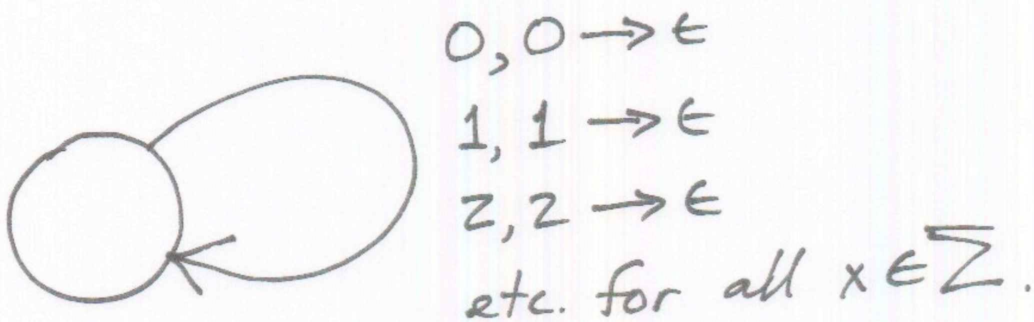
37

RULE:

$A \rightarrow 0102B3C$

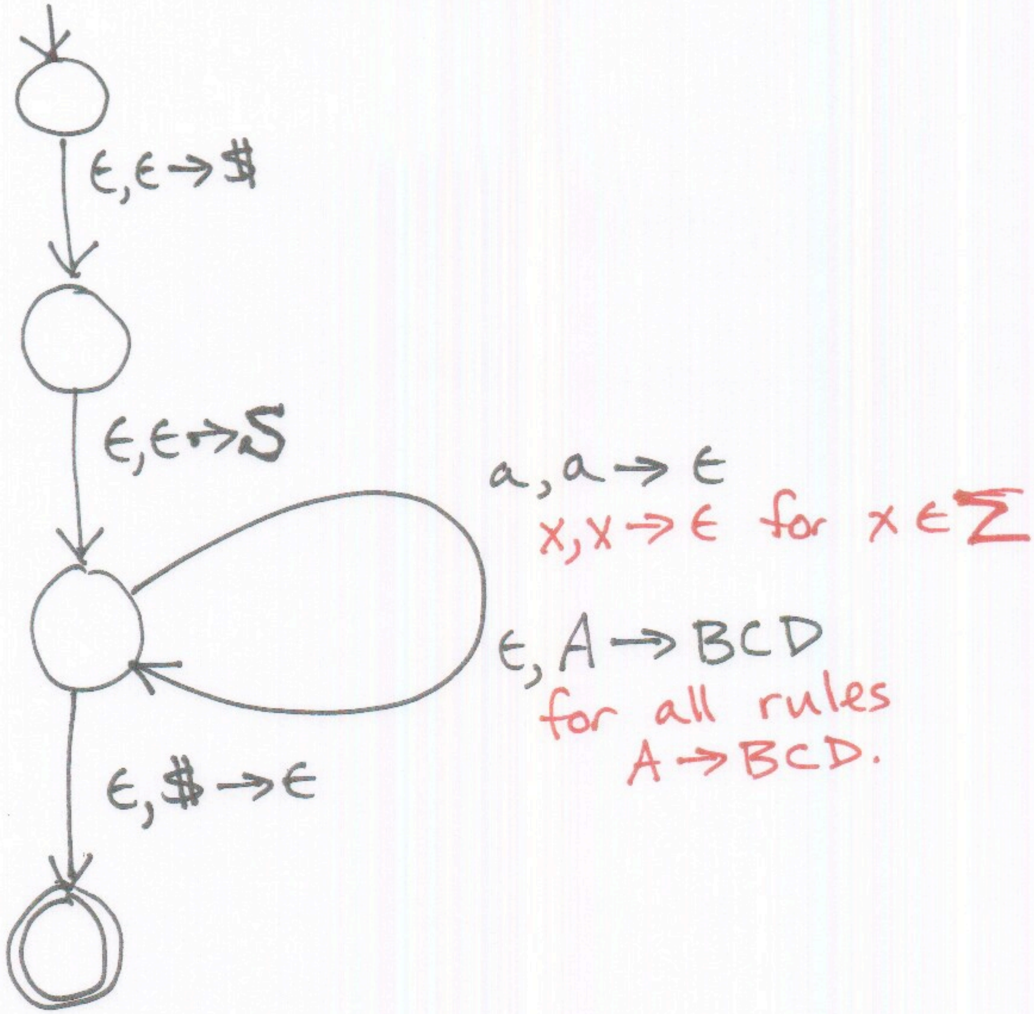
GIVES THIS:

SO:
MATCH TERMINAL SYMBOLS TO THE STACK TOP.



38

THE FINAL MACHINE:



PROOF, PART 2

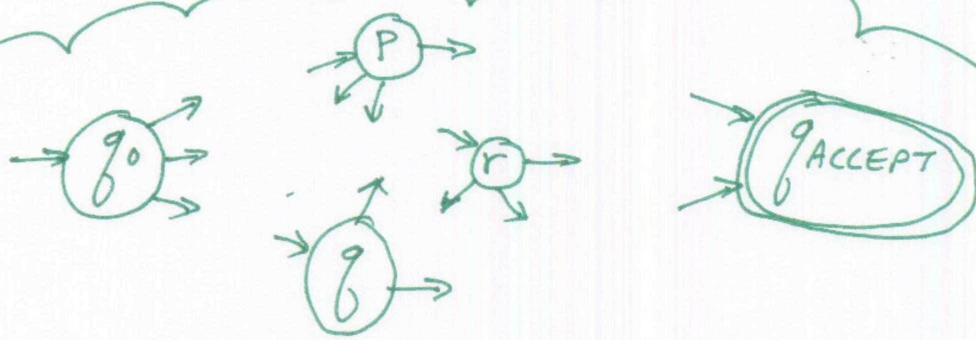
WE ARE GIVEN: A P.D.A.
MUST BUILD: A CFG FROM IT.

STEP 1

SIMPLIFY THE PDA.

STEP 2

BUILD THE CFG.



There will be a non-terminal for ever PAIR of states.

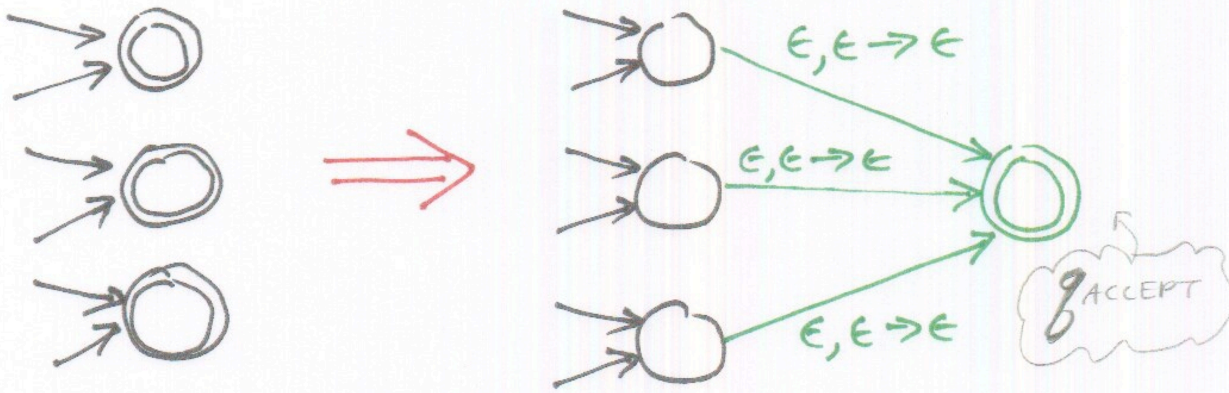
A_{pq} A_{qr} A_{rq_0} ...

The starting nonterminal will be

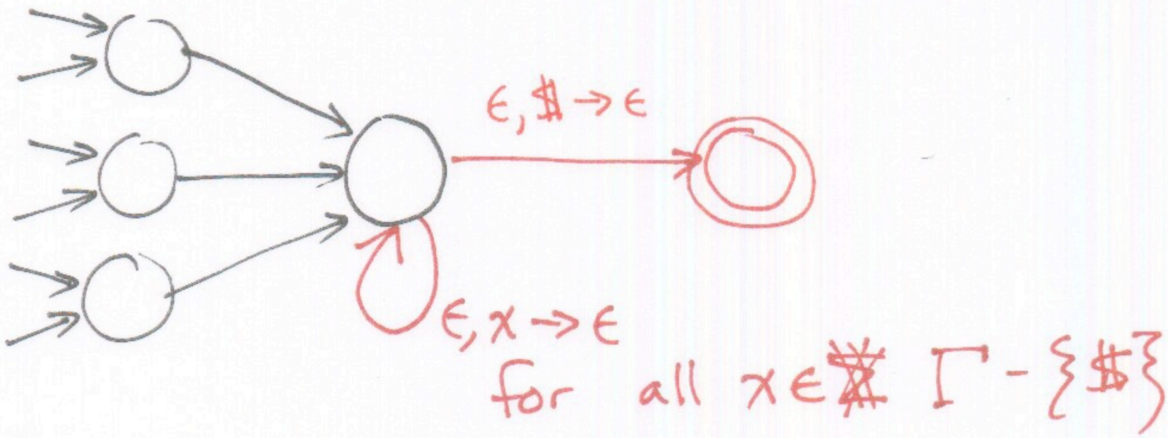
$A_{q_0 q_{ACCEPT}}$

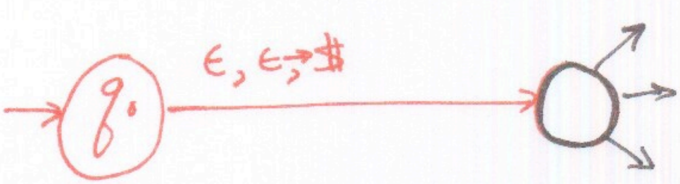
SIMPLIFY THE PDA

(1) The PDA has only one ACCEPT state.

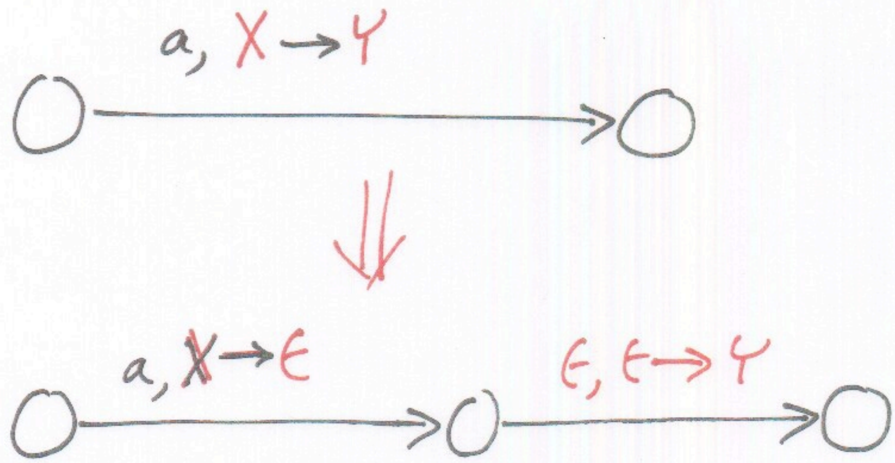


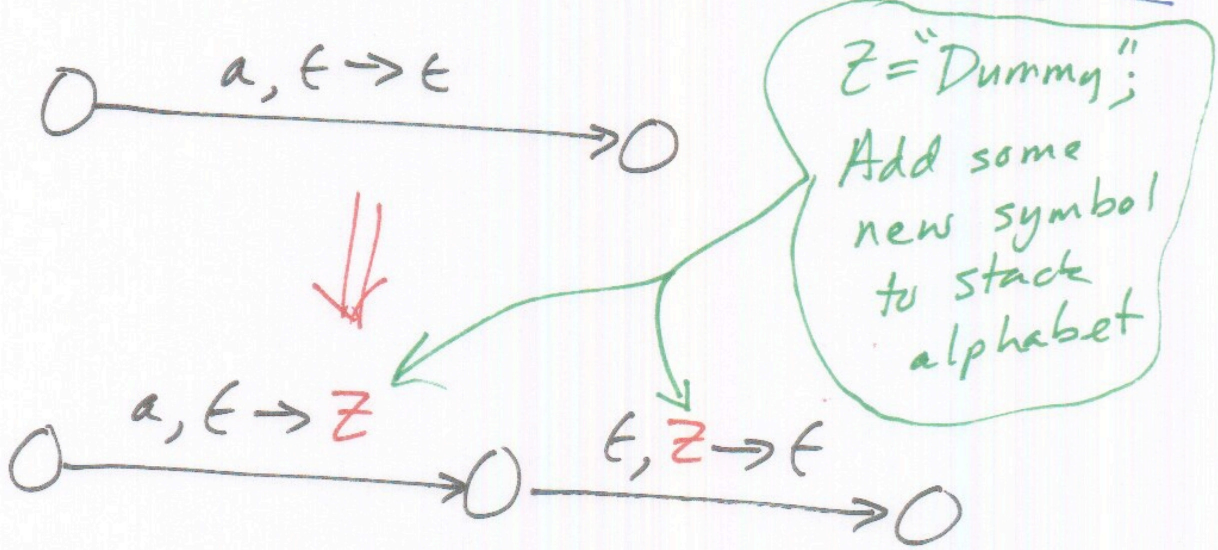
(2) The PDA empties its stack before ACCEPTING.





(3) Each transition either PUSHES or ~~POPS~~ POPS, but does not do both.

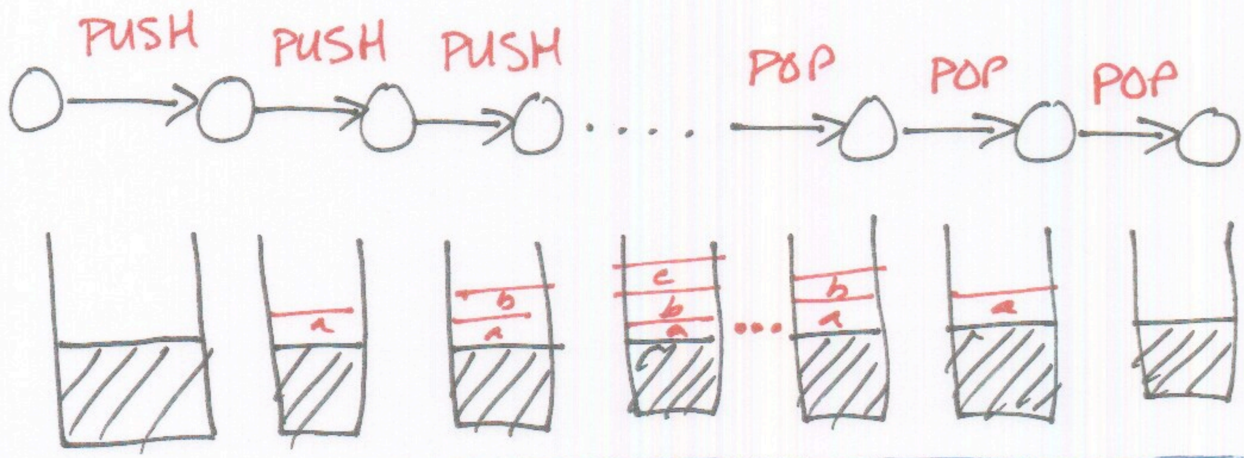


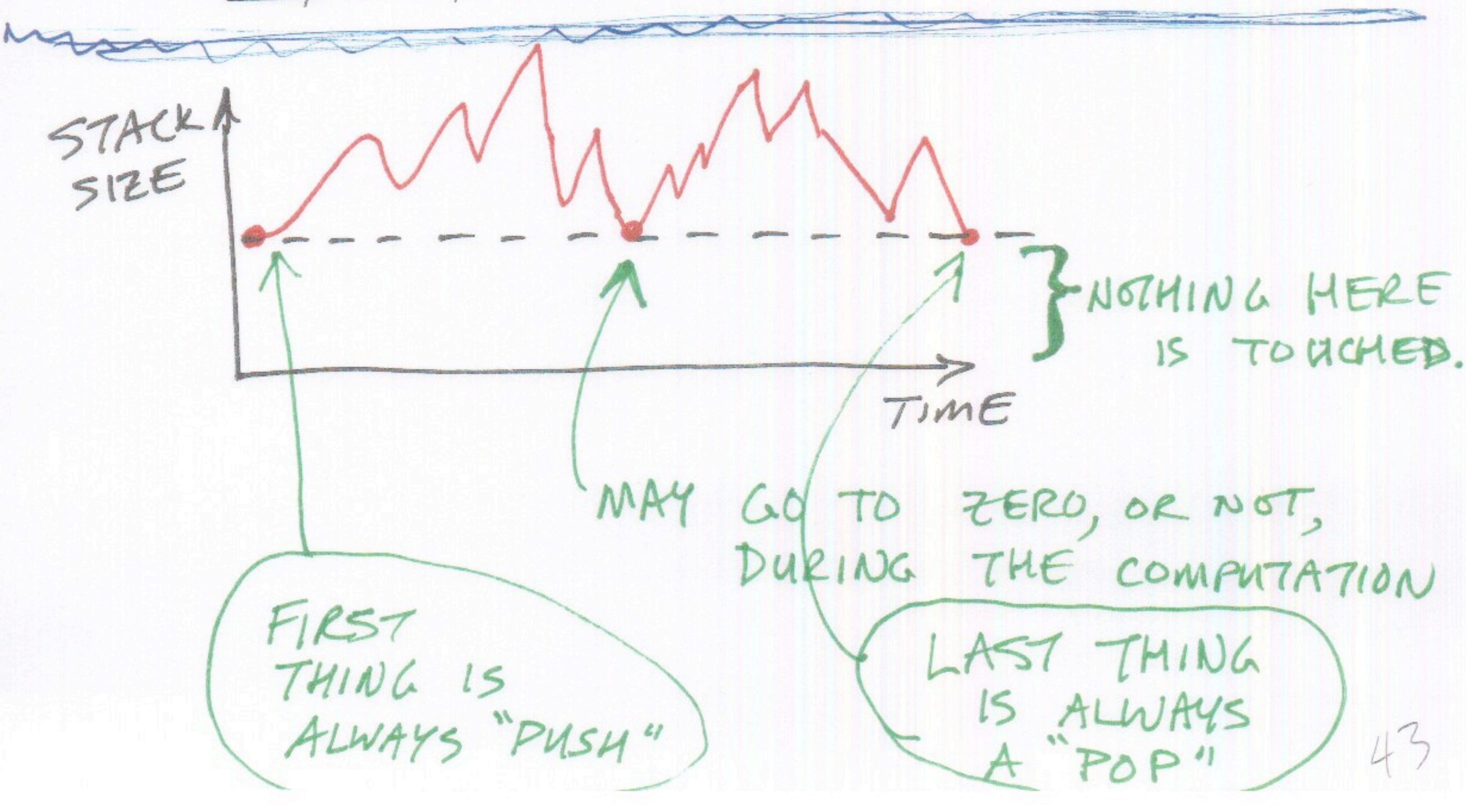


"DON'T MODIFY THE STACK"

= "START w/ AN EMPTY STACK AND FINISH WITH AN EMPTY STACK"

= "DON'T TOUCH THE STACK".





43

MAIN IDEA.

Consider two states p and q in the PDA.

Could we go from p to q without touching the stack?

What strings would do that?

That is:

Starting w/ an empty stack, We could go from p to q and end up with an empty stack.

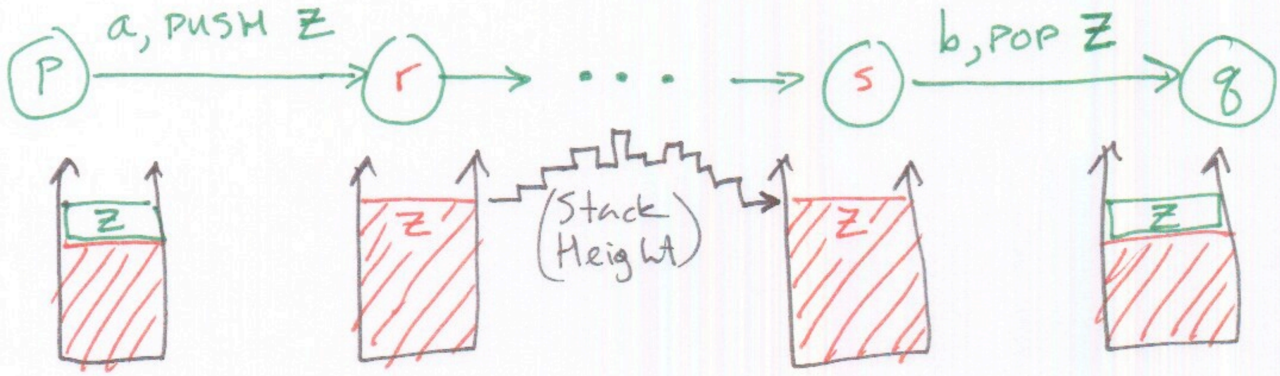
Or if somethings were on the stack they would never be touched.

The grammar we build will have a non-terminal

⇒ We'll call it A_{pq}

that will generate exactly these strings!

44

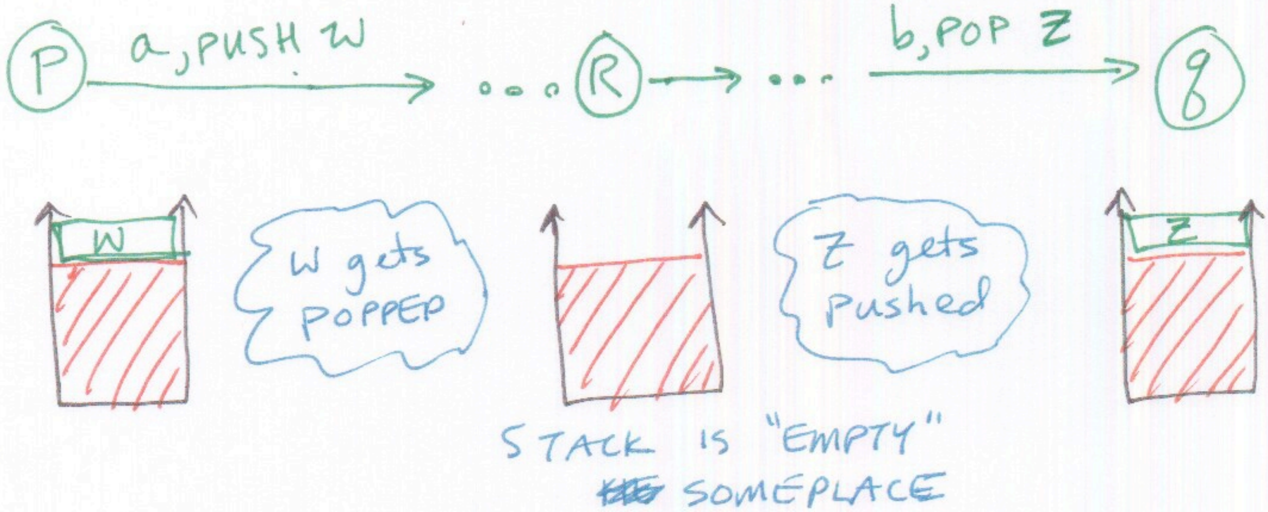


What strings can be generated/accepted by following this path?

"a ... b"

$$A_{pq} \rightarrow a \; \underline{A_{rs}} \; b$$

This rule will generate exactly those strings!



What strings can be generated by following this path?

$$\text{"}\underbrace{aaaa\ldots a}_{\text{From } p \text{ to } r}\,\underbrace{b\ldots bb}_{\text{From } r \text{ to } q}\text{"}$$

$$A_{pq} \rightarrow A_{pr}\, A_{rq}$$

This rule will generate exactly those strings!

If we have these edges

$$p \xrightarrow[\text{PUSH}]{a,\ \epsilon \to t} r \qquad\qquad s \xrightarrow[\text{POP}]{b,\ t \to \epsilon} q$$

And we could get from r to s without touching the stack,
Then we need a grammar rule:

$$A_{pq} \longrightarrow a\, A_{rs}\, b$$

FOR EACH $p, q, r, s \in Q$ in the PDA,
such that $\delta(p, a, \epsilon)$ contains (r, t)
and $\delta(s, b, t)$ contains (q, ϵ)
[i.e., "and the edges are labelled as above, for any $a, b \in \Sigma$ and $t \in \Gamma$."]
THEN ADD THIS RULE TO THE CFG:

$$A_{pq} \rightarrow a\, A_{rs}\, b$$

47

If we have a way to get from state p to state r that doesn't touch the stack

$$p \xrightarrow[A_{pr}]{} r$$

And a way to get from r to q that doesn't touch the stack.

$$r \xrightarrow[A_{rq}]{} q$$

THEN We have a new way to get from p to q without touching the stack.

For every state $p, r, q \in Q$

Add this rule to the grammar:

$$A_{pq} \rightarrow A_{pr} A_{rq}$$

48

There is a trivial way to get from state p to itself without touching the stack: The string ϵ.

So add

$$A_{pp} \rightarrow \epsilon \quad \text{for every state } p.$$

If the PDA accepts some string then there is a way to go from q_0 to q_{ACCEPT} that does not ~~&~~ modify the stack.

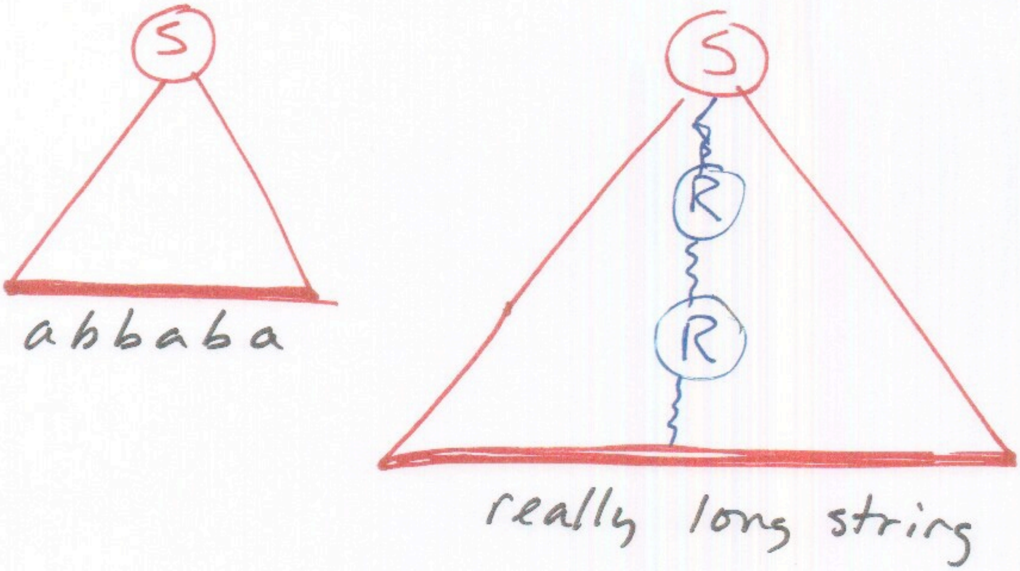
The grammar we seek should ~~accept~~ generate exactly these strings.

~~Our~~ OUR START NON-TERMINAL IS:

$$A_{q_0 q_{ACCEPT}}$$

49

PUMPING LEMMA FOR CFG's



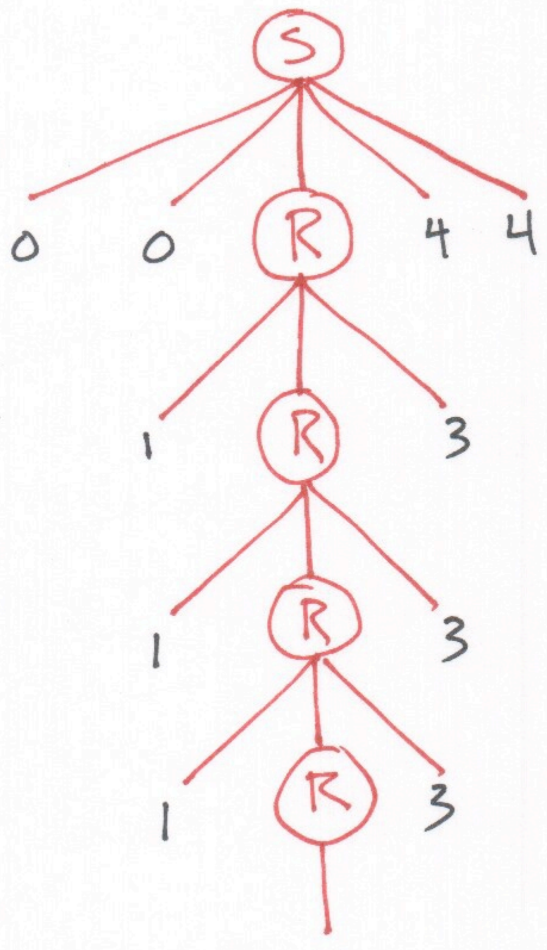
SOME NON-TERMINAL "R" MUST BE USED MORE THAN ONCE.

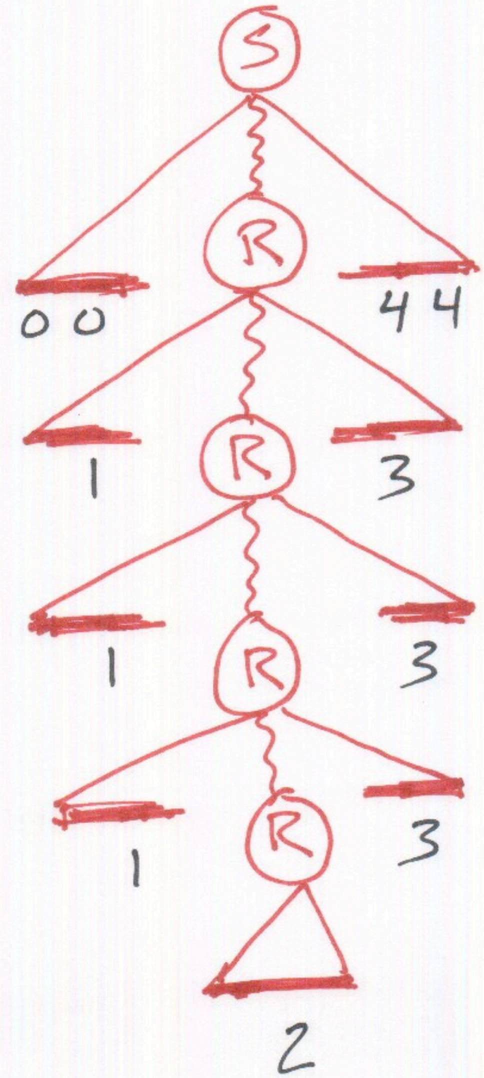
50

CONSIDER: $L = \{ 001^N 23^N 44 \mid N \geq 0 \}$

$$\boxed{\begin{array}{l} S \rightarrow 00R44 \\ R \rightarrow 1R3 \mid 2 \end{array}}$$

HOW CAN WE GENERATE "REALLY LONG" STRINGS?





51

$$S \rightarrow 00R44$$
$$R \rightarrow 1R3 \mid 2$$

$$\underbrace{00}_{u}\;\underbrace{2}_{x}\;\underbrace{44}_{z}$$

$$\underbrace{00}_{u}\;\underbrace{1}_{v}\;\underbrace{2}_{x}\;\underbrace{3}_{y}\;\underbrace{44}_{z}$$

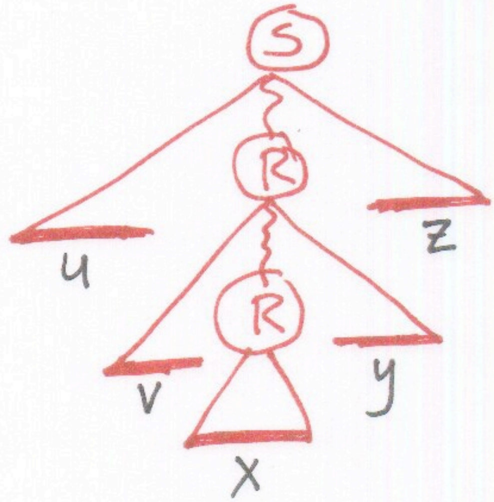
$$\underbrace{00}_{u}\;\underbrace{1111}_{v^4}\;\underbrace{2}_{x}\;\underbrace{3333}_{y^4}\;\underbrace{44}_{z}$$

uv^ixy^iz

is also in the Language!

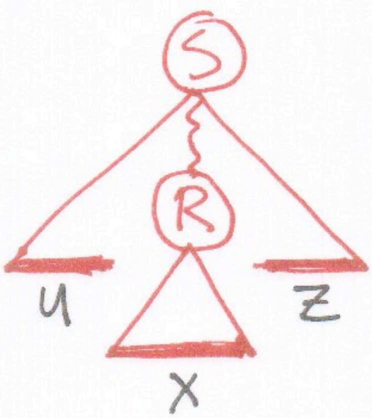
52

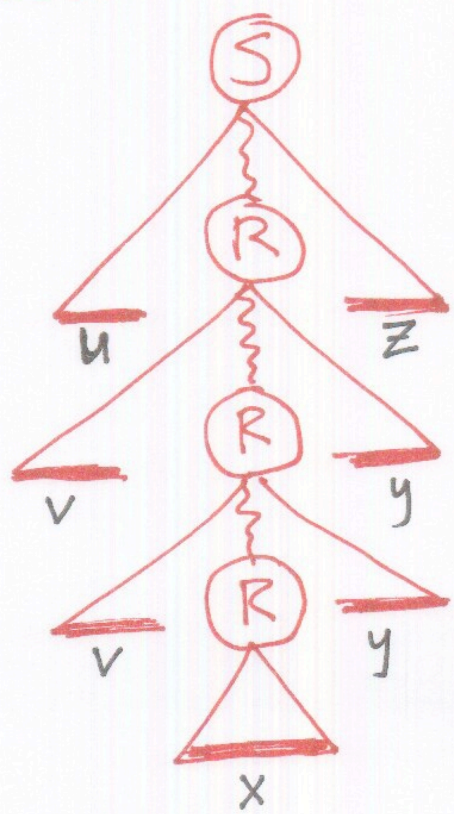
FOR ALL STRINGS THAT ARE "LONG ENOUGH", SOME NON-TERMINAL HAS TO BE REPEATED IN THE PARSE TREE.



$R \overset{*}{\Rightarrow} vRy$

THEREFORE, THESE ARE ALSO LEGAL PARSE TREES:





uv^ixy^iz
is also in the language!

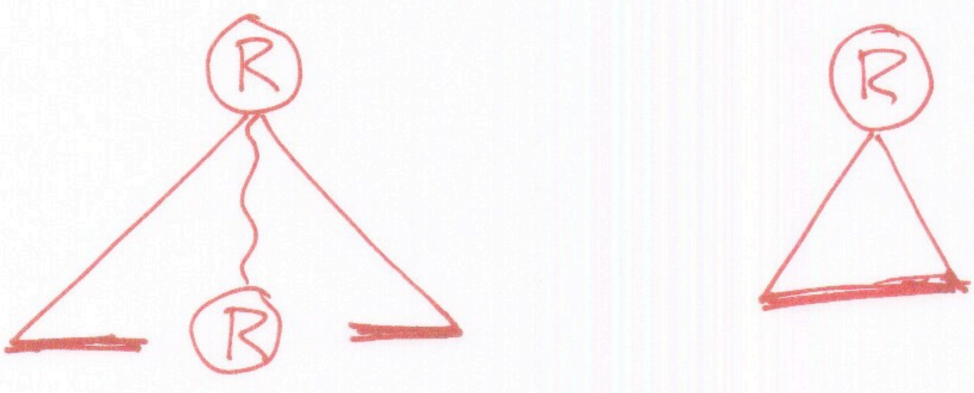
53

IN OTHER WORDS, TO GET LONG STRINGS
WE MUST USE RECURSION
IN THE GRAMMAR.

$$R \overset{*}{\Longrightarrow} \ldots R \ldots$$

AND FINALLY TO FINISH:

$$R \overset{*}{\Longrightarrow} x$$



54

PUMPING LEMMA FOR CFG's

If a string is sufficiently long, $|s| \geq p$
then it can be pumped.
That is, the string can be
broken into parts (someway)

$$s = uvxyz$$

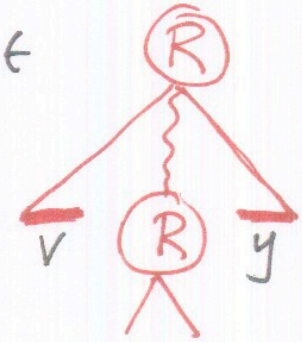
such that all strings of the form

$$uv^ixy^iz$$

are also in the language.

NOTES:

- v and y cannot both be ϵ
  $|vy| > 0$
- The beginning of v and the end of y can't be too far apart
  $|vxy| \leq p$



PUMPING LEMMA FOR CFG'S (REWORDING IT)

IF A IS A CONTEXT-FREE LANGUAGE, THEN THERE IS A PUMPING LENGTH p SUCH THAT, FOR ANY STRING IN A WHOSE LENGTH IS LONG ENOUGH, $|s| \geq p$, THAT STRING CAN BE BROKEN INTO PIECES

$$s = uvxyz$$

IN A WAY THAT SATISFIES ALL THREE OF THESE CONDITIONS:

CONDITION 1:

uv^ixy^iz is in A, for all $i \geq 0$

CONDITION 2:

$|vy| > 0$

CONDITION 3:

$|vxy| \leq p$

56

LOGIC REFRESHER

HOW CAN WE NEGATE

"FOR ALL"

$$\sim \forall x.\ p(\ldots) \equiv \exists x.\ \sim p(\ldots)$$

"THERE EXISTS"

$$\sim \exists x.\ p(\ldots) \equiv \forall x.\ \sim p(\ldots)$$

"AND"

$$\sim (\ldots P \ldots \land \ldots Q \ldots) \equiv (\sim(\ldots P \ldots) \lor \sim(\ldots q \ldots))$$

"It's not the case that all numbers are even."

$\Updownarrow$

"There exists a number that is not even."

"It's not the case that there exists a green number."

$\Updownarrow$

"All numbers have the "NOT-GREEN" property."

57

PUMPING LEMMA LOGIC

If L is a context-free language....

PUMPING PROPERTY

$\exists p$

$\forall s$ in L where $|s| \geq p$

$\exists uvxyz = s$

Such that

1. $uv^ixy^iz \in L, \forall i \geq 0$ AND
2. $|vy| > 0$ AND
3. $|vxy| \leq p$

To show L is not context-free, we must show that $\sim$(PUMPING PROPERTY) holds.

NOT-PUMPING PROPERTY

$\forall p$

$\exists s$ in L where $|s| \geq p$

$\forall uvxyz = s$

Such That

1. $uv^ixy^iz \notin L, \forall i \geq 0$ OR
2. $|vy| \not> 0$ OR
3. $|vxy| \not\leq p$

58

SHOW $B = \{a^N b^N c^N \mid N \geq 0\}$
IS NOT CONTEXT-FREE.

Assume it is CFL. Show ~(PUMPING PROPERTY)

Let p be the pumping length.
(No constraints on P. We'll show it $\forall p$.)

There exists a string.... $|s| \geq p$
We'll use: $a^P b^P c^P$ [$\exists s \ldots$]

Now look at all ways to divide it up. [$\forall uvxyz = s \ldots$]

~~Som~~ We'll show some condition will always be violated.

Condition (2) says $|vy| > 0$

Assume this condition ~~is~~ holds.
Now look at two cases.

~COND 1 OR ~COND 2 OR ~COND 3

~cond 2

COND 2

show ~cond 1

59

CASE 1

v and y each contain only one type of symbol.

$$a\,\underbrace{a\,a}_{v}\,a\quad b\,b\,b\,b\quad c\,\underbrace{c}_{y}\,c\,c$$

$$a\,a\,a\,a\quad \underbrace{b\,b\,b\,b}_{v}\quad c\,\underbrace{\;}_{y=\epsilon}\,c\,c\,c$$

One symbol will always be left out.

Pump s to uv^2xy^2z

$$a\,\underbrace{a\,a}_{v}\,\underbrace{a\,a}_{v}\,a\quad b\,b\,b\,b\quad c\,\underbrace{c}_{y}\,\underbrace{c}_{y}\,c\,c$$

At least one symbol will increase in ~~num~~ number.

At least one symbol will not increase in number.

The string cannot still be in the form

$$a^N b^N c^N$$

60

CASE 2

Either v or y has more than one kind of symbol:

$$a\underbrace{aabb}_{v}\underbrace{b}_{y}ccc$$

$$a\underbrace{a}_{v}abb\underbrace{bc}_{y}cc$$

Pump to uv^2xy^2z. We might have right number of symbols, but the order will be wrong.

$$a\underbrace{a}_{v}\underbrace{a}_{v}abb\underbrace{bc}_{y}\underbrace{bc}_{y}cc$$

Show $D = \{ww \mid w \in \{0,1\}^*\}$
is not context-free.

Assume it is a CFL.
Show ~(PUMPING PROPERTY)
Let p be the pumping length.
[No constraints on P. Show $\forall p$]
There exists a string s, $|s| \geq p$ . . .
[To show $\exists s$, just provide an
example; just give ~~this~~ an s that exists.]

$0^p 1^p 0^p 1^p$

Look at all ways to divide s into parts.
[Show $\forall uvxyz = s$]
Must show that some condition~~s~~
is not satisfied.
We'll assume condition 3 holds

$|vxy| \leq p$

and show that the other conditions
must fail.

62

CONSIDER THE BOUNDARIES BETWEEN 0's AND 1's IN OUR STRING.

MIDPOINT

$\underbrace{00000}_{P}\ \underbrace{11111}_{P}\ \underbrace{00000}_{P}\ \underbrace{11111}_{P}$

P symbols in length

BOUNDARIES

$\boxed{\text{CASE 1:}}$ vxy does not straddle a boundary.

$00000\ 1\underbrace{111}_{vxy}1\ 00000\ 11111$

PUMPING UP WILL YIELD A STRING WITH MORE 1'S AND AN "IMBALANCE"

NEW MIDPOINT.

$\underbrace{00000}_{P}\ \underbrace{11\cdots\cdots11}_{\text{MORE THAN P}}\ \underbrace{00000}_{P}\ \underbrace{11111}_{P}$

THE STRING NOW HAS THE FORM

NEW MIDPOINT.

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

This string is not of the form ww.

Since uv^2xy^2z is <u>not</u> in the language, CONDITION 1 is violated.

63

$\boxed{\text{CASE 2:}}$ vxy straddles the first boundary.

$000\underbrace{00\ \{\ 111}_{vxy}11\ \{\ 00000\ \{\ 11111$

MIDPOINT.

SINCE $|vxy| \leq p$ it cannot straddle the midpoint.

PUMPING DOWN WILL MAKE THE STRING SHORTER.

NEW MIDPOINT

$000\underbrace{\qquad\qquad}_{\text{FEWER SYMBOLS}}11\ \underbrace{000000}_{p}\ \underbrace{11111}_{p}$

NOTE: The string now has the form:

NEW MIDPOINT.

_ _ _ _ _ _ _ _ 0 _ _ _ _ _ _ _ _ 1

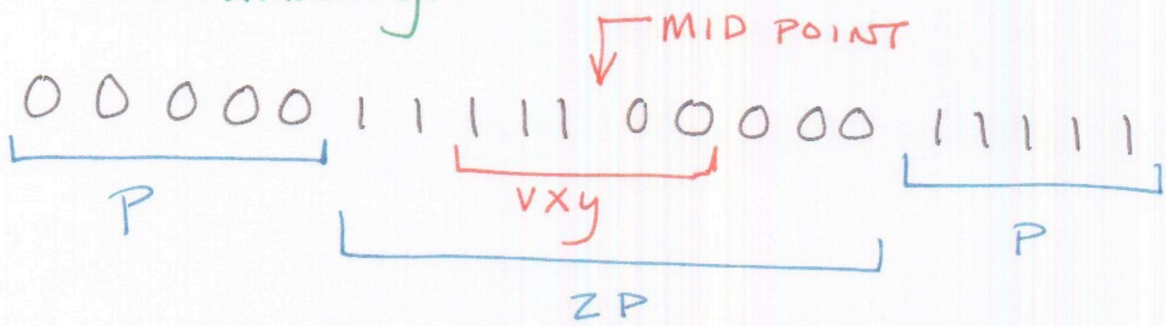
The string is not of the form WW.

$\boxed{\text{CASE 2b:}}$ vxy straddles the third boundary: It is similar.

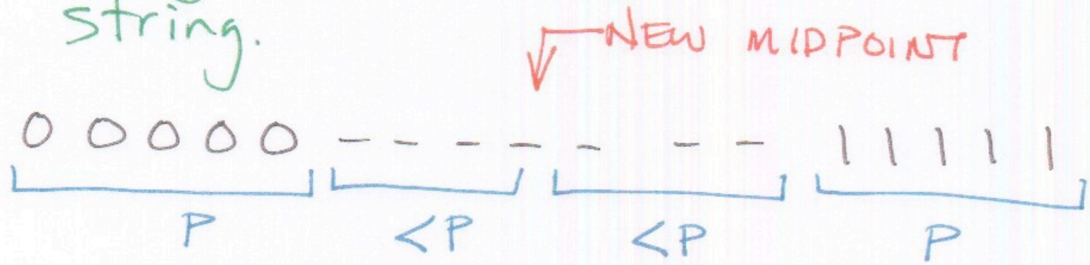
$00000\ \{\ 11111\ \{\ 000\underbrace{00\ \}\ 111}_{vxy}11$

CASE 3: vxy straddles the midpoint.

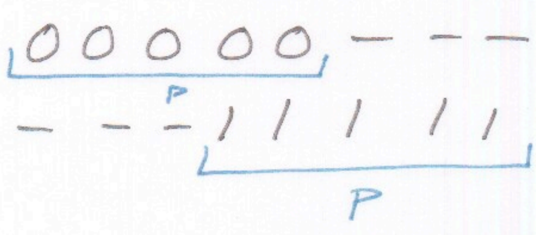
Since $|vxy| \le p$, it cannot also straddle the first or third boundary.



Pumping down will give us a shorter string.



Look at the first half of the string and the second half.



They cannot be equal.
This string fails condition 1.

65