

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

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

$$E \rightarrow E + T$$

$$E \rightarrow T$$

VARIABLE (= "NON TERMINALS")

TERMINALS (SYMBOLS FROM ALPHABET)

RULES (= "PRODUCTIONS")

$$E ::= E + T$$

START VARIABLE. Often "S"

Assume: First rule uses start symbol.

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

DERIVATION

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

WE WRITE:

$$E \xrightarrow{*} a + (a \times a)$$

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

LEFT-MOST DERIVATION:

Always choose Left-most Variable

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

RIGHT-MOST DERIVATION:

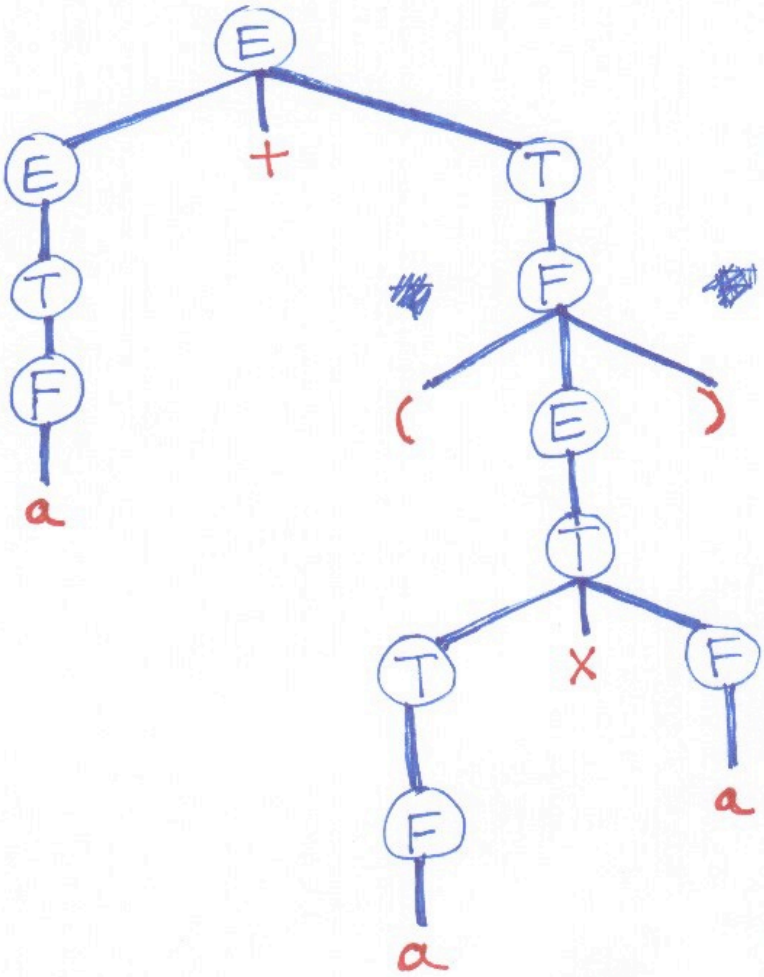
Always choose Right-most Variable

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

PARSE TREES

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

$E \Rightarrow^* a+(axa)$



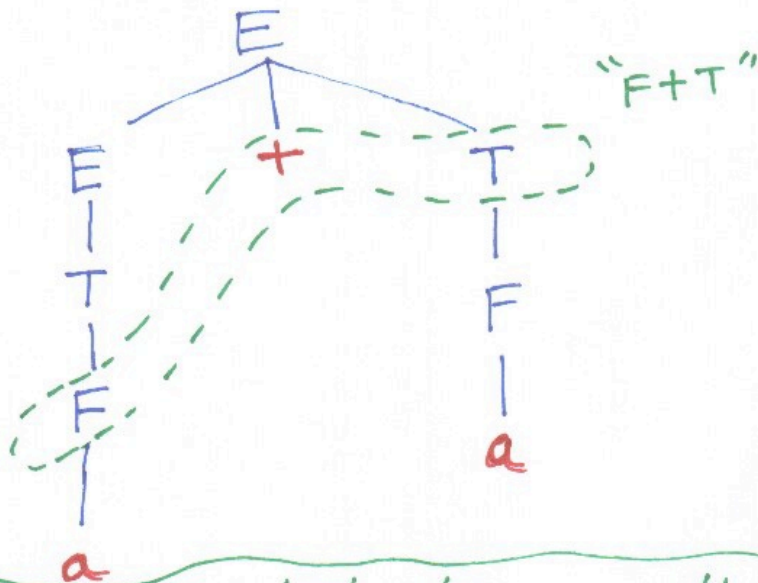
YOU GET THE SAME RESULT:

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

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

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

PARSE TREE:



The parse tree abstracts away the actual order in which the rules are used. It only remembers which 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
 $\{w \mid w \in \Sigma^* \text{ and } S \xRightarrow{*} w\}$

DEFINITION

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

EXAMPLE CFG

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

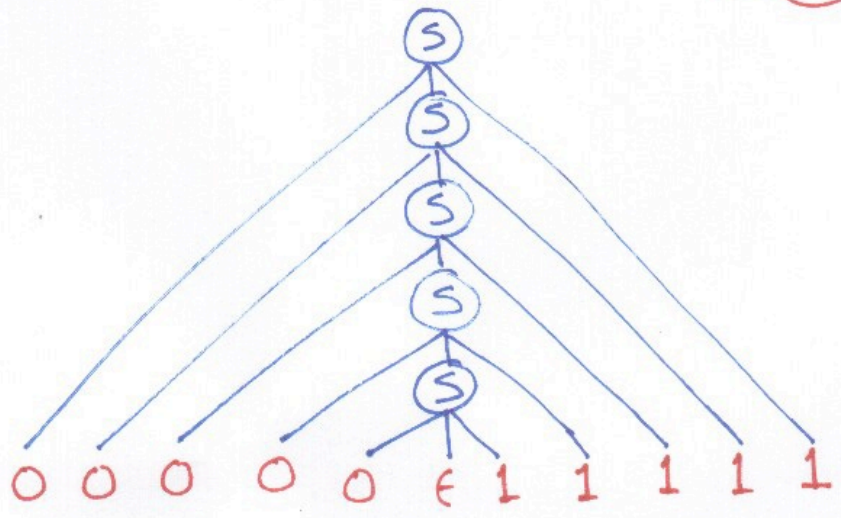
Language = $\{\epsilon, (), ()(), ((\)()), ((\)())((\)()), ((\)()), ((\)()()()()), \dots\}$

EXAMPLE

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

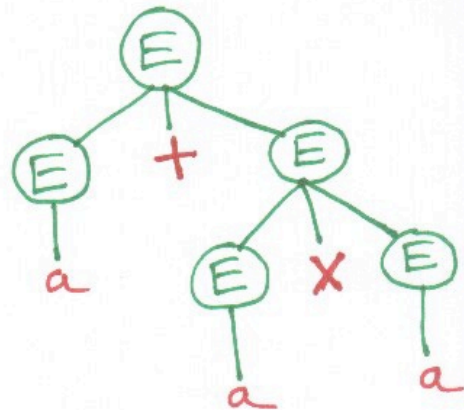
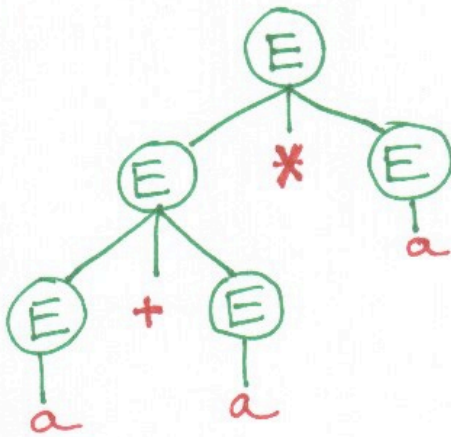
$$S \rightarrow \epsilon$$
$$S \rightarrow 0S1$$

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



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

$a + a \times a$



- 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.

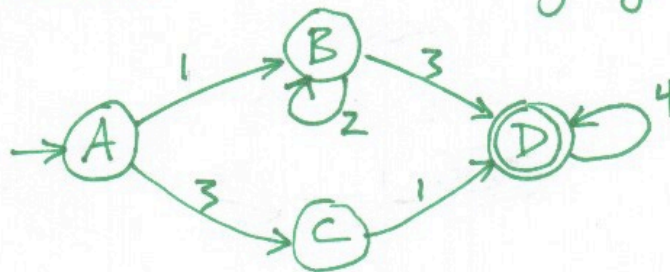
AMBIGUOUS GRAMMAR

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

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.

$$A \rightarrow 1B$$

$$A \rightarrow 3C$$

$$B \rightarrow 2B$$

$$B \rightarrow 3D$$

$$C \rightarrow 1D$$

$$D \rightarrow 4D$$

$$D \rightarrow \epsilon$$

THE LANGUAGE ONION:

All Languages

Turing Recognizable Languages
(TM will halt if "yes", but may loop if "no")

Decidable Languages
(TM will always halt)

Context-Free Languages
(Nondeterministic Pushdown Automaton)

Unambiguous Languages

LR(k) Languages
(Deterministic Pushdown Automaton)

LL(k) Languages
(Predictive Parser)

Regular Languages
(Finite State Machine, Reg. Expr.)

CHOMSKY NORMAL FORM

EVERY RULE IN THE
GRAMMAR HAS THE FORM

$$A \rightarrow BC$$

OR

$$A \rightarrow a$$

WE CAN ALSO HAVE

$$S \rightarrow \epsilon$$

EXACTLY TWO
VARIABLES;
CAN'T BE "S"

OR A
TERMINAL
SYMBOL.

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.

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

STEP 2:

Can't have $A \rightarrow \epsilon$. (unless A is start variable)
Remove such rules.

$$B \rightarrow \underline{BCACB} \underline{AB}$$



$$B \rightarrow BCACB \underline{AB}$$

$$B \rightarrow BC \underline{C} B \underline{AB}$$

$$B \rightarrow BC \underline{A} C B \underline{B}$$

$$B \rightarrow BC \underline{C} B \underline{B}$$

} Add these new rules.

$$\begin{aligned} S_0 &\rightarrow S \\ S &\rightarrow \underline{A} S A \mid a \underline{B} \\ A &\rightarrow \underline{B} \mid S \\ \underline{B} &\rightarrow \underline{b} \mid \underline{\epsilon} \end{aligned}$$

Remove $B \rightarrow \epsilon$

$$\begin{aligned} S_0 &\rightarrow S \\ S &\rightarrow \underline{A} S A \mid a B \mid a \\ \underline{A} &\rightarrow B \mid S \mid \underline{\epsilon} \\ B &\rightarrow b \end{aligned}$$

Remove $A \rightarrow \epsilon$

$$\begin{aligned} S_0 &\rightarrow S \\ S &\rightarrow \underline{A} S A \mid a B \mid a \mid \underline{.SA} \mid \underline{AS} \mid S \\ A &\rightarrow B \mid S \\ B &\rightarrow b \end{aligned}$$

STEP 3:

GET RID OF ALL "UNIT RULES"

$$A \rightarrow B$$

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

~~ADD: $C \rightarrow xyz$~~

FROM BEFORE:

$$\begin{aligned} S_0 &\rightarrow S \\ S &\rightarrow ASA \mid aB \mid a \mid SA \mid AS \mid \underline{S} \\ A &\rightarrow B \mid S \\ B &\rightarrow b \end{aligned}$$

Remove $S \rightarrow S$ (do nothing)

$$\begin{aligned} \del{S_0} &\rightarrow S \\ S &\rightarrow ASA \mid aB \mid a \mid SA \mid AS \\ A &\rightarrow B \mid S \\ B &\rightarrow b \end{aligned}$$

Remove $S_0 \rightarrow S$

$$\begin{aligned} S_0 &\rightarrow ASA \mid aB \mid a \mid SA \mid AS \\ S &\rightarrow ASA \mid aB \mid a \mid SA \mid AS \\ A &\rightarrow B \mid S \\ B &\rightarrow b \end{aligned}$$

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 $A \rightarrow S$:

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

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

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

$$B \rightarrow b$$

STEP 4:

Replace

$$A \rightarrow BCDE$$

With

$$A \rightarrow BA_1$$

$$A_1 \rightarrow CA_2$$

$$A_2 \rightarrow D \del{A} E$$

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

FROM BEFORE:

$$S_0 \rightarrow \underline{ASA} \mid aB \mid a \mid SA \mid AS$$

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

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

$$B \rightarrow b$$

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

$$S_0 \rightarrow AA_1 \mid aB \mid a \mid SA \mid AS$$

$$S \rightarrow AA_1 \mid aB \mid a \mid SA \mid AS$$

$$A \rightarrow b \mid AA_1 \mid aB \mid a \mid SA \mid AS$$

$$A_1 \rightarrow SA$$

$$B \rightarrow b$$

Step 4b:

Replace $A \rightarrow bC$

with: $A \rightarrow A_1C$

$A_1 \rightarrow b$

FROM BEFORE:

$S_0 \rightarrow AA_1 \mid \underline{a}B \mid a \mid SA \mid AS$

$S \rightarrow AA_1 \mid \underline{a}B \mid a \mid SA \mid AS$

$A \rightarrow b \mid AA_1 \mid \underline{a}B \mid a \mid SA \mid AS$

$A_1 \rightarrow SA$

$B \rightarrow b$

INTRODUCE ~~$A_1 \rightarrow a$~~ $A_2 \rightarrow a$

$S_0 \rightarrow AA_1 \mid A_2B \mid a \mid SA \mid AS$

$S \rightarrow AA_1 \mid A_2B \mid a \mid SA \mid AS$

$A \rightarrow b \mid AA_1 \mid A_2B \mid a \mid SA \mid AS$

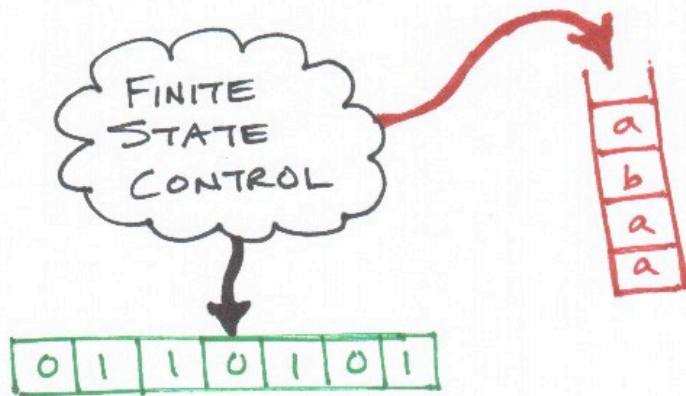
$A_1 \rightarrow SA$

$B \rightarrow b$

$A_2 \rightarrow a$

THIS GRAMMAR
IS NOW IN
CHOMSKY
NORMAL
FORM!

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.

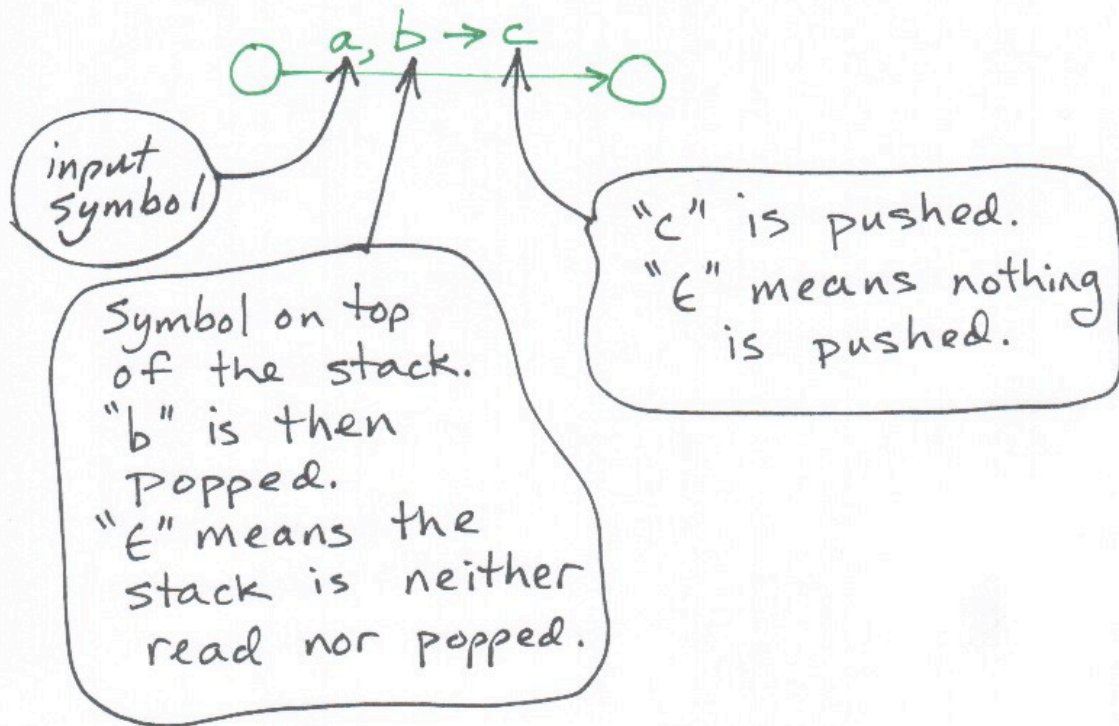
MAY PUSH ONTO THE STACK.

NON-DETERMINISTIC!

FINITE STATE MACHINE



PUSHDOWN AUTOMATON

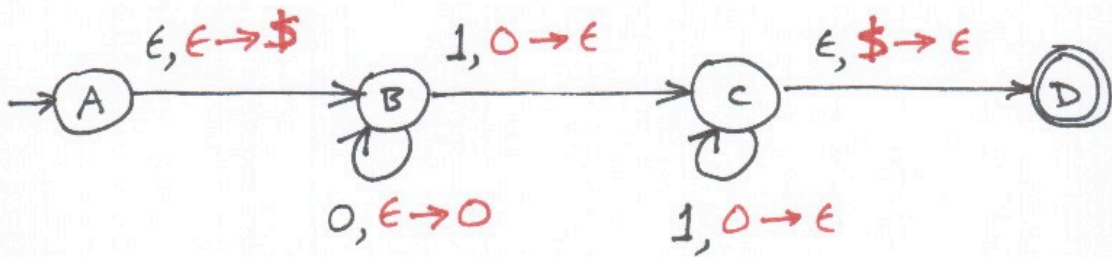


$$\{0^n 1^n \mid n \geq 0\}$$

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

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

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



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.



- NON-DETERMINISTIC:
You just have to find one path to a ACCEPT state.

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 $q_0 \in Q$

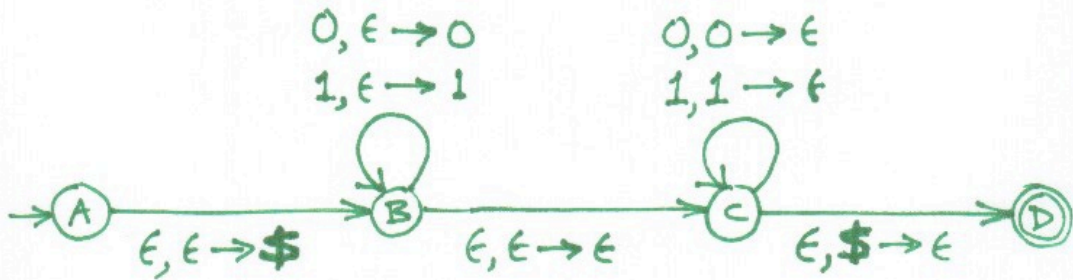
F : Accept States $F \subseteq Q$

EXAMPLE

$\{w \mid w \text{ is a palindrome}\}$

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

$S \rightarrow 0S0$
 $\rightarrow 1S1$
 $\rightarrow \epsilon$



GRAMMAR DESIGN CHALLENGE

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

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:

$S \rightarrow OA \mid 1B \mid \epsilon$

$A \rightarrow 1S \mid 0AA$

$B \rightarrow 0S \mid 1BB$

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^*

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

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

GENERATES:

ABABABAB...

EVERY A CAN GO TO ϵ .

EVERY B CAN GO TO ϵ .

\Rightarrow ANY STRING OF A'S AND B'S!

01010101
A A A A

00001111
S
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.

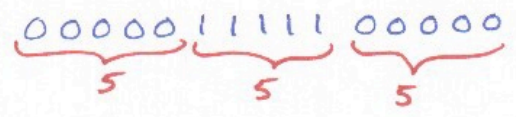
(THIS IS DECIDABLE.)

ACCEPTED BY GRAMMAR #2?

- FIND A COUNTER-EXAMPLE?
HALT; PRINT "NO EQUIVALENT".
- OTHERWISE KEEP LOOKING.
MAY NOT HALT... OR MAY...?
- NO WAY TO KNOW WHEN TO STOP LOOKING.

NON-CONTEXT-FREE GRAMMARS

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



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

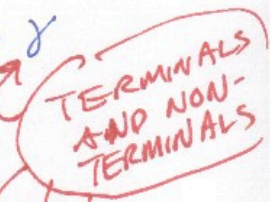
CONTEXT-FREE

$$A \rightarrow \gamma$$

TYPE-1 LANGUAGES

CONTEXT-SENSITIVE

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



TYPE-0 LANGUAGES

RECURSIVELY-ENUMERABLE
TURING-ENUMERABLE.
TURING-RECOGNIZABLE

ex: $ABx \rightarrow ADEy$

$$L = \{1^n 2^n 3^n \mid n \geq 0\}$$

APPROACH:

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

$$S \xrightarrow{*} \dots AAABBBCCC \xrightarrow{*} \dots 111 222 333$$

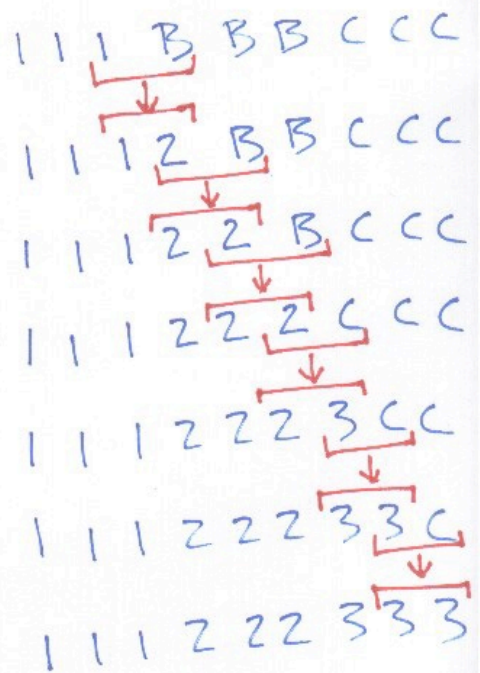
INITIALLY THE STRING WILL START WITH "1"s

111 ~~BBB~~ BBB CCC

Will never use "A" actually.

RULES TO FINISH IT UP:

- 1B → 12
- 2B → 22
- 2C → 23
- 3C → 33



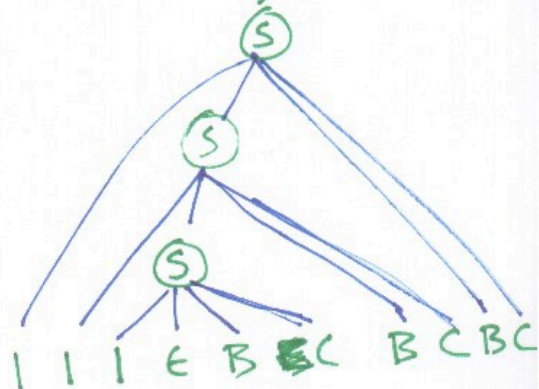
NOTE
 ... CB...
 CAN NEVER BE REDUCED.
 "C" CAN ONLY TURN INTO A "3".
 AND "3B" CAN NEVER BE REDUCED.
 ⇒ B'S MUST PRECEED "C'S".

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

$S \rightarrow 1SBC$

~~$S \rightarrow 1SBC$~~

$S \rightarrow \epsilon$



1111BCBCBCBC

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

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

$$\underline{S} \rightarrow \underline{1SBC}$$

$$\underline{S} \rightarrow \underline{\epsilon}$$

FROM
BEFORE

$$\underline{CB} \rightarrow \underline{HB}$$

$$\underline{HB} \rightarrow \underline{HC}$$

$$\underline{HC} \rightarrow \underline{BC}$$

REVISED

$$\underline{1B} \rightarrow \underline{12}$$

$$\underline{2B} \rightarrow \underline{22}$$

$$\underline{2C} \rightarrow \underline{23}$$

$$\underline{3C} \rightarrow \underline{33}$$

FROM
BEFORE

ARE CONTEXT-FREE LANGUAGES
CLOSED UNDER UNION?

GRAMMAR 1:

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

GRAMMAR 2:

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

UNION:

$S \rightarrow S_1 \mid S_2$

$S_1 \rightarrow \dots$

$S_2 \rightarrow \dots$

YES!

ARE CONTEXT-FREE LANGUAGES
CLOSED UNDER CONCATENATION?

$S \rightarrow S_1 S_2$

$S_1 \rightarrow \dots$

$S_2 \rightarrow \dots$

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 NOT A CONTEXT-FREE LANGUAGE.

No!

NOT IN GENERAL,
BUT FOR SOME LANGUAGES
THE INTERSECTION
MAY ALSO BE
CONTEXT-FREE.

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

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

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

L is not context-free.

HOWEVER

\bar{L} IS CONTEXT-FREE.

PALINDROMES
= $\{w^R w \mid w \in \Sigma^*\}$

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.

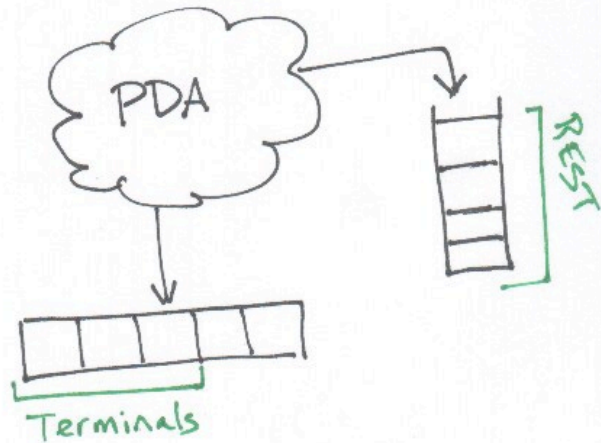
GIVEN: A GRAMMAR

$$\begin{aligned} S &\rightarrow BS \mid A \\ A &\rightarrow \epsilon \mid 0A \\ B &\rightarrow BB \mid 1 \end{aligned}$$

FIND (OR BUILD) A PDA.

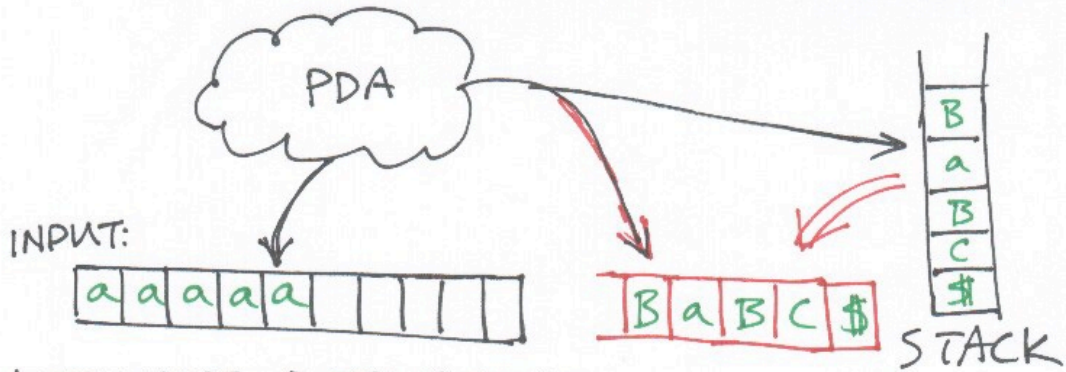
CONSIDER A DERIVATION: (LEFT-MOST)

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



GENERAL FORM:

$$\# \underbrace{aaaaa}_{\text{Terminals}} \underbrace{BaBBaCa}_{\text{Rest (both)}}$$



LEFT-MOST DERIVATION:

$S \Rightarrow \dots$ aaaaa

B	a	B	C
---	---	---	---

 $\Rightarrow \dots$

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

RULE:
 $B \rightarrow ASA \times BA$

$\dots \Rightarrow$ aaaaa

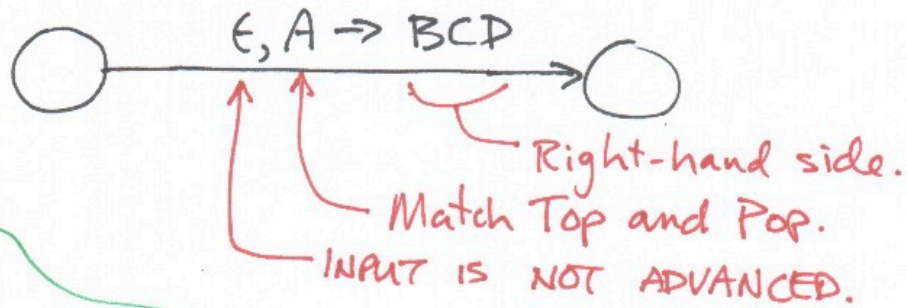
A	S	A	x	B	A	a	B	C
---	---	---	---	---	---	---	---	---

So:

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

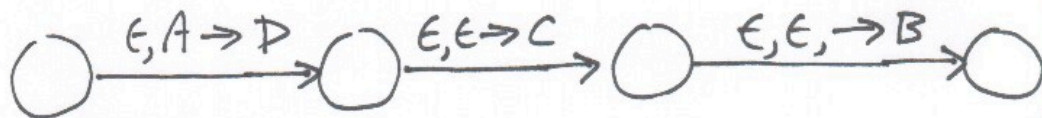
RULE: $A \rightarrow BCD$

Add this to PDA



NOTE:

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



WHICH RULE TO USE?

P.D.A.'s ARE NON-DETERMINISTIC!

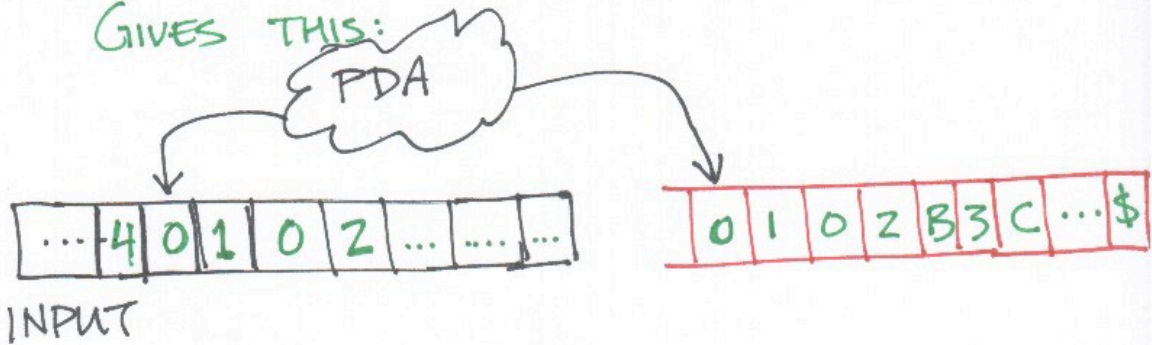
(TRY THEM ALL IN PARALLEL.)

(JUST CHOOSE THE "RIGHT" RULE.)

RULE :

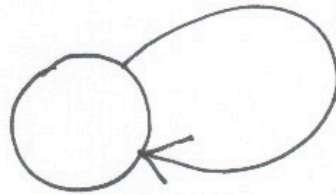
$A \rightarrow 0102B3C$

GIVES THIS:



So:

MATCH TERMINAL SYMBOLS TO
THE STACK TOP.



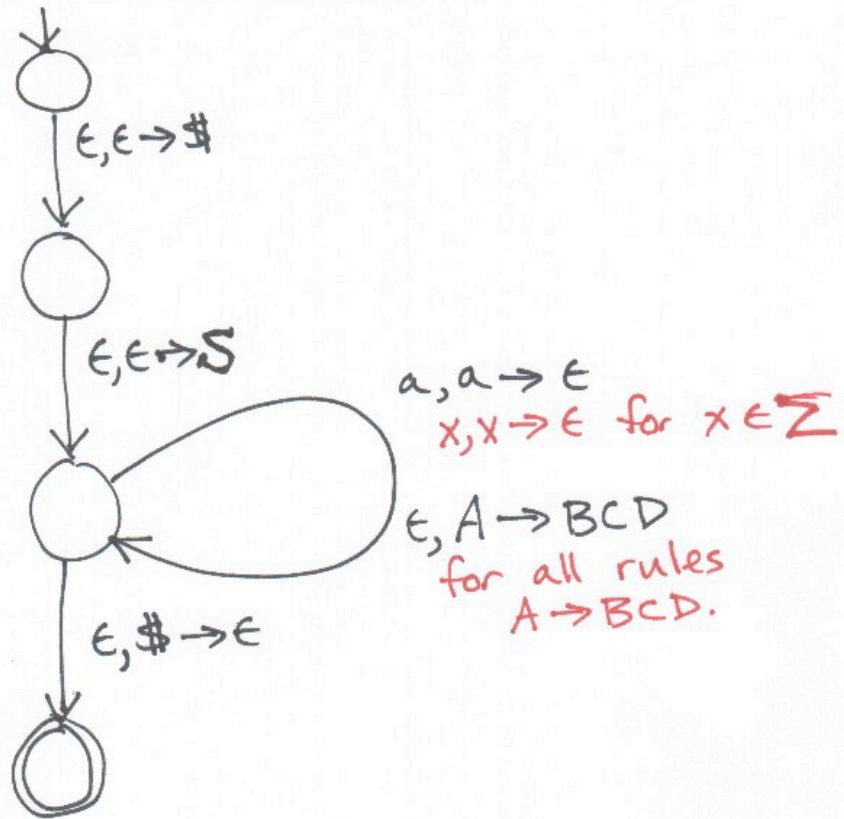
$0, 0 \rightarrow \epsilon$

$1, 1 \rightarrow \epsilon$

$2, 2 \rightarrow \epsilon$

etc. for all $x \in \Sigma$.

THE FINAL MACHINE:



PROOF, PART 2

WE ARE GIVEN A P.D.A.

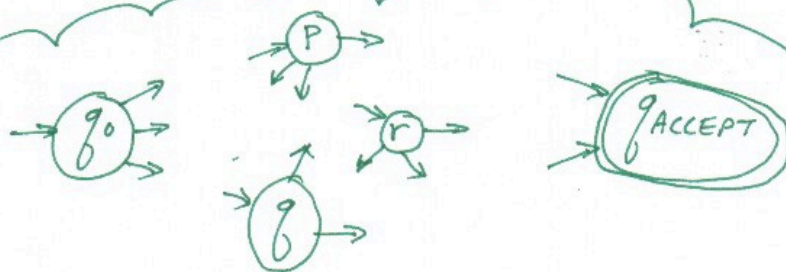
MUST BUILD A CFG FROM IT.

STEP 1

SIMPLIFY THE PDA.

STEP 2

BUILD THE CFG.



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

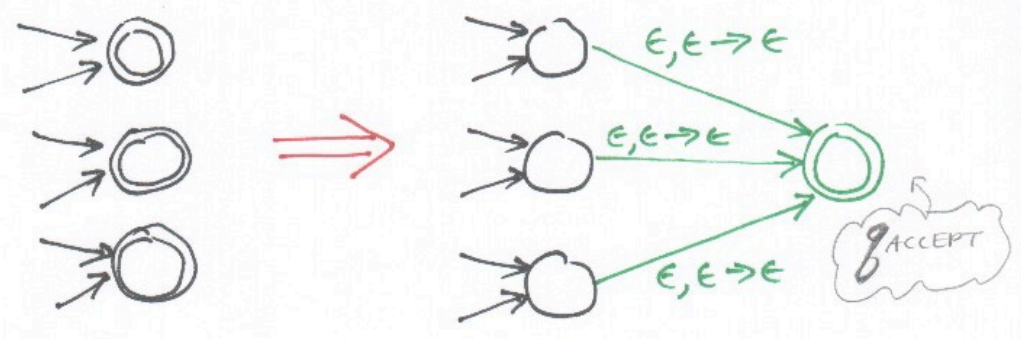
A_{pq} A_{qr} A_{rq} ...

The starting nonterminal will be

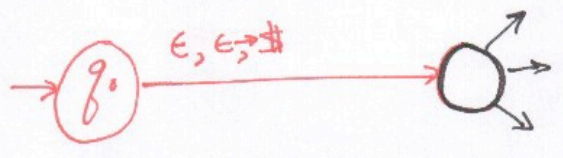
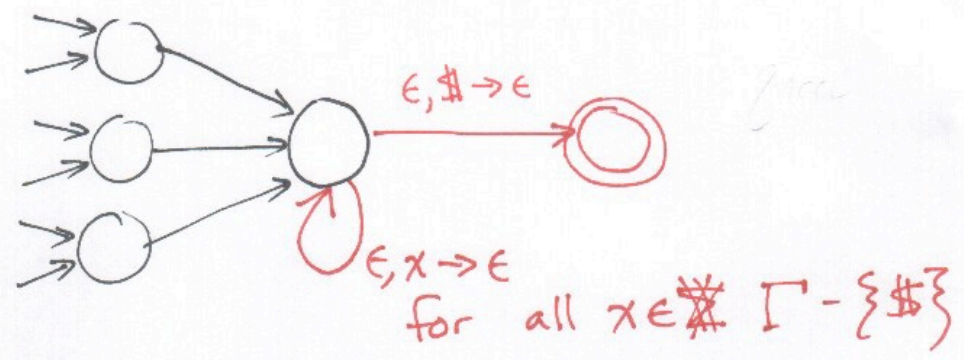
$A_{q_0 q_{ACCEPT}}$

SIMPLIFY THE PDA

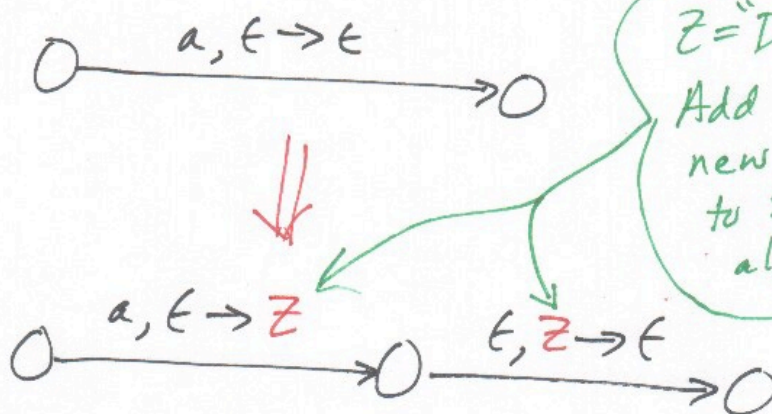
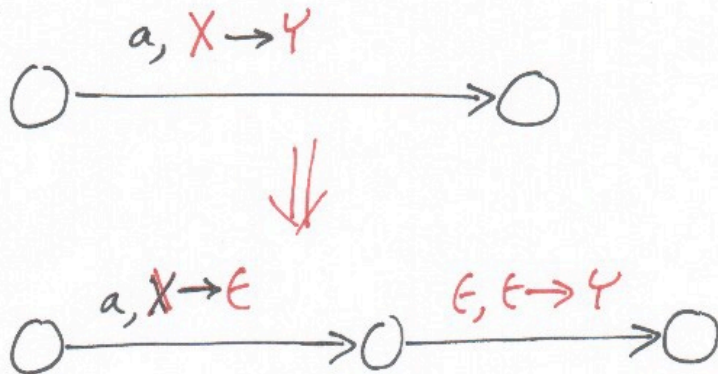
① The PDA has only one ACCEPT state.



② The PDA empties its stack before ACCEPTING.



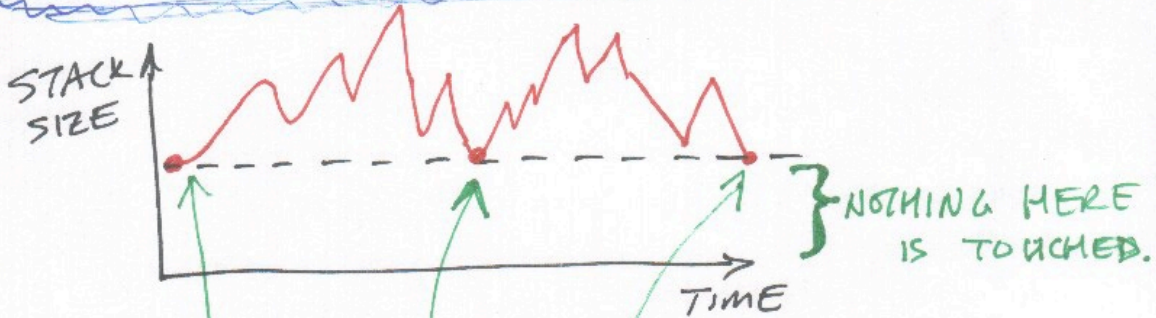
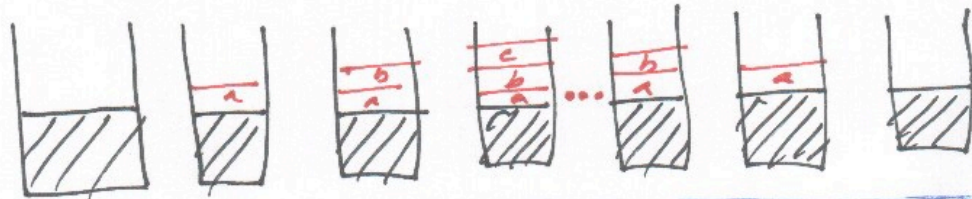
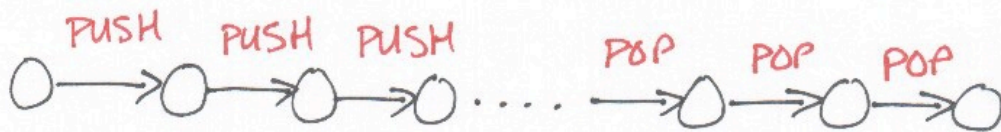
③ Each transition either PUSHES or POPS, but does not do both.



"DON'T MODIFY THE STACK"

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

= "DON'T TOUCH THE STACK"



FIRST
THING IS
ALWAYS "PUSH"

MAY GO TO ZERO, OR NOT,
DURING THE COMPUTATION

LAST THING
IS ALWAYS
A "POP"

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:

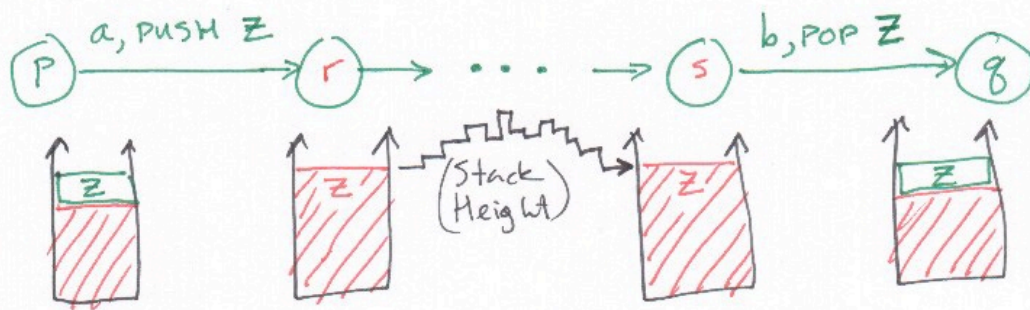
Starting w/ an empty stack,
We could go from p to q
and end up with an
empty stack.

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!

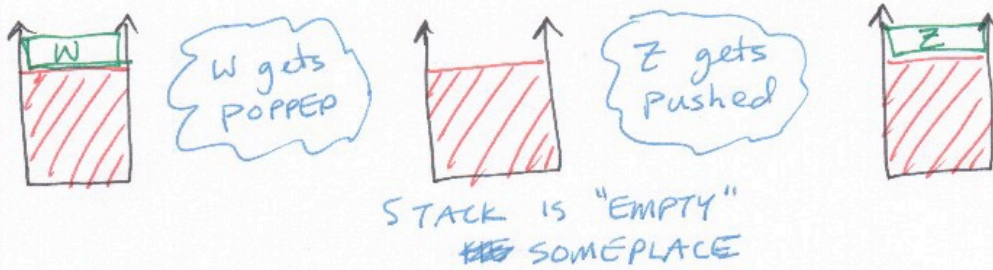
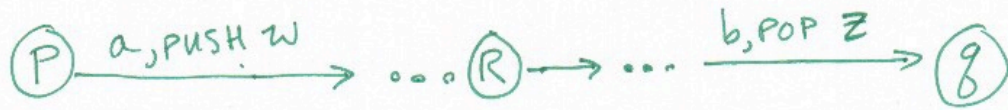


What strings can be generated/accepted by following this path?

"a...b"

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

This rule will generate exactly those strings!



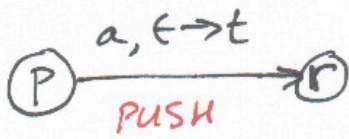
What strings can be generated by following this path?

$" \underbrace{aaaa \dots a}_{\text{From } (P) \text{ to } (R)} \mid \underbrace{b \dots bb}_{\text{From } (R) \text{ to } (Q)} "$

$A_p q \rightarrow A_p r A_r q$

This rule will generate exactly those strings!

If we have these edges



And we could get from \textcircled{r} to \textcircled{s}
without touching the stack,

Then we need a ~~new~~ grammar
rule:

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

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 $t \in \Gamma$!"]

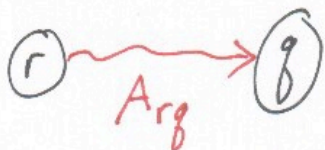
THEN ADD THIS RULE TO THE CFG:

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

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



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



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:

$$Apq \rightarrow Apr Arg$$

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

So add

$A_{pp} \rightarrow \epsilon$ for every state (p) .

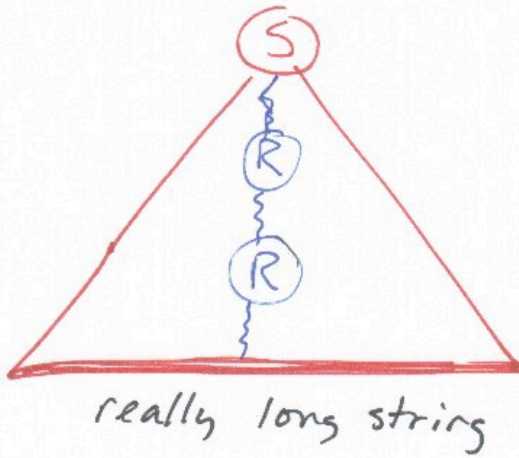
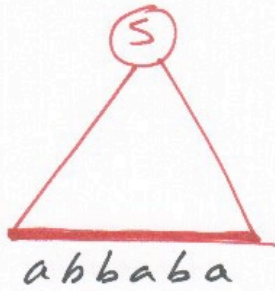
If the PDA accepts some string then there is a way to go from (q_0) to (q_{ACCEPT}) that does not ~~&~~ modify the stack.

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

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

$A_{q_0 q_{ACCEPT}}$

PUMPING LEMMA FOR CFG's

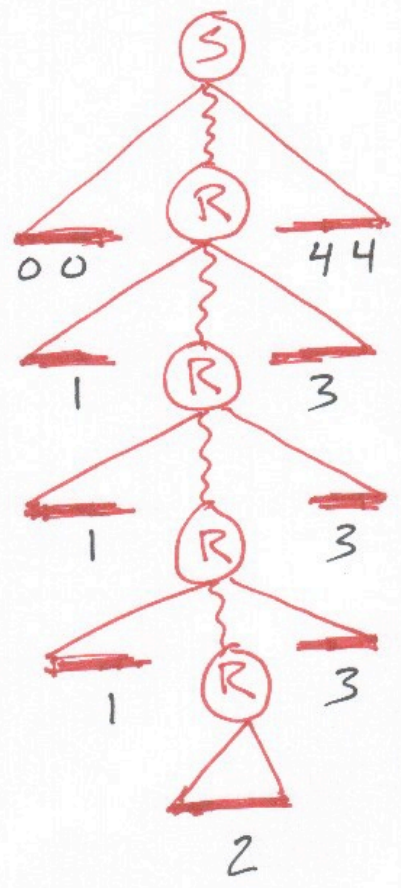
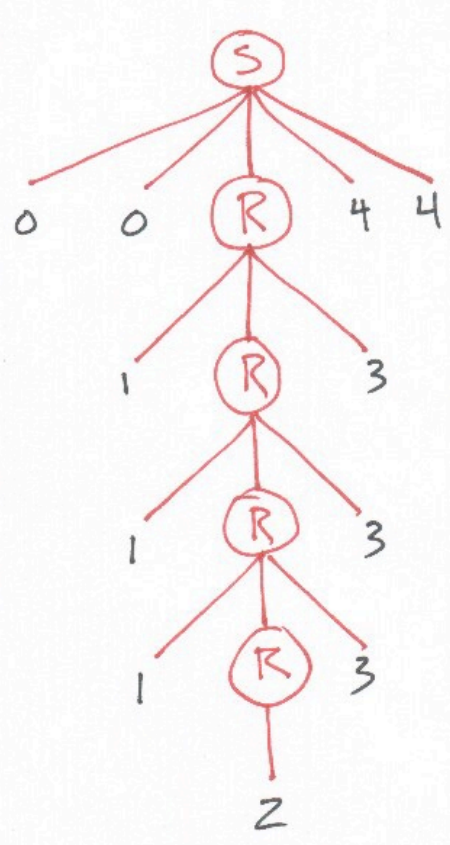


SOME NON-TERMINAL "R" MUST BE USED MORE THAN ONCE.

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

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

HOW CAN WE GENERATE "REALLY LONG" STRINGS?



$$\begin{aligned}
 S &\rightarrow 00R44 \\
 R &\rightarrow 1R3|2
 \end{aligned}$$

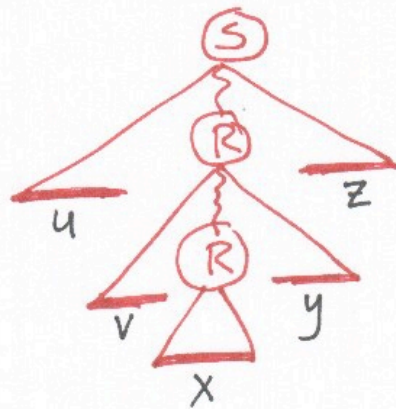
$$\begin{array}{c}
 \underline{00} \quad \underline{2} \quad \underline{44} \\
 u \quad x \quad z
 \end{array}$$

$$\begin{array}{c}
 \underline{00} \quad \underline{1} \quad \underline{2} \quad \underline{3} \quad \underline{44} \\
 u \quad v \quad x \quad y \quad z
 \end{array}$$

$$\begin{array}{c}
 \underline{00} \quad \underline{1111} \quad \underline{2} \quad \underline{3333} \quad \underline{44} \\
 u \quad v^4 \quad x \quad y^4 \quad z
 \end{array}$$

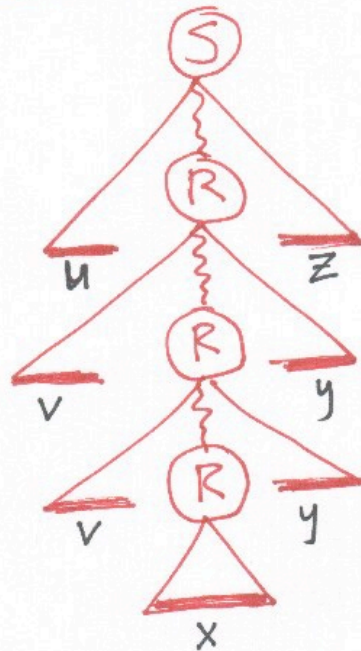
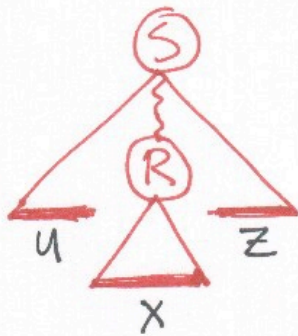
$uv^i x y^i z$
 is also in
 the Language!

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



$$R \Rightarrow^* vRy$$

THEREFORE, THESE ARE ALSO LEGAL
PARSE TREES:



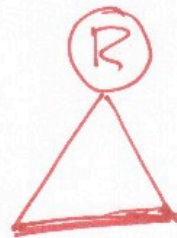
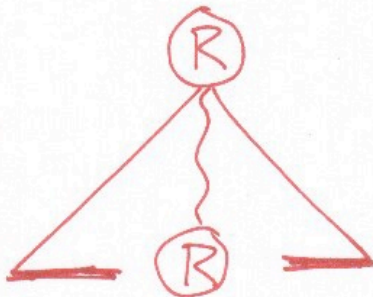
$uv^i xy^i z$
is also in
the language!

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

$$R \xrightarrow{*} \dots R \dots$$

AND FINALLY TO FINISH:

$$R \xrightarrow{*} x$$



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 (some way)

$$s = UVXYZ$$

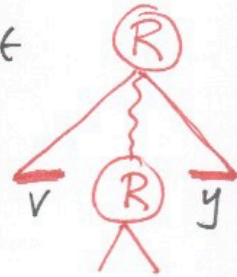
Such that all strings of the form
 UV^iXY^iZ
are also in the language.

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:

$uv^i xy^i z$ is in A , for all $i \geq 0$

CONDITION 2:

$|vy| > 0$

CONDITION 3:

$|vxy| \leq p$

LOGIC REFRESHER

HOW CAN WE NEGATE

"FOR ALL"

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

"THERE EXISTS"

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

"AND"

$$\sim (\dots p \dots \wedge \dots q \dots) \equiv (\sim (\dots p \dots) \vee \sim (\dots q \dots))$$

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

↕
"There exists a number that is not even."

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

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

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 $|vy| = 0$ OR
- ③ $|vxy| > p$

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

Assume it is CFL. Show \sim (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: $a^p b^p c^p$ $[\exists s \dots]$

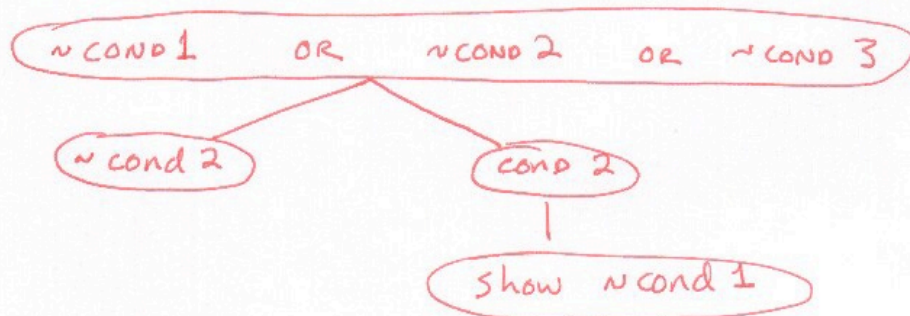
Now look at all ways to divide
it up. $[\forall uvxyz = s \dots]$

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

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

Assume this condition ~~is~~ holds.

Now look at two cases.



CASE 1

v and y each contain only one type of symbol.

$a \underline{a a a} a \quad b b b b \quad c \underline{c} c c$
 $v \qquad y$

$a a a a \quad \underline{b b b b} \quad c c c c$
 $v \qquad y = \epsilon$

One symbol will always be left out.
Pump s to uv^2xy^2z

$\underline{a a a} \underline{a a} a \quad b b b b \quad \underline{c c} \underline{c c} c$
 $v \quad v \qquad y \quad y$

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

At least one symbol will not increase in number.

The string cannot still be in the form

$$a^n b^n c^n$$

W

CASE 2

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

a a a b b b c c c

 v y

a a a b b b e c c c

 v y

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

a a a a b b b c b c c c

 v v y y

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

Assume it is a CFL.

Show \sim (PUMPING PROPERTY)

Let p be the pumping length.

[No constraints on p . Show $\forall p$]

There exists a string s , $|s| \geq p \dots$

[To show $\exists s$, just provide an
example; just give ^{an} ~~the~~ s that exists.]

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

Look at all ways to divide s into parts.

[Show $\forall uvxyz = s$]

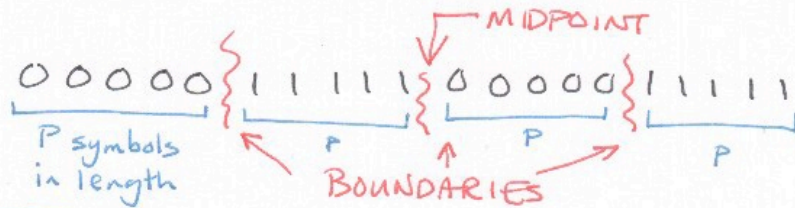
Must show that some condition
is not satisfied.

We'll assume condition 3 holds

$$|vxy| \leq p$$

and show that the other conditions
must fail.

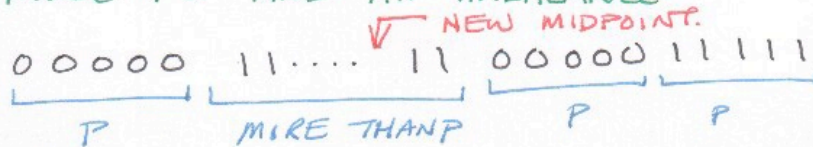
CONSIDER THE BOUNDARIES BETWEEN
0'S AND 1'S IN OUR STRING.



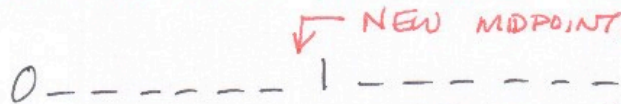
CASE 1: vxy does not straddle
a boundary.



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



THE STRING NOW HAS THE FORM



This string is not of the form w^2 .

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

CASE 2: vxy straddles the first boundary.

00000 { 11111 } 00000 { 11111 }
 ↑
 vxy MIDPOINT.

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

PUMPING DOWN WILL MAKE THE STRING SHORTER.

000 11 00000 11111
 FEWER SYMBOLS P P
 ↓
 NEW MIDPOINT

NOTE: The string now has the form:

-----0-----1
 ↓
 NEW MIDPOINT.

The string is not of the form ww .

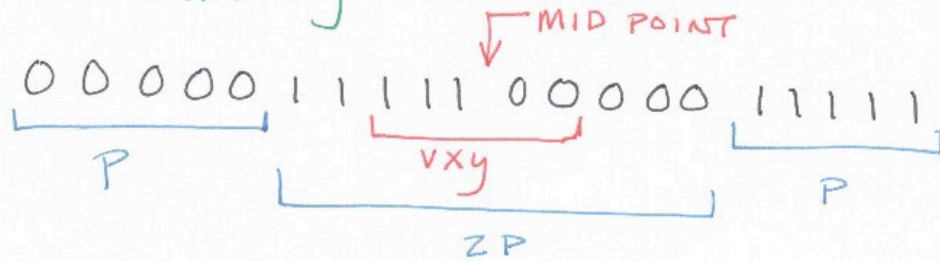
CASE 2b: vxy straddles the ~~second~~ third boundary: It is similar.

00000 { 11111 } 00000 { 11111 }

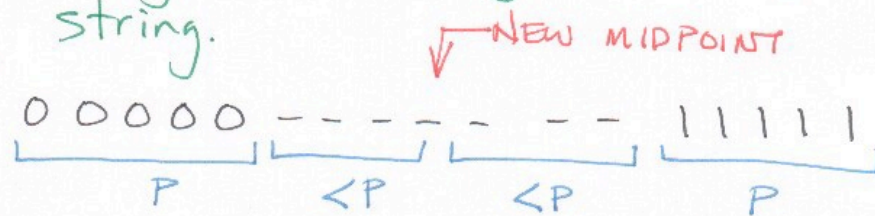
 vxy

CASE 3: vxy straddles the midpoint.

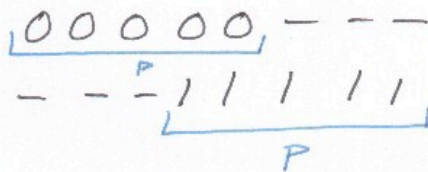
Since $|vxy| \leq p$, it cannot also straddle the first or third boundary.



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.