CSCE 355
3/20/2023

CFG $\longrightarrow$ PDA

Recall: CFG $G = \langle \underbrace{V}_{\text{grammar symbols}}, \Sigma, S, P \rangle$

One state PDA

$$P = \langle \{q\}, \Sigma, V \cup \Sigma, \delta, \{q\}, S, \emptyset \rangle$$

so that $\boxed{N(P) = L(G)}$. For every $a \in \Sigma$,

$$\delta(q, a, a) = \{(q, \varepsilon)\} \quad \text{"matching } a\text{"}$$

and for every $A \in V$,

$$\delta(q, \varepsilon, A) = \{(q, \alpha) : A \to \alpha \text{ is a production in } P\}$$

no other transitions allowed

Ex: $E \to E + T \mid T$
$T \to T * F \mid F$
$F \to c \mid \text{'('} E \text{')'}$

Input $c*(c+c)$

$(q, \text{"}c*(c+c)\text{"}, E) \vdash (q, \text{"}c*(c+c)\text{"}, T) \vdash (q, \text{"}c*(c+c)\text{"}, T*F)$

②

$\vdash (q, \text{"c*(c+c)"}, F*F) \vdash (q, \text{"\underline{c}*(c+c)"}, c*F)$

$\vdash (q, \text{"*(c+c)"}, *F) \vdash (q, (c+c), F)$

$\vdash (q, \text{"(c+c)"}, \text{'('}E\text{')'}) \vdash (q, \text{"c+c)"}, E\text{')'})$

$\vdash (q, \text{"c+c)"}, E+T\text{')'})$

$\vdash (q, \text{"c+c)"}, T+T\text{')'})$

$\vdash (q, \text{"c+c)"}, F+T\text{')'})$

$\vdash (q, \text{"c+c)"}, c+T\text{')'})$

$\vdash (q, \text{"+c)"}, +T\text{')'}) \vdash (q, \text{"c)"}, T\text{')'})$

$\vdash (q, \text{"c)"}, F\text{')'}) \vdash (q, \text{"c)"}, c\text{')'})$

$\vdash (q, \text{')'}, \text{')'}) \vdash (q, \varepsilon, \varepsilon)$ accept.

---

observe: The expansion steps (i.e. the non-matching steps) use the same sequence of productions used in a leftmost derivation of the input string.

Proof of correctness (in general, arbitrary CFG G):

Part 1: If P accepts a string $w \in \Sigma^*$ via empty stack, then there exists a leftmost derivation of w in G.

Proof: (Formally by Induction on the length of the sequence of ~~ID~~ IDs in the accepting computation of P on input w).

Sketch: somewhere in the computation:

$$\cdots \vdash ID_i \overset{\text{expansion on } A \to \alpha}{\vdash} \underbrace{ID_{i+1} \cdots \vdash}_{\text{matching}} ID_j \overset{\text{expansion}}{\vdash} \cdots \qquad (i < j)$$

$ID_i = (q, y_i, \alpha_i) \vdash (q, y_i, \alpha\beta) = ID_{i+1}$, $\quad ID_j = (q, y_j, \alpha_j)$

$\alpha_i = A\beta$ (some β)

let x_i by the input consumed already. So:

$w = x_i y_i$

Note: $x_i A\beta \Rightarrow x_i \alpha\beta$

$= x_j \alpha_j$

matching step

$(q, a y_k, \overbrace{a\gamma}^{x_k a \gamma}) \vdash (q, y_k, \gamma)$

$$(q, \underbrace{ay}, a\gamma) \vdash (q, y, \gamma)$$

~~x~~ x consumed so far

xa consumed so far

(concatenate prev consumed symbols with stack contents) does not change in a matching step.

So "nothing happens to the concatenation during matching steps, but an expansion step corresponds to a single step in a leftmost derivation.

<u>Part 2</u>: If $w \in L(G)$, then P accepts w via empty stack:

Sketch: Given $S \Rightarrow \alpha_1 \Rightarrow \cdots \Rightarrow \alpha_k = w$ leftmost derivation, then $\alpha_i \Rightarrow \alpha_{i+1}$

corresponds to an expansion step of P preceded by matching steps

Ex: $(q, y, \cdots A \cdots) \xrightarrow[\text{matching}]{} (q, y', A \cdots)$

$\vdash_{\text{expand}} (q, y', \alpha \ldots)$ ($A \to \alpha$ production)

Next up: PDA $\Longrightarrow$ CFG

Convert an arbitrary PDA P into a CFG G such that $L(G) = N(P)$.

Plan:
PDA $\Longrightarrow$ ~~resp~~ restricted PDA $\Longrightarrow$ CFG

Def: A PDA is restricted if the only allowed transitions are of the following 2 forms:

$\delta(q, a, X)$ contains (r, ε)

for $q, r \in Q$

$a \in \Sigma \cup \{\varepsilon\}$

$X \in \Gamma$

"pop X" or "pop"

or

$\delta(q, a, \underline{X})$ contains $(r, \underline{YX})$

q, r, a as above, $X, Y \in \Gamma$

"push Y"

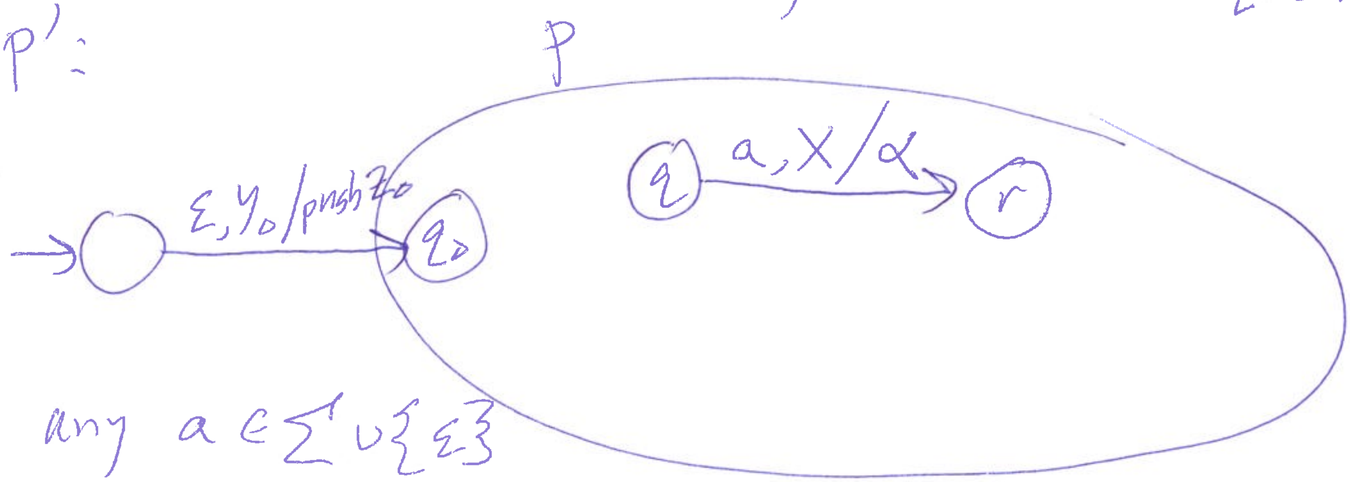
<u>Lemma</u>: For every PDA P there exists a restricted PDA P' such that $N(P') = N(P)$.

<u>Proof</u>: By construction. Let

$P = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, - \rangle$

$P' = \langle Q \cup (\text{more states}), \Sigma, \Gamma \cup \{Y_0\}, \delta', q_0', Y_0, \emptyset \rangle$

P':



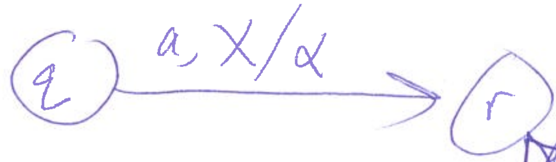
any $a \in \Sigma \cup \{\varepsilon\}$

$X \in \Gamma$

$\alpha \in \Gamma^*$

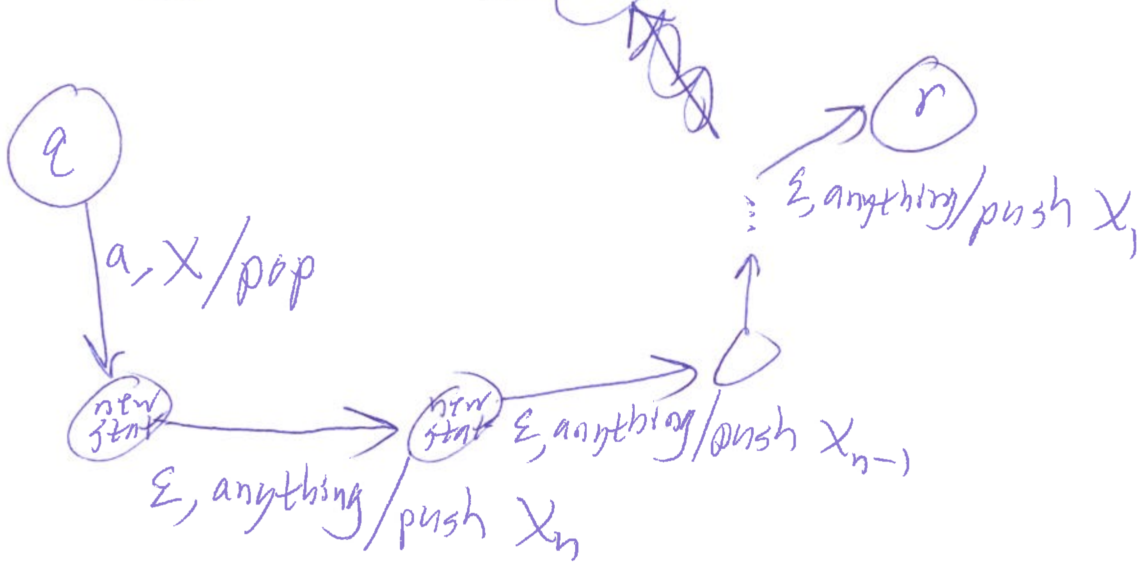
Let $\alpha = X_1 X_2 \cdots X_n$ for some $n \geq 0$

replace



in P

with



Finally, for every state q of P, add a transition

$N(P') = N(P)$ by construction.

What remains: convert a restricted PDA into an equivalent CFG.

CSCE 355
3/27/2023

Today: PDA $\Rightarrow$ CFG
$P \mapsto G$

1

Last time: PDA $\Rightarrow$ restricted PDA

A restricted PDA is a PDA that allows only two types of actions

~~$\delta(q, a)$ contains~~

$\delta(q, a, X)$ contains $\underbrace{(r, \varepsilon)}_{(r,\ \text{pop})}$

or

$\delta(q, a, X)$ contains $\underbrace{(r, YX)}_{(r,\ \text{push } Y)}$

Today: restricted PDA $\Rightarrow$ CFG

Given a restricted PDA $P = \langle Q, \Sigma, \Gamma, \delta, q_0, z_0 \rangle$
Construct a CFG G such that $L(G) = N(P)$.

$G = \langle V, \Sigma, S, P \rangle$

V contains symbols of the form,

$\underline{[qXr]}$ for all states $q, r \in Q$

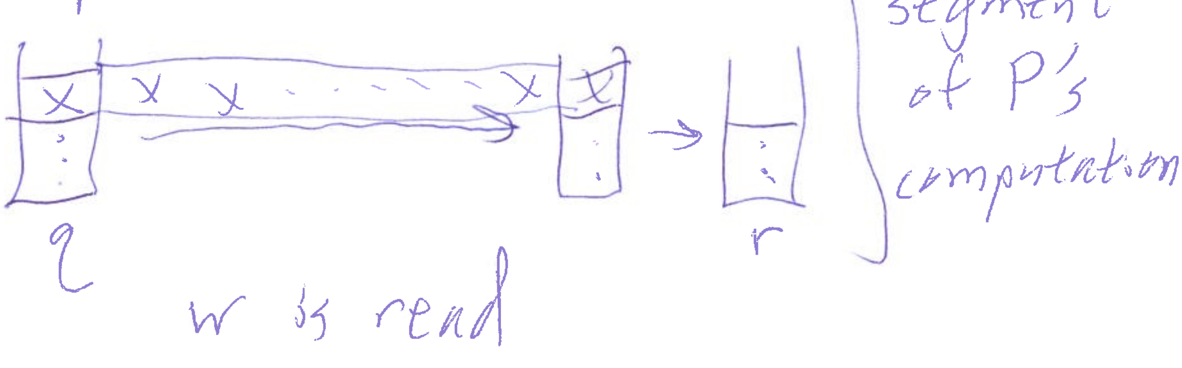
and ~~stk~~ stack symbols $X \in \Gamma$,

as well as S.

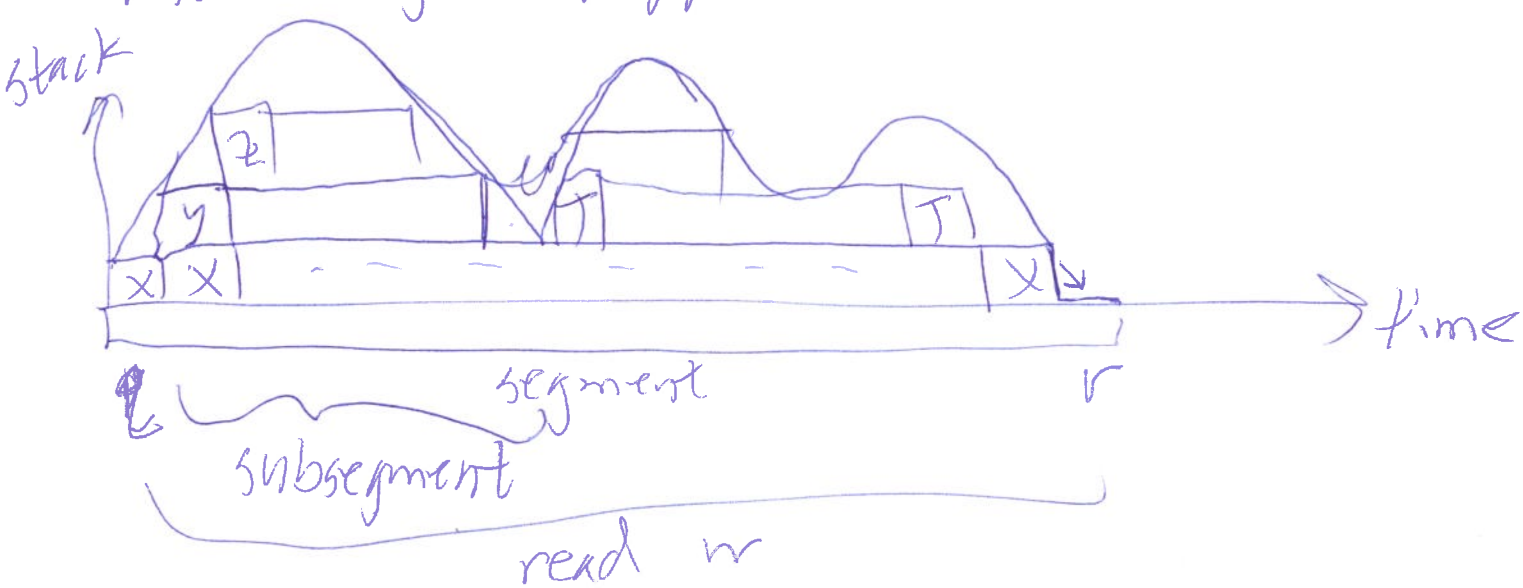
Want productions so that

$$[qXr] \Rightarrow^* w \qquad (w \in \Sigma^*)$$

iff P, starting with X on top of stack, and in state q can reach state r and X stays on top of the stack until ~~then~~ it gets popped, and w is read from the input.



A segment of P's comp starts with some X on top of the stack and in some state q and ends in some state r, where X stays on the stack until the last step, when it gets popped



want $[qXr] \Rightarrow^* w$

whole computation (assuming w is accepted)

S-productions (S is the start symbol) 4

For every state $r \in Q$, add the production

$$S \to [q_0 Z_0 r]$$

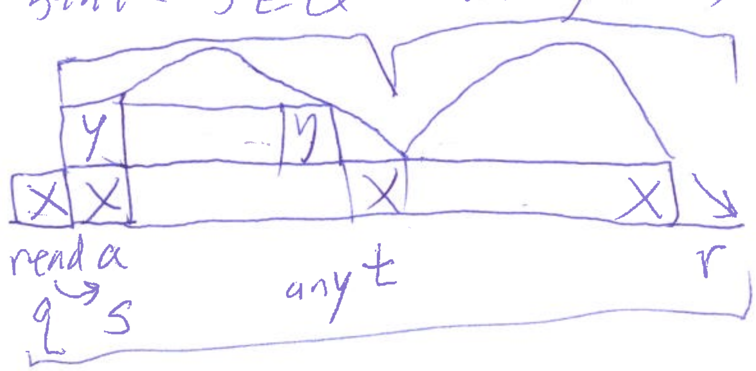
Popping productions: For any $a \in \Sigma \cup \{\varepsilon\}$ and state $q \in Q$ and stack symbol $X \in \Gamma$, if $\delta(q, a, X)$ contains (r, pop) (for any $r \in Q$), add the production

$$[qXr] \to a \qquad \text{(shortest segment)}$$

Pushing productions: suppose

$$\delta(q, a, X) \text{ contains } (s, \text{push } y)$$

(some state $s \in Q$ and $y \in \Gamma$)



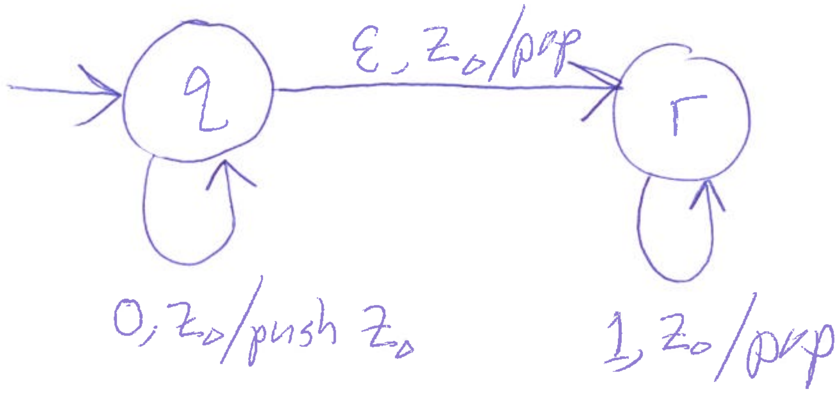
Then for every state $t \in Q$ add the production

$$[qXr] \to a[sYt][tXr]$$

No other productions.

~~End~~ Completes the construction of G.

Ex: $L = \{0^n 1^n : n \geq 0\}$ $\Gamma = \{Z_0\}$



G:

$$V = \{S, [qZ_0q], [qZ_0r], [rZ_0q], [rZ_0r]\}$$

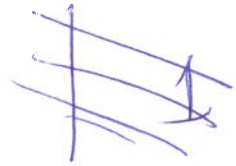
S-productions

$$S \to [qZ_0q] \mid [qZ_0r]$$

Popping productions:

$[q Z_0 r] \to \varepsilon$

$[r Z_0 r] \to 1$



Push productions

$$[q Z_0 q] \to 0[q Z_0 q]^{t=q}[q Z_0 q]$$

$$\mid 0[q Z_0 r][r Z_0 q]$$

$A := [q Z_0 q]$

$B := [q Z_0 r]$

$C := [r Z_0 q]$

$D := [r Z_0 r]$

$S \to A \mid B$

$B \to \varepsilon \mid 0AB \mid 0BD$

$A \to 0AA \mid 0BC$

$D \to 1$ bypass

$$[q Z_0 r] \to 0[q Z_0 q][q Z_0 r]$$

$$\mid 0[q Z_0 r][r Z_0 r]$$

$S \to A \mid B$

$B \to \varepsilon \mid 0AB \mid 0B1$

$A \to 0AA \mid \underbrace{0BC}_{\text{useless}}$

$\Downarrow$

$S \to \overbrace{A}^{\text{useless}} \mid B$

$B \to \varepsilon \mid \overbrace{0AB}^{\text{useless}} \mid 0B1$

$A \to 0AA$] useless

$\Downarrow$

$S \to B$

$B \to \varepsilon \mid 0B1$

$\Downarrow$ make B the start symbol:

$$\boxed{B \to 0B1 \mid \varepsilon}$$

Proof of correctness omitted.

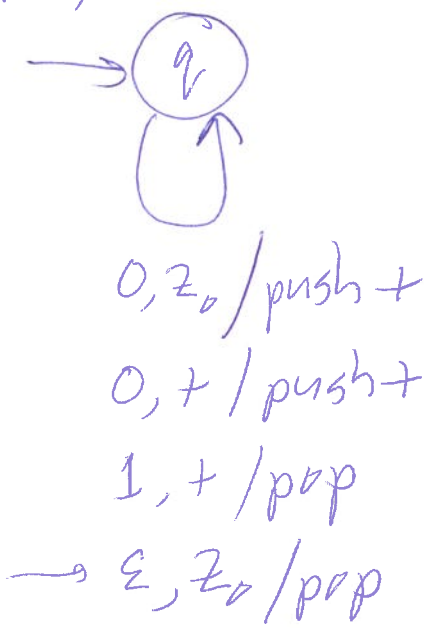
Proof idea: $N(P) \subseteq L(G)$ by induction on the length ~~of a derivat~~ of an accepting path of P

$L(G) \subseteq N(P)$ by induction on the length of a derivation of G

Ex: Properly nested parentheses

$0 = ($

$1 =)$



$S \to [q Z_0 q]$

$[q Z_0 q] \to \varepsilon$

$[q + q] \to 1$

$[q Z_0 q] \to 0 [q + q][q Z_0 q]$

$[q + q] \to 0 [q + q][q + q]$

$A := [q Z_0 q]$

$B := [q + q]$

$\cancel{S \to A}$

$A \to \varepsilon \mid 0 B A$

$B \to 1 \mid 0 B B$

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

is not CFL-pumpable (hence not a CFL by the Lemma)

What "not CFL-pumpable" means:

$\forall p > 0$

$\quad \exists s \in L, |s| \geq p$

$\quad\quad \forall u, v, \cancel{x, y, z}\ w, x, y$ such that

$s = uvwxy$

$\rightarrow |vwx| \leq p$ ☆

$\rightarrow |vx| > 0$

$\exists i,\ uv^i w x^i y \notin L.$

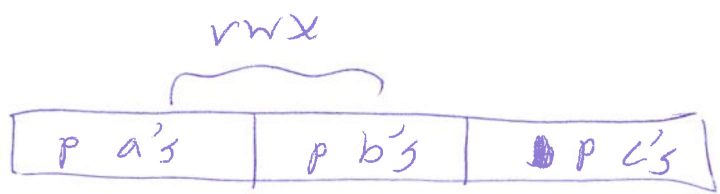
L is not CFL-pumpable:

Given $p > 0$, $s = a^p b^p c^p$. ($s \in L$ & $|s| = 3p \geq p$)

Given u, v, w, x, y as above,

let $i := 0$.

This works. Why?



Since $|vwx| \leq p$, vwx cannot contain both some a's and some c's. So

$uwy = uv^0 w x^0 y$ has either the same

CSCE 355
3/29/2023

Prog project handout:
1) ε-NFA $\Rightarrow$ NFA
2) simulate an NFA on input strings

Pumping Lemma for CFLs

~~Use~~ Used to show a lang is not a CFL.

Lemma (Pumping Lemma for CFLs): Every context-free language is CFL-pumpable.

Def: A lang. L is CFL-pumpable if

$\exists p > 0$, ("pumping length")

$\forall s \in L$ such that $|s| \geq p$,

$\exists u, v, w, x, y$ strings such that

1) $s = uvwxy$
2) $|vwx| \leq p$
3) $|v| + |x| > 0$ (i.e., v, x cannot both be ε)

and $\forall i \geq 0$, $uv^iwx^iy \in L$.

Proof (later).

number of c & fewer a's or b's, ③

or the same # of a's and fewer b's or fewer c's (or both)

$\therefore uwy \notin L.$ //

Similarly:

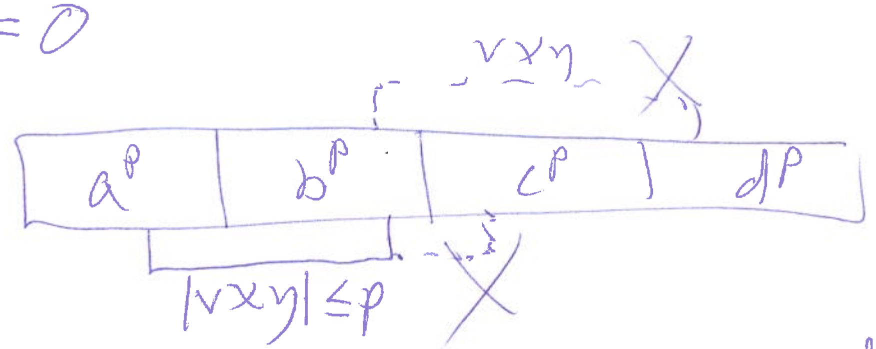
$$L := \{a^i b^j c^i d^j : i, j \geq 0\}$$

is not CFL-pumpable.

$\forall p > 0$, let $s := a^p b^p c^p d^p$

Given u, v, x, y, z s.t. $s = uvxyz$, $|vxy| \leq p$, $|vy| > 0$,

Let $i := 0$



vxy ~~can't~~ can't have both a's & c's, and can't have both b's & d's. (too far apart).

So any $i \neq 1$ will work: adjust # of one of a, c's with out the other, or one of b's & d's & not the other, leaving $\neq$ numbers of a's & c's or b's and d's.

So $uv^ixy^iz \notin L$ for any $i \neq 1$.

```
int f(~~a, b, c~~ x, y, z) {      // x, y, z: 3
  ...
}
int g(a, b) {                     // a, b: 2
  ...
}
  ...
  f(2, 3, 4)                      // 3
  ...
  g(5, 6)                         // 2
```

OK b/c # actual args for f, g matches the number of formal params for each function

A CFG alone cannot check this — not a context-free part of the language (C++)

Need the symbol table — a global data struct that remembers this info (# formal params)

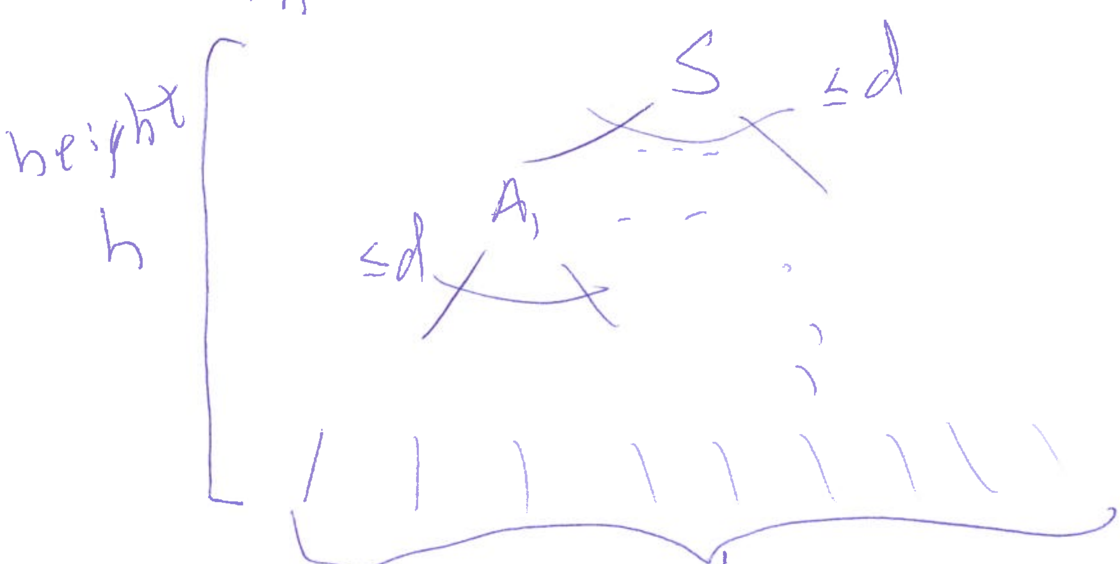
Proof (of the Pumping Lemma for CFLs)

[Every CFL is CFL-pumpable]

Let L be any CFL. Fix a CFG G such that $L = L(G)$.

~~Let G~~ Let n be the number of noterminals of G, and let d be the max length of any body of a production of G.

Any ~~complete~~ parse tree of G has branching $\le d$

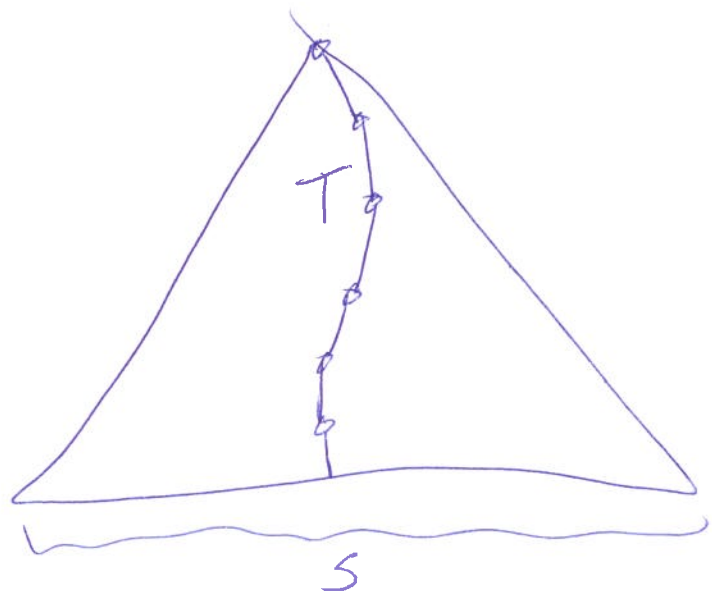


$\le d^h$ leaves

Let $p := d^{n+1}$

Let $s \in L$ be any string of length $\ge p$.

Since $s \in L$, there is a parse tree ⑥
yielding s. Let T be a min-size (min # of nodes) parse tree of G yielding s.



$|s| \geq p = d^{n+1}$. What is the height of T?

Letting h be the height of T,

T has $\leq d^h$ many leaves, so

$\# \text{leaves} = |s| \geq d^{n+1}$, so

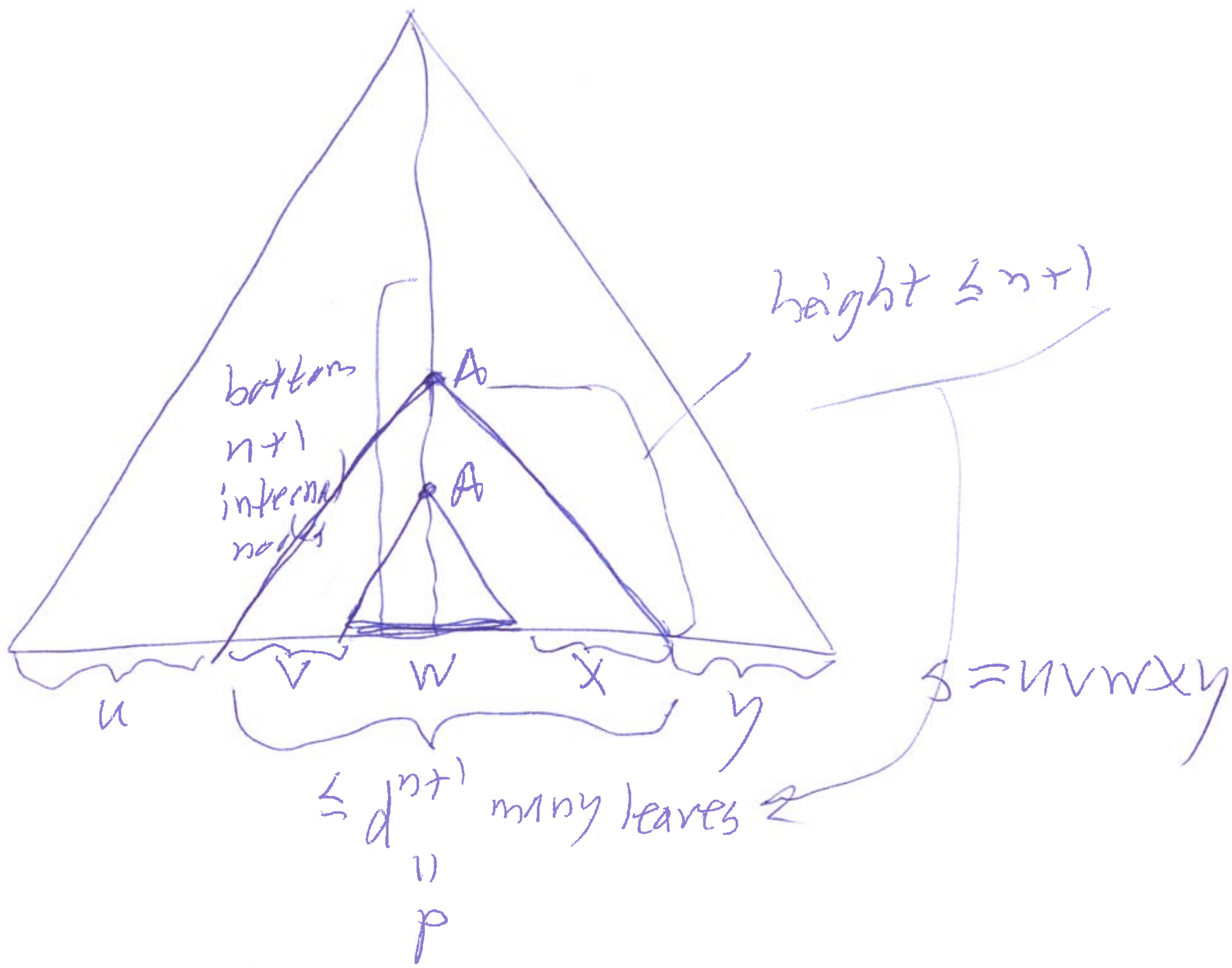
$d^h \geq d^{n+1}$

$\therefore h \geq n+1$

So T has height $\geq n+1$

$\therefore$ there is a path in T with $\geq n+1$ many internal nodes, each labeled by a nonterminal. But only n nonterminals

∴ (pigeonhole principle) there is some nonterminal (A, say) that is repeated (occurs $\geq$ twice) among the bottom $n+1$ internal nodes along the path.



$\therefore |vwx| \leq p$

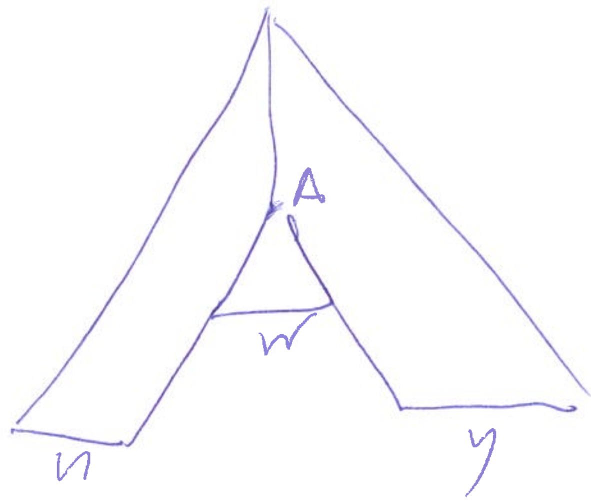
What's left: show that $|vx| > 0$

and $uv^iwx^iy \in L$ for all $i \geq 0$.

Show the 2nd one first.

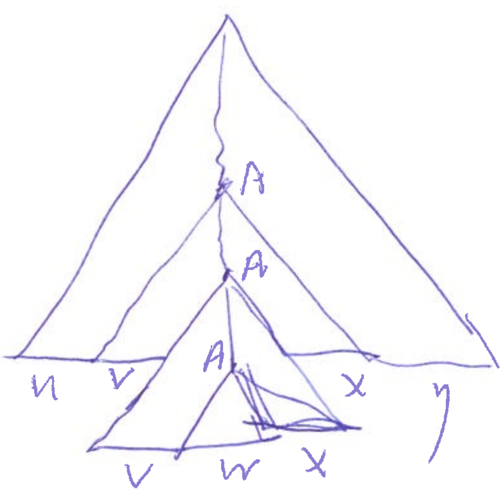
$i := 0$: Parse tree for $uv^0wx^0y = uwy$: delete the "wedge" and merge the lower A with the upper A:

(8)



$i := 2$. Duplicate the wedge:

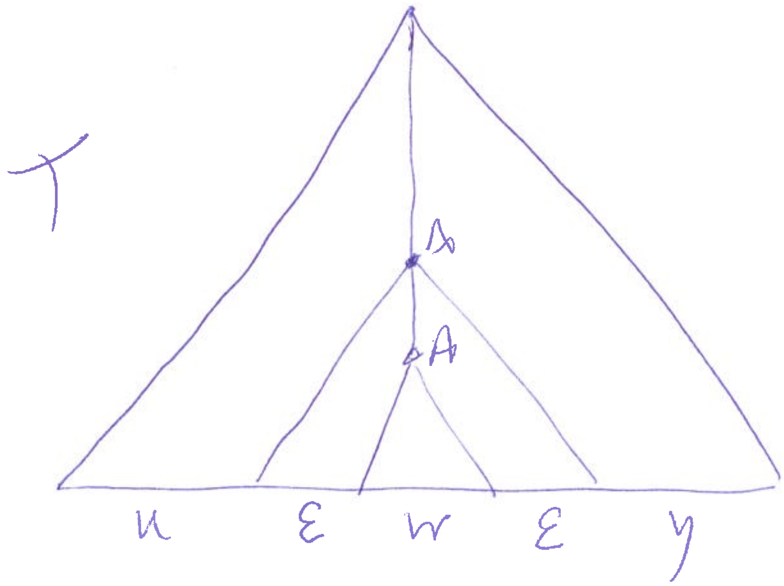
parse tree for uv^2wx^2y



For any i, have i copies of the wedge, one on top of the other to get a parse tree for uv^iwx^iy

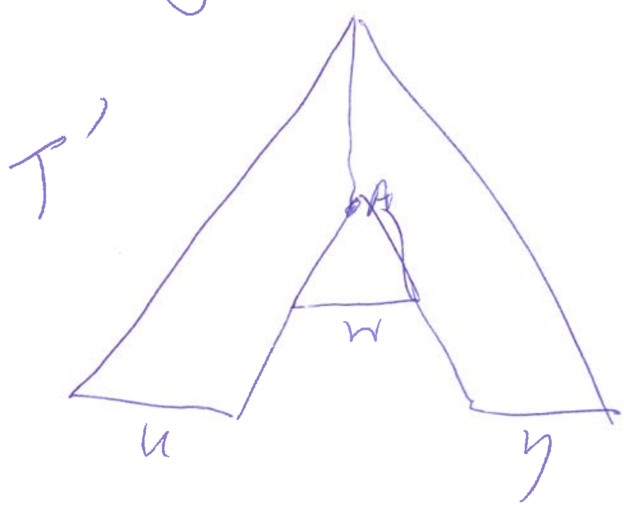
$\therefore uv^iwx^iy \in L$ for all i.

Last thing: show that v and x can't both be ε. Suppose otherwise.



$s = u\varepsilon w\varepsilon y$
$= uwy$

Remove the wedge & merge the two A's: get another parse tree T' also yielding s:



but T' is smaller than T. But we picked T to be min size yielding s. $\Rightarrow$ so $|vx| > 0$.

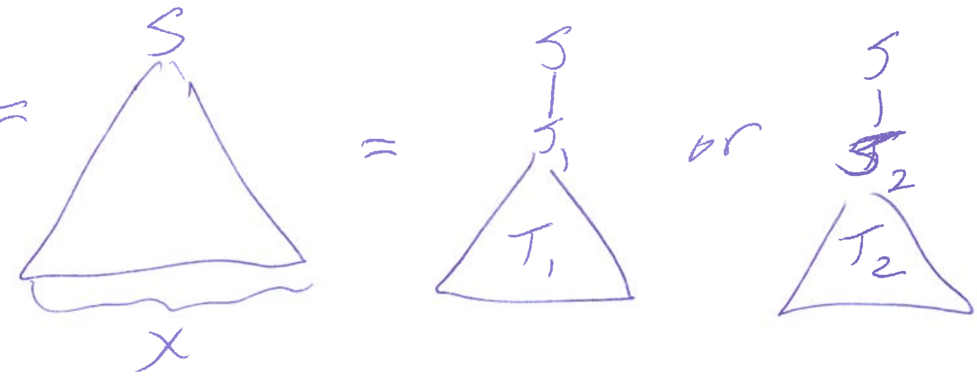
Therefore, $w \in L(G)$.
Similarly, any string in L_2 is also in G.

Thus $L_1 \cup L_2 \subseteq L(G)$

want $=$

Let x be any string in $L(G)$, and let T be a complete parse tree of G yielding x.

Then $T =$



Then T_1 (or T_2, whichever) is a complete parse tree of either G_1 or G_2 yielding x.

$\therefore x \in L(G_1)$ or $x \in L(G_2)$

$\therefore L(G) \subseteq L_1 \cup L_2$

$\therefore L(G) = L_1 \cup L_2$ ∎

<u>Prop</u>: The concatenation of two CFLs is a CFL.

<u>Proof</u>: Given $L_1 = L(G_1)$, $L_2 = L(G_2)$ as in the previous proof. It suffices to find a grammar G such that $L(G) = L_1 L_2$.

CSCE 355
4/3/23

Closure Properties of CFLs

Closure under union, concat, *

<u>Prop:</u> The union of any two CFLs is a CFL.

<u>Proof:</u> Let $L_1 = L(G_1)$ and $L_2 = L(G_2)$ for CFGs $G_1 = \langle V_1, \Sigma, S_1, P_1 \rangle$

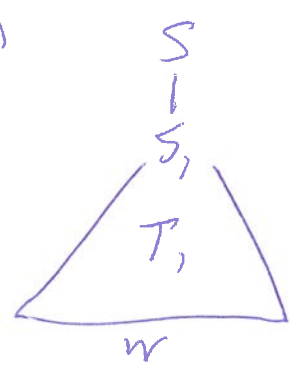
and $G_2 = \langle V_2, \Sigma, S_2, P_2 \rangle$

WLOG, $V_1 \cap V_2 = \emptyset$ (by renaming nonterminals).

Let $G := \langle V_1 \cup V_2 \cup \underbrace{\{S\}}_{\substack{\text{new} \\ \text{symbol} \\ \text{not in} \\ V_1 \cup V_2}}, \Sigma, S, P \rangle$

where $P = P_1 \cup P_2 \cup \{\underbrace{S \to S_1}, S \to S_2\}$

Given $w \in L_1$, let T_1 be a complete parse tree of G_1 yielding w. Then

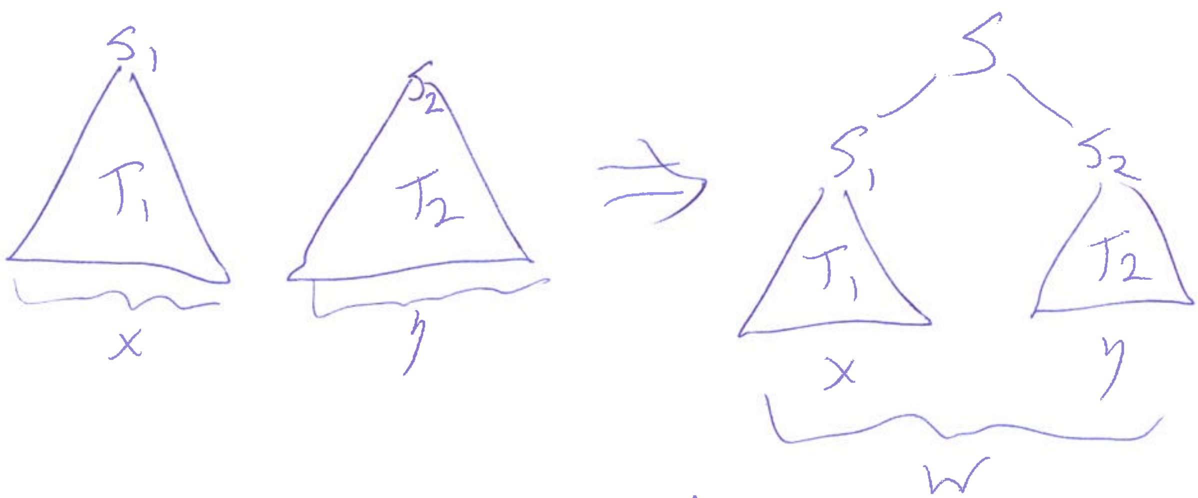


is a complete parse tree in G yielding w.

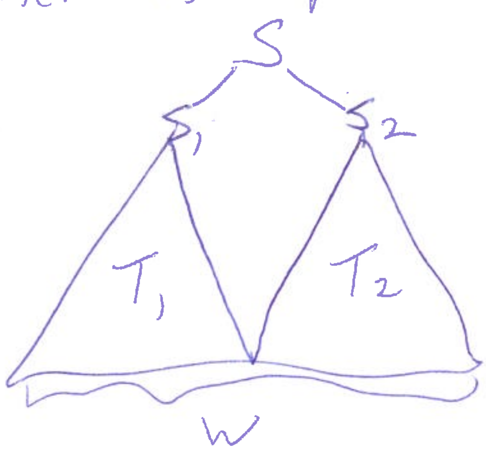
$G := \langle V_1 \cup V_2 \cup \{S\}, \Sigma, S, P \rangle$

where $P = P_1 \cup P_2 \cup \{S \to S_1 S_2\}$.

Let w be any string. If $w = xy$ for $x \in L_1$, $y \in L_2$, then there are parsetrees



so $\exists$ parsetree of G yielding w $\therefore w \in L(G)$ $\therefore L_1 L_2 \subseteq L(G)$. Conversely, if $w \in L(G)$, then there is a parse tree of G yielding w



T_1 is a parse tree of G_1 yielding some string x; T_2 is a parse tree of G_2 yielding some string y, and $w = xy$.

$\therefore w \in L_1 L_2$ $\quad \therefore L(G) \subseteq L_1 L_2$ $\quad \therefore L_1 L_2 = L(G)$. //

Kleene closure ($*$-operator)

Prop: If L is a CFL then L^* is a CFL.

[recall: $L^* = \{\varepsilon\} \cup L \cup LL \cup LLL \cup \cdots$]

Let $L = L(G_1)$ where

$G_1 = \langle V_1, \Sigma, S_1, P_1 \rangle$ is a CFG.

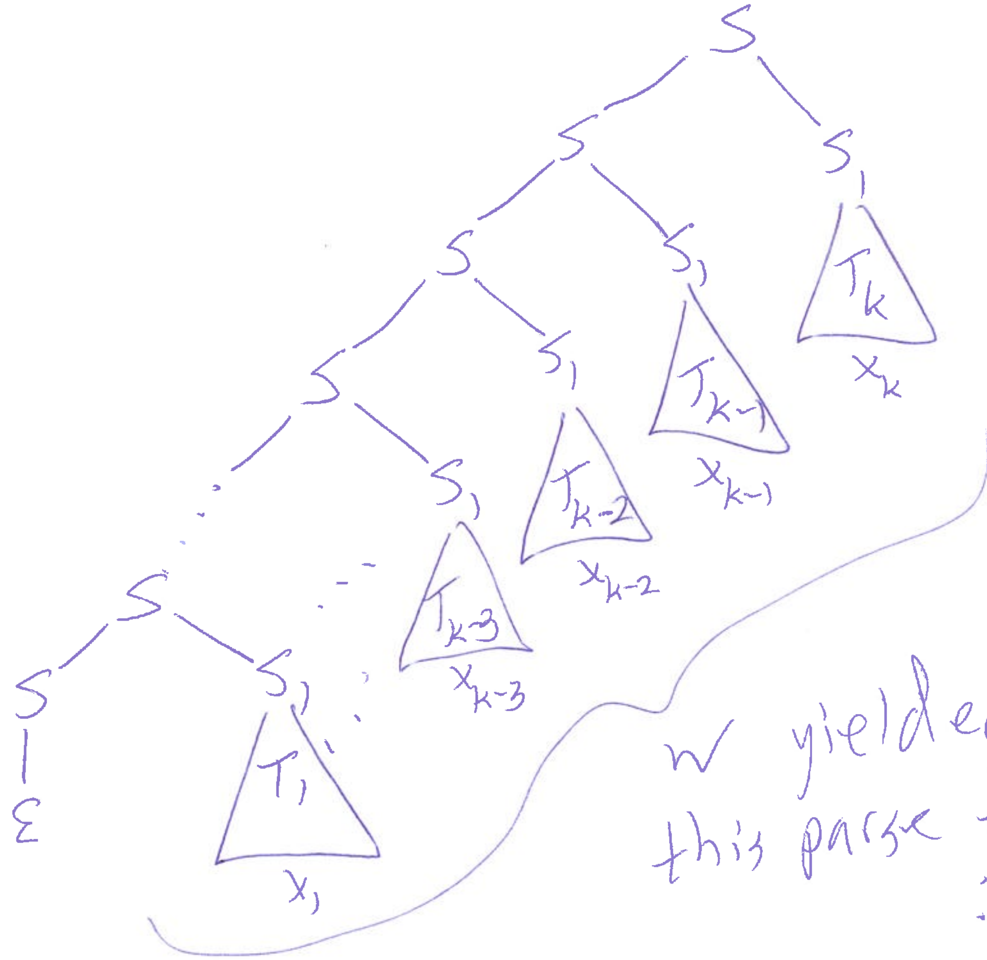
Define $G := \langle V_1 \cup \{S\}, \Sigma, S,$

$P := P_1 \cup \{S \to SS_1, S \to \varepsilon\} \rangle$

Let $w \in L^*$. Show that $w \in L(G)$.

Suppose $w = x_1 x_2 \cdots x_k$ for some $k \geq 0$ and each $x_i \in L$

For $1 \leq i \leq k$, let T_i be a complete parse tree of G_1 yielding x_i. Then the following complete parse tree of G yields w:



w yielded by this parse tree of G.
$\therefore\ w \in L(G)$.

Conversely, if $w \in L(G)$, then any parse tree ~~of~~ of G yielding w must look like the one drawn above, for some k, where the T_i are all parse trees of G_1. $\therefore\ w = x_1 \cdots x_k$, where x_i ~~???~~ is the yield of T_i, and so is in L.

$\therefore\ w \in L^k \subseteq L^*$

$\therefore\ w \in L^* \iff w \in L(G)$ (any string $w \in \Sigma^*$)

$\therefore\ L(G) = L^*$ //

<u>Cor</u>: Every regular language is context-free.

Pf: Only thing left is to find grammars for the atomic regexes, corresp to languages $\emptyset$ and $\{a\}$ for all $a \in \Sigma$.

$$L(G) = \emptyset \quad \text{for} \quad G := \langle \{S\}, \Sigma, S, \emptyset \rangle$$

or $\{S \to S\}$

$\forall a \in \Sigma$, $L(G_a) = \{a\}$ for $G_a := \langle \{S\}, \Sigma, S, \{S \to a\} \rangle$ //

Alternate proof of the corollary:

Any DFA can be simulate by a PDA ($\therefore$ equivalent) that ignores its stack. //

Prop: If L is a CFL, then L^R is a CFL.

[recall: $L^R := \{w^R : w \in L\}$] $G = \langle V, \Sigma, S, P \rangle$

Proof: Let G be a CFG s.t. $L = L(G)$.

Want a CFG for L^R:

$L^R = L(G^R)$, where $G^R := \langle V, \Sigma, S, \{A \to \alpha^R : A \to \alpha \in P\} \rangle$ //

[proof of correctness omitted]

Prop: There exist CFLs L_1 and L_2 such that $L_1 \cap L_2$ is not context-free.

Proof: Recall that we showed that the language

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

is not a CFL (by the pumping lemma for CFLs). But $L = L_1 \cap L_2$ for CFLs L_1, L_2 where

$$L_1 := \{a^i b^i c^k : i, k \geq 0\}$$

$$L_2 := \{a^k b^i c^i : i, k \geq 0\}$$

$L_1 = L(G_1)$, $L_2 = L(G_2)$, where

G_1: $S \to Sc \mid T$
$T \to aTb \mid \varepsilon$

G_2: $S \to aS \mid T$
$T \to bTc \mid \varepsilon$

//

Corollary: There exists a CFL L such that $\overline{L}$ is not a CFL.

Proof: Suppose CFLs are closed under complements. Know (proved) that CFLs under union. But by De Morgan's laws, for any L_1 and L_2

$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$$

~~The CFLs are~~ The right-hand side is a CFL by assumption, for any CFLs L_1, L_2

$\therefore$ $L_1 \cap L_2$ is a CFL for any CFLs L_1 & L_2, but we just saw a counterexample to this $\lightning$ $\therefore$ CFLs not closed under complement.

Ex: Find an explicit CFL L such that $\overline{L}$ is not a CFL.

Ex: Show that $\{ww : w \in \Sigma^*\}$ is not pumpable $\therefore$ not a CFL, for $|\Sigma| \geq 2$.

Ex: Show that $\{x : x$ is not of the form $ww\}$ is a CFL.

CSCE 355
4/5/2023

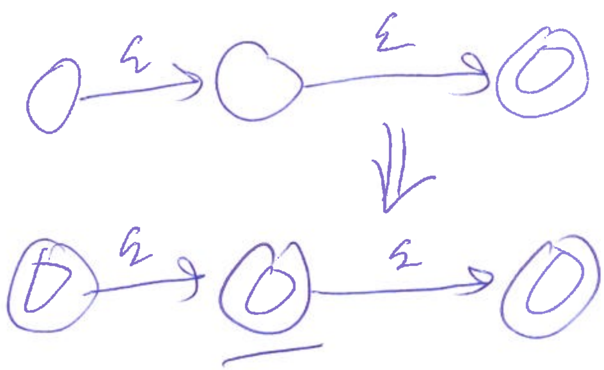
Programming Project

1

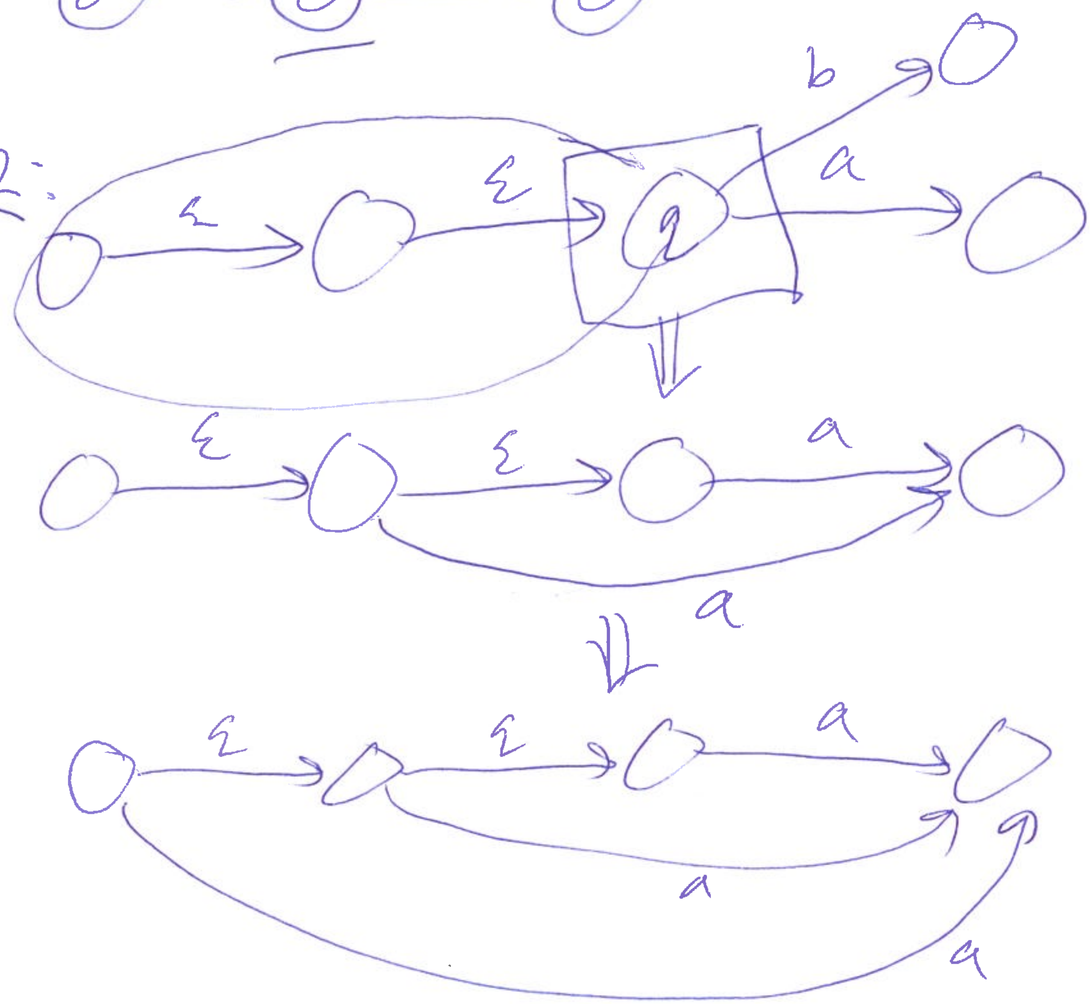
1. Script grades your project on a linux machine.

2. Do your own work.

3. Step 1:



Step 2:



Useful precomputation step:
find the reversals of all the
ε-moves (back edges)

In steps 1 & 2: given a state q, search
for all states reachable from q
following back edges only (reverse ε-moves).
Can use BFS or DFS for this.

Pass: do this starting at each state
in sequence, for all states.

Repeat passes until nothing changes during
a complete pass.

Turing Machines (TMs)

"Turing machine model captures the
(informal) notion of computation"

Church-Turing thesis

Next few lectures will convince you of this.

Def. A Turing machine (TM) is a 7-tuple

$\langle Q, \Sigma, \Gamma, \delta, q_0, B, F \rangle$ where

Q is a finite set (elements are __states__)

Σ, Γ are alphabets:

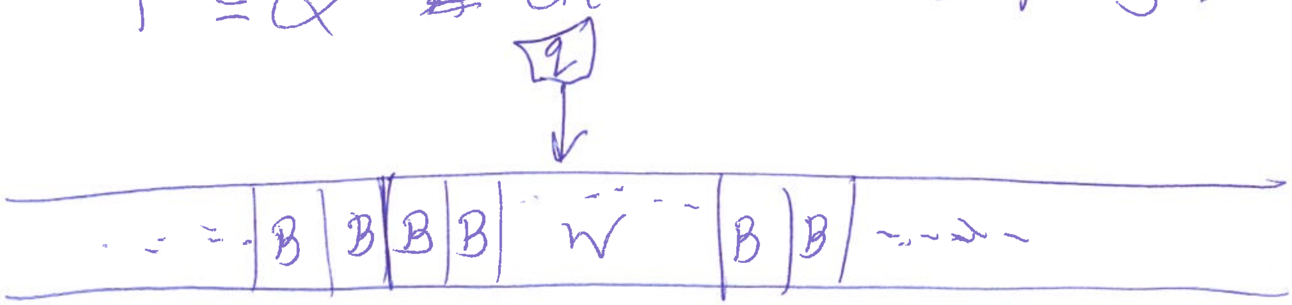
- Σ is the __input alphabet__
- Γ is the __tape alphabet__

$\Sigma \subseteq \Gamma$

$q_0 \in Q$ is the __start state__

$B \in \Gamma \setminus \Sigma$ is the __blank symbol__

$F \subseteq Q$ elements are __accepting states__



New things a TM can do:

head can move in both directions
(but only one cell at a time)
and can move off the input.

cell contents can be altered by the TM.

In one time step: TM can
- change state
- overwrite the scanned cell
- move head one cell left or right.

Finally $\delta : Q \times \Gamma \to Q \times \Gamma \times \{\leftarrow, \rightarrow\}$ is a partial function (not necessarily ~~well~~ defined for all combos of state, tape symbol.

Informally: $q \in Q$, $a \in \Gamma$

$$\delta(q, a) = (r, b, d)$$

$r \in Q$
$b \in \Gamma$
d is either $\leftarrow$ or $\rightarrow$

means: If current state is q and a is in the currently scanned cell, the in the next time step, the state becomes r, the scanned a is changed to b, and then the head moves one cell to the left or right, depending on d.

Initially, given an input string $w \in \Sigma^*$, the tape has w in contiguous cells, ~~starts~~ with the rest of the tape blank (cells contain B). The head scans the leftmost symbol of w, if there is one.

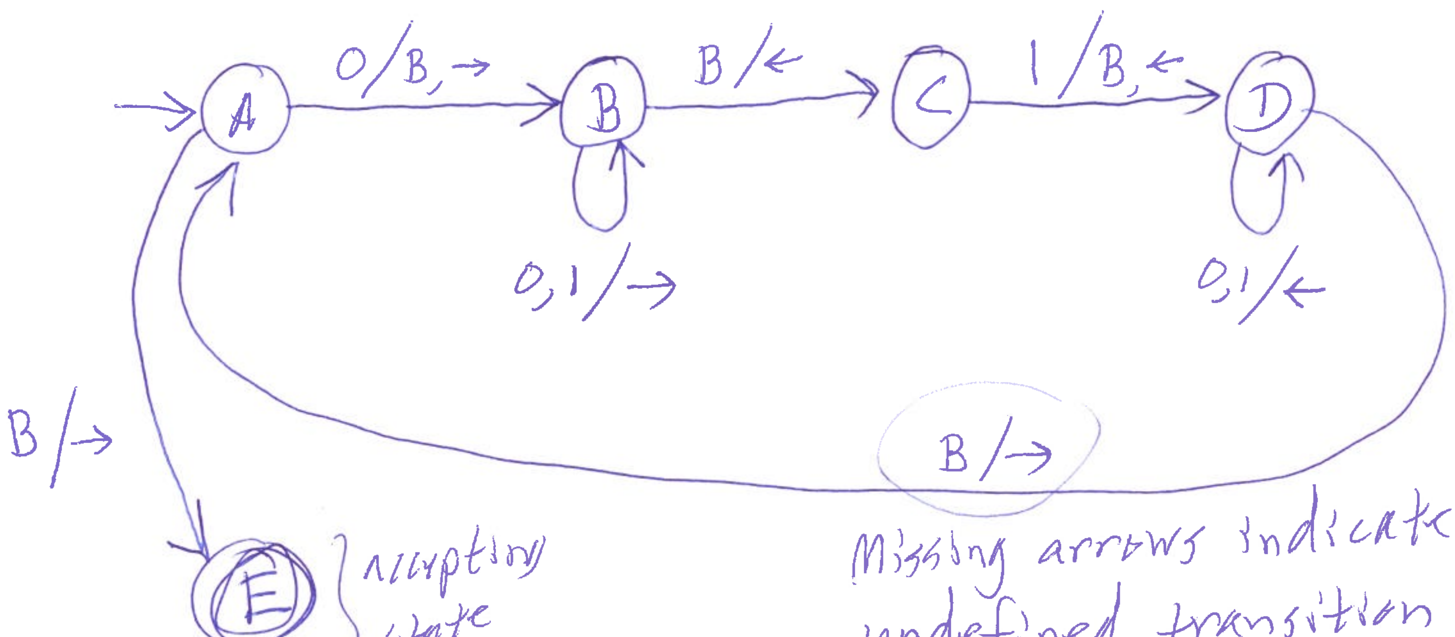
At any time, if the state is q and scanned symbol is a such that $\delta(q, a)$ is undefined then the computation halts (does not proceed).

~~Ma~~ The TM accepts iff it halts in an accepting state.

The TM rejects iff it halts in a rejecting state (not in F)

Third possibility: the computation goes on forever without halting. (It "loops")

Ex TM as a transition diagram:



} accepting state

Missing arrows indicate undefined transition (so halt)

6

Input: 00001111

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

$\Gamma = \{0, 1, B\}$