CSCE 355
2/22/2023

Pumping Lemma for reg langs ①

<u>Def</u>: $L \subseteq \Sigma^*$ is <u>pumpable</u> if

$\exists p > 0$ (the pumping length)

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

$\exists x, y, z \in \Sigma^*$ (such that)

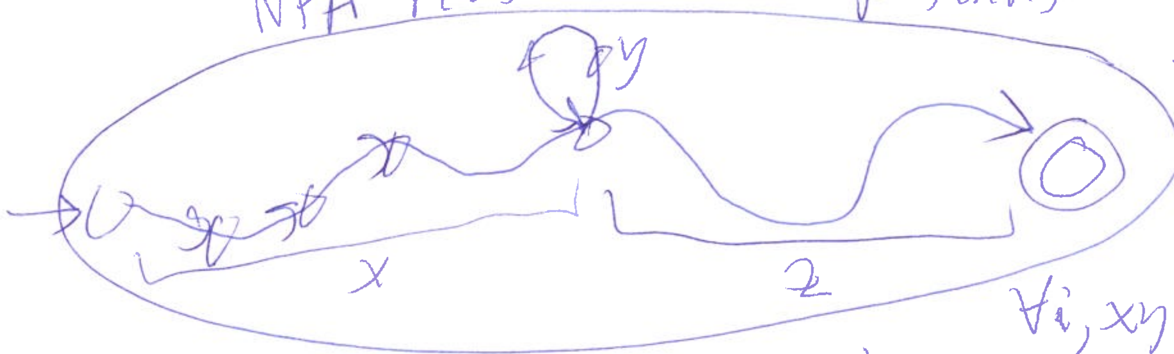
- $s = xyz$
- $|xy| \leq p$
- $|y| > 0$ and

$\forall i \geq 0 \quad xy^iz \in L$.

<u>Pumping Lemma</u>: If L is regular then L is pumpable

Proof idea:

NFA recog L with p states



$xyz = s$
$|xy| \leq p$
$|y| > 0$
$\forall i, xy^iz \in L$

$s \in L$

L is not pumpable iff

$\to \forall p > 0$

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

$\to \forall x, y, z$ such that

- $s = xyz$
- $|xy| \leq p$
- $|y| > 0$

$\to \exists i \geq 0, xy^i z \notin L.$

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

Prop: L is not pumpable.

Proof: Given $p > 0$, let $s := 0^p 1^p$

$[s \in L \ \& \ |s| = 2p \geq p]$

Given x, y, z such that $s = xyz$, $|xy| \leq p$, $|y| > 0$,

[Since $|xy| \leq p$, $|y| > 0$, $y = 0^k$ for some $k > 0$]

[For any i, $xy^i z = 0^{p+(i-1)k} 1^p$]

[Suffices to choose i such that $p+(i-1)k \neq p$.

Set $i := 0$. Then $xy^0z = xz = 0^{p-k}1^p \notin L$

because $p-k < p$, so fewer 0's than 1's. //

$\therefore$ L not pumpable

Cor: L is not regular.

Pf: By the pumping lemma. //

Ex: $L := \{ w \in \{0,1\}^* : w$ has the same # of 0's as 1's$\}$

Prop: L not pumpable, hence not regular.

Proof: Identical to the previous proof (word for word). //

Ex: $L = \{ 0^m 1^n : m,n \geq 0$ and $m \geq n \}$

Prop: L not pumpable.

Proof is identical to the prev proof. Review

Given $p > 0$, chose $s := 0^p 1^p$ (still in L)

Given x, y, z, ...

Had $\boxed{i := 0}$ — still works, because

only value of i that works

$\underbrace{p-k}_{\# 0\text{'s in } xz} < p$ (just $\neq$ not good enough)

Note: There are langs that are pumpable but not regular, so converse of the pumping lemma does not hold.

Ex: $L = \{ 0^m 1^n : 0 \le m \le n \}$

Prop: L not pumpable

Pf: Given $p > 0$, let $s := 0^p 1^p$.

Given x, y, z s.t. ... — know that $y = 0^k$ for some $k > 0$

Let $i := 2$. Then

$xy^2z = 0^{p+k} 1^p \notin L$ ~~pe~~ because $p + k > p$

Proof template for nonpumpability of L:

"Given $p > 0$, let $s :=$ ________

[then $s \in L$ and $|s| \ge p$]

Given x, y, z such that $s = xyz$, $|xy| \le p$, $|y| > 0$,

let $i :=$ ______. Then $xy^i z \notin L$

because ____________."

Mistake 1: Given $p>0$, let $s := \underbrace{000111}_{\text{fails if } p \geq 7}$.

Mistake 2: Let $p := 6$. Let $s := 000111$.

Can't choose p. p is given to you.

Mistake 3: Given $p>0$, let $s := 0^p 1^p$

then let $x :=$ ___ , $y :=$ ___ , $z :=$ ___

So $s = xyz$, $|xy| \leq p$, $|y| > 0$ ....

Can't choose specific x, y, z. Can only assume x, y, z satisfy the three conditions.

Mistake 4: .... let $i := 1$.

Then $xy^1z = xyz = s \in L$.

So $i := 1$ never works.

Ex $L := \{ w \in \{a, b, c\}^* :$ $|w|$ is even and w has a "c" somewhere in its first half $\}$

Prop: L not pumpable

Pf: Given $p > 0$, let $s := a^p c a^{p+1}$

Given x, y, z - - - - -

[know that $y = a^k$ some $k > 0$]

Set ~~i~~ $i := 2$ (or anything ≥ 2)

Then $xy^2z = a^{p+k}ca^{p+1} \notin L$

Ex: Same lang except w has a c in its 2nd ~~[illegible]~~ half.

Given $p > 0$, $s := \cancel{a^{p-1}ca^p}$

$a^p c a^{p-1}$

Given x, y, z, $i := \underline{0}$

only i that works.

Strategic mistake: $s := a^{p+4}ca^{p-1}$

Then if $\underline{y} := \varepsilon$, $\underline{x} = aa$, $\underline{z} = a^{p+2}ca^{p-1}$

Then $xz = a^{p+2}ca^{p-1} \in L$ won't work

Last EX: $L := \{0^m 1^n : m, n \geq 0 \text{ and } m \neq n\}$

Prop: L is not regular.

Proof: Suppose L is regular.

Then $\overline{L}$ is regular.

Then $\overline{L} \cap L(0^* 1^*)$ is regular.

But $\overline{L} \cap L(0^* 1^*) = \{0^m 1^n : m, n \geq 0 \text{ and } m = n\}$

$$= \underbrace{\{0^n 1^n : n \geq 0\}}_{\text{not pumpable hence not regular}}$$

↯

$\therefore$ L is not regular.

CSCE 355
2/27/2023

Context-free languages & grammars
Derivations & parse trees

Good for describing programming language syntax.

A (context-free) grammar is basically a finite set of "rewrite" rules or "edit" rules allowing one substring to substitute for another.

Ex:

$$S \to 0S1$$
$$S \to \varepsilon$$

Can derive:

$$S \Rightarrow 0\underline{S}1 \Rightarrow 00\underline{S}11 \Rightarrow 000\underline{S}111 \Rightarrow 000111$$

$$S \Rightarrow \varepsilon$$

$$S \Rightarrow 0S1 \Rightarrow 01$$

$\{0^n 1^n : n \geq 0\}$ context-free language
(this one is not regular)

Ex:

$$S \to (S)S$$
$$S \to \varepsilon$$

gets all strings of well-balanced parentheses

Derive: $(())()$

$S \Rightarrow (S)S \Rightarrow ((S)S)S \Rightarrow ((S)S)(S)S$

$\Rightarrow (()S)(S)S \Rightarrow (())(S)S$

$\Rightarrow (())()S \Rightarrow (())()$

HWK: Derive: $((()))$

Ex: $\{a^n b^m c^n : m, n \geq 0\} = L$

Try: $\cancel{S \Rightarrow aSc}$, $\cancel{S \Rightarrow bS}$, $\cancel{S \Rightarrow \varepsilon}$

$S \to aSc$
$S \to bS$
$S \to \varepsilon$

won't work

Derive $aabbbcc \in L$:

$S \Rightarrow aSc \Rightarrow aaScc \Rightarrow aabScc$
$\Rightarrow aabbScc \Rightarrow aabbbScc \Rightarrow aabbbcc.$

But can ~~der~~ derive strings not in L:

$S \Rightarrow bS \Rightarrow baSc \Rightarrow bac \notin L$

Must be able to derive all strings in the language but no others.

Fix: $S \to aSc$ $\quad T \to bT$
$S \to T$ $\quad T \to \varepsilon$

correct

Can't derive bac, but can derive everything in L

~~$S \Rightarrow aSb \Rightarrow aaS$~~

$S \Rightarrow aSc \Rightarrow aaScc \Rightarrow aaTcc$

$\Rightarrow aabTcc \Rightarrow aabbTcc \Rightarrow aabbbTcc$

$\Rightarrow aabbbcc.$

Def: A context-free grammar (CFG) is a 4-tuple $\langle V, \Sigma, S, P \rangle$ where

- V is a (finite) alphabet. Symbols in V are called variables or nonterminals or syntactic categories (usually: uppercase Roman letters, e.g., $S, T, \ldots$)
- Σ is an alphabet (symbols in Σ are called terminals or tokens) and $V \cap \Sigma = \emptyset$
- $S \in V$ is called the start symbol
- P is a finite set of productions

(i.e., rewrite rules). Each production has the form,

$$A \to \alpha$$

where $A \in V$ and $\alpha \in (\underbrace{V \cup \Sigma}_{\text{grammar symbols}})^*$

A is the <u>head</u> and α is the <u>body</u> of the production.

E.g. $G = \langle \{S\}, \{0,1\}, S, \{S \to 0S1, S \to \varepsilon\} \rangle$

or

$G = \langle \{S,T\}, \{a,b,c\}, S, \{S \to aSc, S \to T, T \to bT, T \to \varepsilon\} \rangle$

<u>Def</u>: Let $G = \langle V, \Sigma, S, P \rangle$ be a CFG. A <u>derivation in G</u> is a sequence of the form,

$$\alpha_0 \Rightarrow \alpha_1 \Rightarrow \cdots \Rightarrow \alpha_n$$

where $\alpha_i \in (V \cup \Sigma)^*$ for $0 \le i \le n$.

$n \ge 0$ is the <u>length</u> of the derivation (# of arrows) and α_{i+1} is obtained from α_i by replacing a single occurrence of some nonterminal A in α_i by the body of some production ~~wh~~ with head A.

Generally, given strings $\alpha, \beta \in (V \cup \Sigma)^*$, say that $\alpha \Rightarrow \beta$ ("α derives β in one step") if

$$\alpha = \alpha_L \underline{A} \alpha_R \quad \text{for some } A \in V, \; \alpha_L, \alpha_R \in (\Sigma \cup V)^*$$

and

$$\beta = \alpha_L \underline{\gamma} \alpha_R \quad \text{for some } \gamma \in (\Sigma \cup V)^*$$

such that $A \to \gamma$ is a production of G.

<u>Def</u>: G as above. A derivation $\alpha_0 \Rightarrow \cdots \Rightarrow \alpha_n$ in G

is <u>complete</u> if

- $\alpha_0 = S$ start symbol
- $\alpha_n \in \Sigma^*$ only terminals

Let $w \in \Sigma^*$ be a string of terminals

w is <u>derivable in G</u> if there is a complete derivation in G

$$S \Rightarrow \cdots \Rightarrow w \qquad (S \Rightarrow^* w)$$

("derivation of w")

$\alpha, \beta \in (V \cup \Sigma)^*$, say

$\alpha \Rightarrow^* \beta$ if there is a derivation

$\alpha \Rightarrow \cdots \Rightarrow \beta$ (any # of steps)

(could be that $\alpha = \beta$).

<u>Def</u>: G be as above. The language of G is the set

$$L(G) := \{ w \in \Sigma^* : w \text{ is derivable in } G \}$$

<u>Def</u>: A language L is <u>context-free</u> (a CFL)

if $L = L(G)$ for some CFG G.

Grammar for arith expressions with

$+, *, -, /, (,)$

$\Sigma = \{+, *, -, /, (,), c\}$

$V = \{E\}$

productions

$$E \to E + E$$
$$E \to E - E$$
$$E \to E * E$$
$$E \to E / E$$
$$E \to c$$
$$E \to (E)$$

Derive $c * (c + c)$

$$E \Rightarrow \underline{E} * E \Rightarrow c * \underline{E} \Rightarrow c * (\underline{E})$$
$$\Rightarrow c * (\underline{E} + E) \Rightarrow c * (c + \underline{E})$$
$$\Rightarrow c * (c + c)$$

Leftmost derivation: at each step, substitute the leftmost nonterminal.

CSCE 355
3/1/2023

Ambiguity
Parse trees
Grammars for arith exprs & statements

$E \to E+E$
$E \to E-E$
$E \to E*E$
$E \to E/E$
$E \to c$
$E \to (E)$

$E \to E+E \mid E-E \mid E*E \mid E/E \mid c \mid (E)$

$c*c+c$

$E \Rightarrow E+E \Rightarrow E*E+E \Rightarrow \cdots \Rightarrow c*c+c$

$E \Rightarrow E*E \Rightarrow c*E \Rightarrow c*E+E \Rightarrow \cdots \Rightarrow c*c+c$

This grammar is ambiguous.

<u>Def:</u> A grammar G is ambiguous if there exists a string ~~of ter~~ w of terminals that is derivable via two or more leftmost derivations.

Parse trees $\equiv$ leftmost derivations $\equiv$ rightmost derivations

Def: Fix a grammar G. A parse tree of G is a rooted ordered tree where each node is labeled with a grammar symbol or ε, and such that each internal node is labeled by some nonterminal A and the children of A, read left to right, form the body of some production whose head is A.

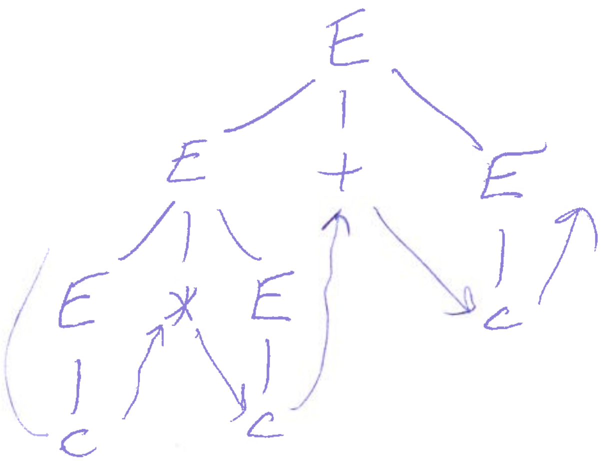
Ex: If $A \to \cancel{Abc}$ Abc is a production

then  can occur in a parse tree of G.

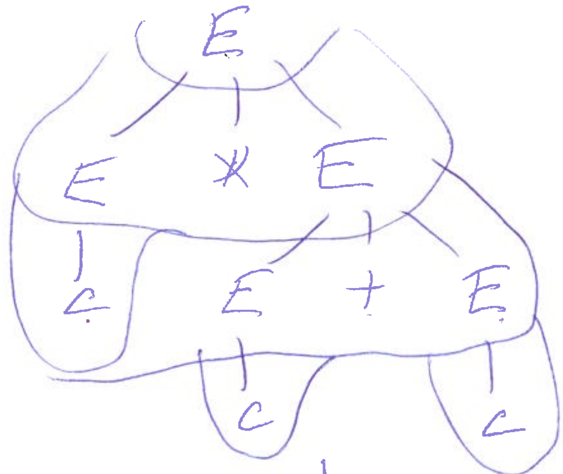
~~Prop~~: A parse tree is complete if the root is labeled with the start symbol and each leaf is labeled with either a terminal or ε.

The yield of a parse tree T is the string of grammar symbols obtained by concatenating the leaves of T in left-to-right traversal order (ε disappears in any such concatenation).

Ex: Parse tree of the grammar for arith exprs above yielding c*c+c:



Another one:



Prop: Given a string w of terminals, There is a one-to-one correspondence between complete leftmost derivations ~~of a string~~ of w and parse tree yielding w.

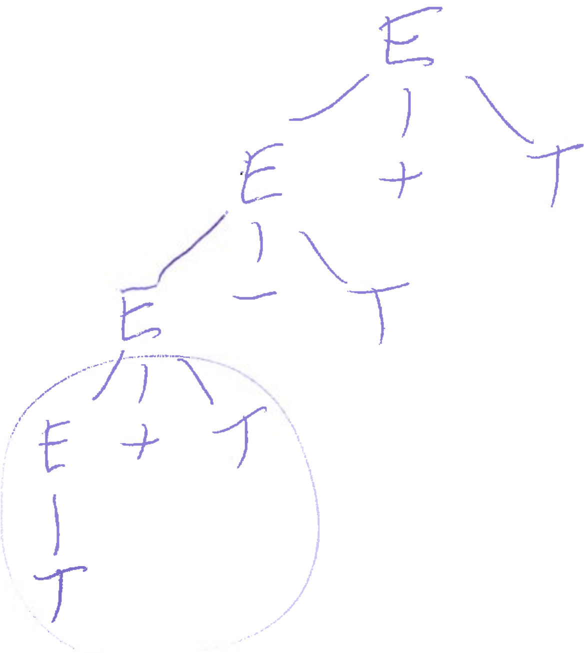
[also a 1-1 correspondence with rightmost derivations of w]

Cor: A string w of terminals is in L(G) iff there exists a complete parse tree of G yielding w.

Unambiguous grammar equivalent to the prev one
giving the same language

$E \to E + T \mid E - T \mid T$

E = expr
T = term



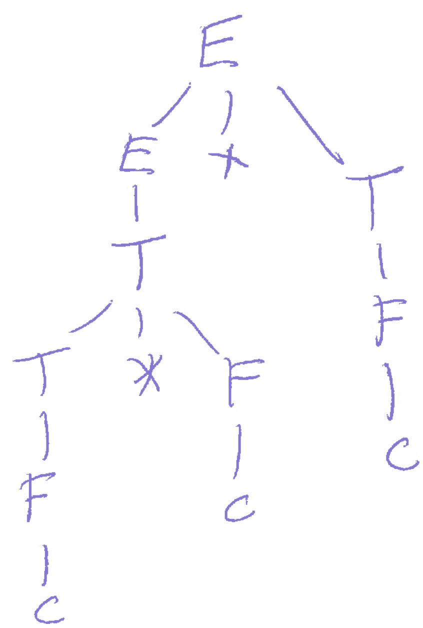
yield is $T + T - T + T$

$T \to T * F \mid T / F \mid F$

$F \to c \mid (E)$

This grammar is unambiguous.

Parse tree for $c * c + c$:

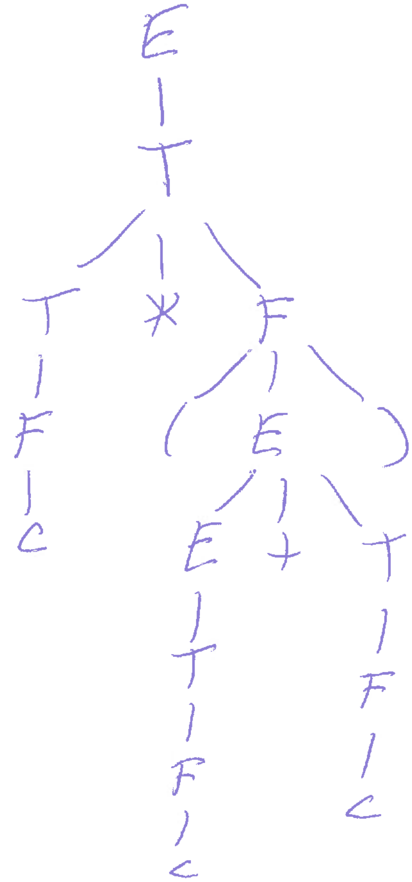


Parse tree for $c * (c + c)$

$E \to E+T \mid E-T \mid T$

$T \to T*F \mid T/F \mid F$

$F \to c \mid (E)$

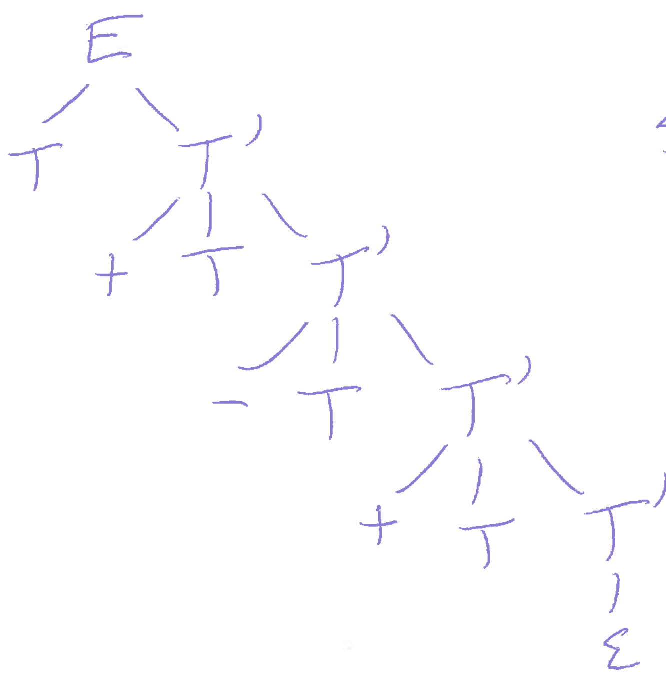


Another unambiguous grammar for arith exprs:

$E \to T\,T'$

$T' \to +\,T\,T' \mid -\,T\,T' \mid \varepsilon$

T' = "maybe more terms"



same yield:

$T + T - T + T$

$T \to F F'$

$F' \to * F F' \mid \cancel{\ } \mid \varepsilon$
$\quad / F F'$

$F \to c \mid (E)$

<u>Top-down parsing</u>: given the input, try to "build" a parse tree from the root down while reading the input.

~~2nd~~ Most recent grammar is suitable for top-down parsing.

Bottom-up parsing: try to build parse tree from leaves up by combining subtrees with a common parent. The 1st unamb. grammar is suitable for bottom-up parsing.

Grammar for statements:

S = "statement"

T = "test"

$S \to$ <u>if</u> T <u>then</u> S
$\mid$ <u>while</u> T <u>do</u> S
$\mid$ <u>if</u> T <u>then</u> S <u>else</u> S
$\mid$ <u>other</u>

control flow part of the grammar

This is ambiguous; 2 ways to parse

if T then if T then S else S

"dangling else ambiguity"

Ex: $S \to aSc \mid T$

$T \to bT \mid \varepsilon$

(unambiguous)

parse tree for aabbbcc