CS 3313
Foundations of Computing:

Properties of Context Free Languages – Part 1

http://gw-cs3313.github.io

1

CFGs and PDAs ?

- Grammars are a formalism for defining (generating) languages
- Automata are machine models to accept a class of languages

- CFGs generate Context Free languages
- PDAs accept context free languages

- Theorem: If L is a context free language (CFL) then there is a PDA M and a CFG G such that L = L(M) = L(G)

2

Designing a PDA for a CFG

- Recall: We described a regular language using a RegEx, and there was a procedure that (automatically) generated a DFA from that expression
- Question: If we provide a grammar for a CFL, then is there a procedure that (automatically) generates a PDA for that grammar ?
  - Think of the PDA as the parser generated from the grammar !

3

Generating PDA for a Grammar

- From results on Normal forms, any context free grammar can be expressed by an equivalent Greibach Normal Form (GNF) grammar where each production is of the form:

  $A \rightarrow a\,\boldsymbol{\alpha}$ where $\boldsymbol{\alpha} \in V^*$

Example:

$S \rightarrow aSB \mid aB$

$B \rightarrow b$

4

Derivations in the grammar…

S → aSB | aB   B → b

- Leftmost derivations – apply production to the leftmost variable in sentential form
  - S => aSB => aaSBB => aaaBBB => aaabBB => aaabbB => aaabbb

5

Derivations in GNF and Moves in a PDA

- … $S =>^* a_1 a_2 a_3 \ldots a_i A_i \alpha_i \ldots \alpha_2 \alpha_1$, where $a_i \in T$ and $\alpha_i \in V^*$
  - Leftmost derivation, at each step we generate terminal symbol a_i
- PDA reads input from left to right
  - It reads $a_1 a_2 a_3 \ldots a_i \ldots$.
- G derives: $S =>^* a_1 a_2 a_3 \ldots a_i A_i \alpha_i \ldots \alpha_2 \alpha_1$
  - *Eventually* $a_1 a_2 a_3 \ldots a_i A_i \alpha_i \ldots \alpha_2 \alpha_1 =>^* a_1 a_2 a_3 \ldots a_i x$
- *iff*
- PDA simulates $(q, a_1 a_2 a_3 \ldots a_i x, S) \vdash^* (q, x, A_i \alpha_i \ldots \alpha_2 \alpha_1)$

6

PDA for a Context Free Language

- Theorem: For every context free language L, there exists a PDA M such that L= L(M).
- Proof:
  - If L is a CFL then it is generated by some GNF grammar G=(V,T,P,S) with L(G)=L
  - Key idea: construct a PDA that simulates leftmost derivations in G

7

PDA for a Context Free Language

- L is generated by a GNF grammar G =(V,T,P,S)
  - All productions are of the form $A \rightarrow a\ \boldsymbol{\alpha}$ where $a \in T$ and $\boldsymbol{\alpha} \in V^*$
- PDA $M= (\{q_0,q_1,q2\}, T, V \cup \{Z\}, \delta, q_0, \{q_2\})$
  - Stack alphabet = Set of Variables in G and the start stack symbol Z
  - Alphabet = set of terminal symbols T
  - $\delta(q_0, \lambda, Z) = \{(q_1, S\ Z)\}$ /* push S to stack, goto q_1 and start simulation
  - $\delta(q_1, \lambda, Z) = \{(q_2, Z)\}$/* if no input and 'empty stack' go to accept state
  - $\delta(q_1, a, A)$ contains (q_1, α) whenever $A \rightarrow a\ \alpha$ is a production in P
    - Simulate a derivation $A \Rightarrow a\ \alpha$

8

Proof – contd..

- Key idea: PDA reads a, pops A from stack, and pushes α to stack if $A \rightarrow a\ \alpha$ is a production in the grammar
- PDA simulates leftmost derivations in G
  - Input is processed left to right
- Prove: $S =>^* x\ \alpha$ (using leftmost derivation) if and only if

$$(q_1, x, SZ) \vdash^* (q_1, \lambda , Z)$$

- Note: from definition of δ, $(q_0, x, Z) \vdash (q_1, x, SZ)$
  - This starts PDA with S on TOS
- Proof by induction:
  1. If $(q_1, x, SZ) \vdash^* (q_1, \lambda , z)$ then $S =>^* x\ \alpha$
  2. If $S =>^* x\ \alpha$ then $(q_1, x, SZ) \vdash^* (q_1, \lambda , \alpha Z)$

9

Example: PDA from CFG

- $S \rightarrow aSB \mid aB$ $\qquad B \rightarrow b$
- PDA $M= (\{q_0,q_1,q2\}, \{a,b\}, \{S,B,Z\}, \delta, q_0,\{q_2\})$
  - $\delta(q_0, \lambda, Z) = \{(q_1, S\ Z)\}$ /* push S to stack, goto q_1 and start simulation
  - $\delta(q_1, \lambda, Z) = \{(q_2, Z)\}$/* if no input and 'empty stack' go to accept state
  - $\delta(q_1, a, S)$ *contains* $\{ (q_1, SB)\ (q_1, B) \}$
  - $\delta(q_1, b, B)$ *contains* $\{ (q_1, \lambda) \}$
    - *Because we have productions* $S \rightarrow aSB$ *and* $S \rightarrow aB$
    - *and* $\delta(q_1, a, A)$ contains (q_1, α) whenever $A \rightarrow a\ \alpha$ is a production in P
- *Derivation for aabb:* $S => aSB => aaBB => aabB => aabb$
- *In PDA:*

$(q_0, aabb, Z) \vdash (q_1, w, SZ) \vdash$

10

PDA to CFG

- Theorem: If L =L(M) for a PDA M, then there is a context free grammar G such that L(G)=L(M)
- Proof: Read theorem 7.2 in textbook.
- Outline – given a PDA, we want to generate a grammar that simulates PDA via leftmost derivations
- The proof is rarely used to construct grammars – its purpose is to show the equivalence of the two formalisms CFG and PDA

11

CFG to PDA Conversion "Algorithm"

- The constructive proof can be implemented as an algorithm that takes a GNF Grammar G and generates a PDA
- We can then feed this PDA to a program that simulates/implements any PDA
  - We have an automated process for "writing" a parser!
- BUT…..the conversion/proof may lead to a non-deterministic PDA
  - Question: Can we convert the grammar to a deterministic PDA ?

12

Deterministic Pushdown Automata

- A *deterministic pushdown automata (DPDA)*never has a choice in its move
- Restrictions on dpda transitions:
  - Any (state, symbol, stack top) configuration may have at most one (state, stack top) transition definition
  - If the DPDA defines a transition for a particular (state, λ, stack top) configuration, there can be no input-consuming transitions out of state s with a at the top of the stack
- Unlike the case for finite automata, a λ-transition does not necessarily mean the automaton is nondeterministic

13

Deterministic Context-Free Languages

- A context-free language L is *deterministic* (DCFL) if there is a *dpda* to accept L
- Sample deterministic context-free languages:

  $\{ a^n b^n : n \geq 0 \}$

  $\{ wcw^R : w \in \{a, b\}^* \}$
- Theorem: Deterministic and nondeterministic pushdown automata are not equivalent: there are some context-free languages for which no DPDA exists that accepts the language
  - *Syntax of most programming languages is deterministic context free*

14

Next: Properties of Context Free Languages

- What are the properties of CFLs ?
- What types of languages are CFL ?
  - Can all properties/semantics of a programming language be captured by a CFL ?
  - Can natural languages be described by CFGs ?
    - Can we determine ambiguity and remove ambiguity ?
    - Can we parse natural languages using a CFG for the syntax ?
- If we combine CFLs using set operations, is the resulting language CFL ?
- How do we prove if a language is not context free ?
  - Pumping lemma for CFLs !!

15

Why bother with Properties/limits of CFLs – Ex1

- Exercise in abstraction:
- Scenario: We ”update” our programming language (defined by grammar G_1) from v1.0 to a new ‘version’ v2.0 defined by a grammar G_2
- we would like to design a compiler that can parse a program in version 1.0 or a (legacy) program in version 2.0
- Is this possible ?
- Rephrase the question: Is there a context free grammar that accepts the union of the two languages v1.0 and v2.0 ?

16

Why bother with Properties/limits of CFLs –Ex2

- Exercise in abstraction:
- Scenario: In a program, we have function declaration and then a function call.
  - The actual and formal parameters need to match
  - Ex: int foo(int x, char y)…. and main has: z= foo(a,b)
    - a must be an int, b must be a char
- Question: Can this property be described/specified by a context free grammar ?
- Abstraction: the property can be captured by $\{a^n b^m c^n d^m\}$
  - a^n, b^m are formal parameters – n of type a (int), m of type b (char)

17

Pumping Lemma: Intuition

- Informally: DFAs don't have external memory, so languages that require "storing" counts, strings, etc. are likely to not be regular
  - Ex: {equal number of a's and b's}, $\{ ww^R \}$,….
- Recall the pumping lemma for regular languages.
  - It told us that if there was a string long enough to cause a cycle in the DFA transition graph, then we could "pump" the cycle and discover an infinite sequence of strings that had to be in the language.
  - Apply it using the 2-person game:
    - You pick the string after adversary picks n (i.e., you cannot specify a value for n)

18

18

Intuition for CFLs

- For CFL's the situation is a little more complicated.
- PDAs have external memory – a stack
  - But stack is limited in its capabilities
    - One "counter"
    - If you store something in the stack then when you check storage (i.e., pop the stack) the reverse pattern is popped.
  - Informal limits:
    - Languages that require multiple counters { $a^n b^n c^n$}
    - Languages that require exact patterns {ww}
  - *If you push a pattern into the stack in the "first part" of the string, then that pattern repeats in "second part"*
- We can always find two pieces of any sufficiently long string to "pump" in tandem.
  - That is: if we repeat each of the two pieces the same number of times, we get another string in the language. 19

19

Properties of Parse Trees

- Lemma 1: Let G in Chomsky Normal Form (CNF), then for any parse tree with yield w (string w generated by grammar) if n is the length of the longest path in the tree then $|w| \leq 2^{n-1}$.
- Proof: What type of tree is a parse tree for a CNF grammar ? – binary tree
- Recall CS1311 !!!
- Or prove by induction on length of the path
  - Basis: n=1 derivation must be $S \rightarrow a$
  - Ind.Step: Since G is in CNF, $S \rightarrow AB$ and $A \Rightarrow^* w_2$ and $B \Rightarrow^* w_2$
    - A derives substring w_1 with path $\leq$ n-1
    - B derives substring w_2 with path $\leq$ n-1
    - From IH: $|w_1| \leq 2^{n-2}$ and $|w_2| \leq 2^{n-2}$
    - $|w| = |w_1| + |w_2| \leq 2^{n-1}$

20

Properties of parse trees for arbitrarily long strings

- From previous theorems, if L is a CFL then there exists CNF G=(V,T,P,S) such that L=L(G)
  - L is generated by a CNF grammar G
  - |V| =m   finite set of variables – m variables

- We are implicitly discussing infinite languages
  - If a language is finite then it is a regular language
    - Implies regular grammar (subset of CFLs)

- Suppose we have z ϵ L(G) and $|z| \geq n = 2^m$
- What can we say about parse tree for z ?
  - From lemma 1, parse tree for z must have a path of length at least m+1
    - Yield of the tree is $\leq 2^m$

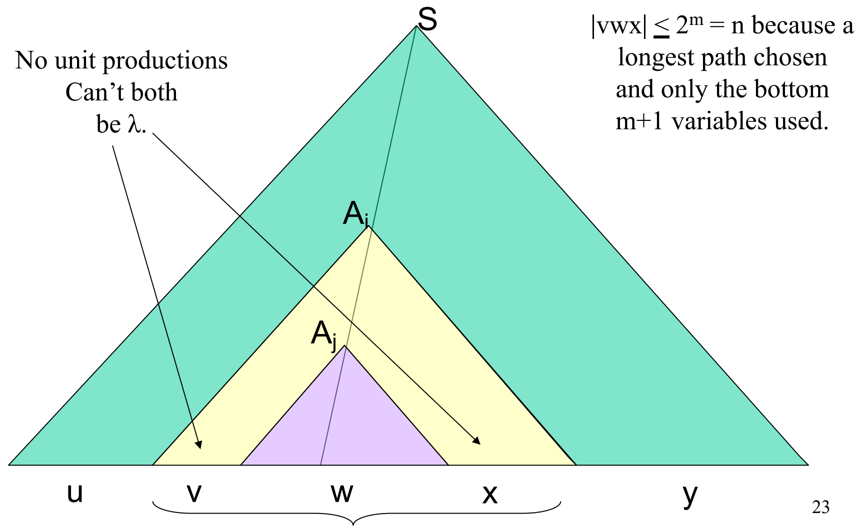
21

Parse tree properties

- If path has length $k \geq m+1$, then it has k+1 vertices/nodes in the path
  - Last vertex is labelled with a terminal
- Therefore path has k internal nodes labelled with variables of the grammar
  - These are $A_1, A_2 \ldots A_i, \ldots A_j \ldots A_k$
  - A_1 is the start symbol S
- We have m distinct variables => from pigeon hole principle, at least two of the vertices A_i and A_j are the same variable
  - In fact, from the leaf, these two occur within path of length m+1
- So what does this tell us about the parse tree for z ?

22

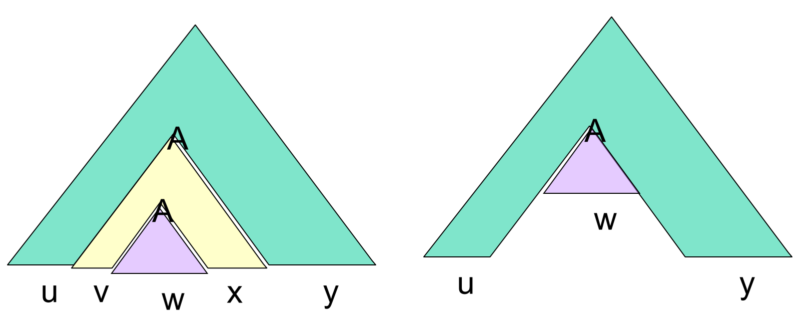
Parse Tree in the Pumping-Lemma Proof



23

23

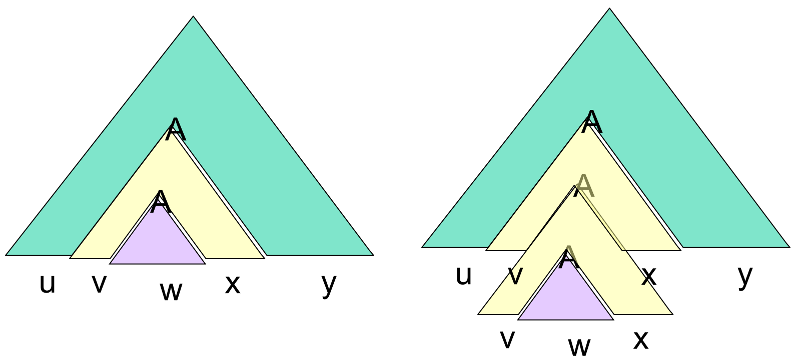
Pump Zero Times



24

24

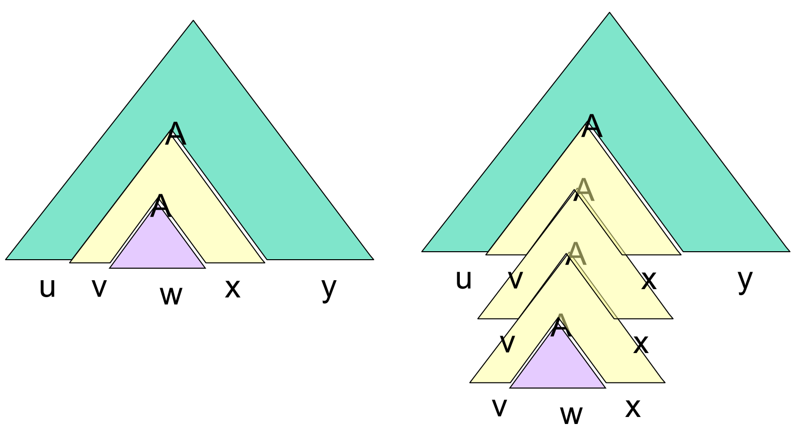
Pump Twice



25

25

Pump Thrice



Pump 4 times, 5 times, Etc., Etc.

26

26

Statement of the CFL Pumping Lemma

For every context-free language L

There is an integer n, such that

For every string z in L of length $\geq n$

There exists $z = uvwxy$ such that:

1. $|vwx| \leq n$.
2. $|vx| > 0$.
3. For all $i \geq 0$, $uv^i wx^i y$ is in L.

27

27

How do use the pumping lemma: recall 2 person adversarial game

- **For all** context free languages L, **there exists** *n*…**for all z** in L

….**there exists** uvw*xy*….

- Logical statements/assertions that have several alternations of for all and there exists quantifiers can be thought of as a game between two players

- Application of the pumping lemma can be seen as a two player game (of 5 steps)

28

Pumping Lemma as Adversarial Game

1. Player 1 (we) picks language we want to show is not a CFL
2. Player 2 "adversary" gets to pick n
   - We do not know the value of n, and must plan for all values of n
3. We get to pick z, and may use n as a parameter
   - Can express z using the parameter n
4. Adversary gets to break z into $uvwxy$ subject only to the constraints that $|vwx| \leq n$ *and* $|vx| \geq 1$.
5. We "win" the game, if we can, by picking i and showing uv^iwx^iy *is not in L*
   - We have to show this for all cases of how adversary breaks z into $uvwxy$

29

Example: $L = \{ a^ib^ic^i \}$

- Informally: CFL (PDA) can count & match two groups of symbols but not three (since we have one counter)
- Apply pumping lemma to prove L is not CFL
- Assume L is CFL
- Let n be the constant of the lemma.
- Pick $z = a^nb^nc^n$
- Big difference from pumping lemma for regular languages
  - For regular languages, the pumping lemma allowed us to focus on the first n symbols/locations in the string
  - In CFL, the lemma only states $|vwx| \leq n$
  - This suggests we have to consider different cases where *vwx* can occur!
  - ***Prove contradiction in every case!***
    - *No matter how adversary breaks up vwx, we prove a contradiction*

30

Example: Cases for *vwx* for $L = \{ a^i b^i c^i \}$

1. *vwx* is entirely within a^n

2. vwx is entirely within b^n

3. *vwx* is entirely within c^n

4. *vwx* has two symbols (a and b, or b and c)

Example: Cases for *vwx* for $L = \{ a^n b^n c^n \}$

1. *vwx* is entirely within a^n
   - $u=a^j\ v=a^k\ w=a^l\ x=a^m\ \ y= a^{n-j-k-l-m}\, b^n\, c^n$ $\qquad 1\leq k+m \leq n$
   - $z' = uv^2wx^2y = a^{n+k}\, b^{n+m}\, c^n$ - *more a's than b's , c's. contradiction*
2. *vwx* is entirely within b^n
   - $u=a^n b^j\ \ v=b^k\ w=b^l\ x=b^m\ y= b^{n-j-k-l-m}\, c^n$ $\qquad 1\leq k+m \leq n$
   - $z' = uv^2wx^2y = a^n b^{n+k+m} c^n$ *more b's than a's , c's. contradiction*
3. *vwx* is entirely within c^n
   - $u=a^n\, b^n\, c^j\ v=c^k\ w=c^l\ x=c^m\ y= c^{n-j-k-l-m}$ $\qquad 1\leq k+m \leq n$
   - $z' = uv^2wx^2y = a^n b^n c^{n+k+m}$ *more c's than a's , b's. contradiction*

- what if *vx* has two symbols (*a* and *b*, or *b* and *c*)

Example: Cases for vwx for $L = \{ a^n b^n c^n \}$

4. vx has two different symbols (a and b, or b and c)
   - $v \in \{a^+ b^+\}$ $x \in \{b^+ c^+\}$ $v \in \{a^+ b^+\}$ $x \in \{b^+ c^+\}$
   - Consider $z' = uv^2wx^2y$: pattern of *a's, b's, and c's ?*
5. v is in a^* *and* x is in b^*
   - $u=a^j$ $v=a^k$ $w=a^{n-j-k}b^l$ $x=b^m$ $y= b^{n-m} c^n$ $1 \leq k+m \leq n$
   - Consider $z' = uv^2wx^2y : a^{n+k} b^{n+m} c^n$ *- since (k+m)>1, either n+k>n or n+m>n (or both) => less c's than a's or b's - contradiction*
6. *v is in b* and x is in c**
   - $u=a^nb^j$ $v=b^k$ $w=b^{n-j-k}c^l$ $x=c^m$ $y= c^{n-l-m}$ $1 \leq k+m \leq n$
   - Consider $z' = uv^2wx^2y : a^n b^{n+k} c^{n+m}$ *- since (k+m)>1, either n+k>n or n+m>n (or both) => less a's than b's or c's - contradiction*

33

Exercise: $L_2 = \{ a^i b^j c^i d^j \}$ a's = c's and b's = d's

- Intuition: L_2 is likely not CFL. If we push a's and b's on the stack (to remember how many), then we pop b's before a's

1. Assume L_2 is CFL *you pick*
2. Let n be the constrant *adversary picks*
3. Consider z= $a^n b^n c^n d^n \in L_2$. *you pick*
4. *$z= uvwxy$, $|vwx| \leq n$, and $|vx| \geq 1$* *adversary picks*
5. For every $i \geq 0$, $uv^iwx^iy \in L_2$ *you pick I*

- **Question: (a) Find all cases for vwx and then (b) show contradiction for each case**

34

"weakness" of the Pumping Lemma

- It allows vwx to be anywhere in the string
  - In contrast to pumping lemma for regular languages
- Looking at the proof, we can see the opportunity to limit the 'areas' to pump…..leads to a stronger pumping lemma:

Ogden's lemma: For every context-free language L, there is an integer n (which may in fact be the same as for the pumping lemma), such that if z is any string in L and we mark any n or more positions of z as "distinguished", then z = uvwxy such that:

1. vwx has at most n distinguished positions
2. vx has at least one distinguished position
3. For all $i \geq 0$, uv^iwx^iy is in L.

Pumping lemma essentially marks all positions as distinguished!

35

Example $L_3 = \{ w\ w \mid w \in \{a,b\}^* \}$

- Is this language a CFL ?
- If we push w into the stack,

  what pattern is popped from the stack ?

36

Example $L_3 = \{ w\, w \mid w \in \{a,b\}^* \}$

- Prove it is not CFL
- Let n be the constant of the lemma
- Consider $w=a^n b^n$, *i.e.*, $z= a^n b^n a^n b^n \in L_3$
- What are the possible cases for vwx ?

aa......aabb……bb aa……aabb……bb

37

Example $L_3 = \{ w\, w \mid w \in \{a,b\}^* \}$

- Let n be the constant of the lemma and consider $z= a^n b^n a^n b^n \in L_3$
- What are the possible cases for vwx ?

aa......aabb……bb aa……aabb……bb

- Case 1: $v=a^j$ $x=a^k$ pick $i=2$ and $z'= uv^2wx^2y$
- Case 2: $v=a^j$ $x=b^k$ pick $i=2$ and $z'= uv^2wx^2y$
- Case 3: $v=b^j$ $x=b^k$ pick $i=2$ and $z'= uv^2wx^2y$
- Case 4: $v=b^j$ $x=a^k$ pick $i=2$ and $z'= uv^2wx^2y$
- Case 5: $v=a^j$ $x=a^k$ pick $i=2$ and $z'= uv^2wx^2y$
- Case 6: $v=a^j$ $x=b^k$ pick $i=2$ and $z'= uv^2wx^2y$
- Case 7: either v or x consists of two different symbols (a^+b^+ or b^+a^+)

$1 \leq |vwx| \leq n$

therefore $1 \leq j+k \leq n$

38

Example $L_3 = \{ w\, w \mid w \in \{a,b\}^* \}$

- We proved L_3 is not CFL
- How about L = $\{ x\, y \mid x \neq y \text{ and } x,y \in \{a,b\}^* \}$

- There is a position in *x* such that the same position in *y* is a different symbol
- $x = x_1 a x_2 \quad y = y_1\, b\, y_2$ and $|x_1| = |y_1| = k$ and $|x_2| = |y_2| = l$
  - x_1, x_2, y_1, y_2 can be arbitrary strings – only their lengths matter to get *a* and *b* are the same position in both halves
- *PDA:* read first *k* symbols and push *"1"* to stack, store *a (in state),* then read and pop *k* symbols –and repeat in second half reading *y*

39