CS 3313
Foundations of Computing:

A Parsing Algorithm for Context Free Grammars

http://gw-cs3313.github.io

1

Context Free Grammars

- A context free grammar is a grammar G=(V,T,P,S) where all production rules are of the form: $V \rightarrow (V \cup T)^*$
  - Production rules have exactly one variable on the left and a string consisting of variables and terminals on the right.
- Derivations: $\alpha A \beta \Rightarrow \alpha\gamma\beta$ if A -> γ is a production, =>*
  - Sentential form: α is in sentential form if S =>* α
- Derivation or Parse Trees
  - S is root node, Variables are internal nodes, terminal is leaf node, yield are leaves in pre-order traversal (left to right leaves)
- Equivalence of Parse Trees and Derivations
- If G is a CFG, then L(G), the *language of G*, is

  L(G) = *{w | S =>* w and w is a string over set T}*.

2

Next…A parsing algorithm -- testing Membership

- Given a CFG G, and an input string w,

  determine if string *w* is in *L(G)*.
  - Generate the parse tree
- Why is this useful …. Parsing is the first step in the compilation process
- How ?
  1. First ”clean up” the grammar
  2. Next, convert to a Normal Form
  3. Design a parsing algorithm for any CFG expressed in the normal form

3

3

Simplification/Clean up of CFGs

- **Simplify** the rules: three-step procedure
  - Removing λ-productions
  - Removing unit-productions
  - Removing non-terminating or non-reachable variables
- **Next step**: converting the rules to a certain "normal form".
- **Before** we apply our automation procedures.

4

Normal Forms for Context Free Grammars

- Any context free grammar can be converted to an equivalent grammar in a "normal form"

- **Chomsky Normal Form (CNF):**

  All productions are of the form $A \rightarrow a$ or $A \rightarrow BC$ where a is a terminal symbol and A,B,C are variables

- **Greibach Normal Form (GNF):**

  All productions are of the form $A \rightarrow a\alpha$ where a is a terminal and α is a string of variables (possibly empty)

5

Testing for Membership – a Parsing Algorithm

- Simple algorithm: Convert CFG to a Greibach Normal Form (all productions are of the form A →aα)
  - For string *w* of length *n,* we have *n* derivation steps.
  - At each step, explore all productions.
  - Time: $O(\ |P|^n)$ – this is exponential (in length of input string *w*)

- Can we do better ?.....Yes
  - Start with conversion to CNF

6

Testing Membership

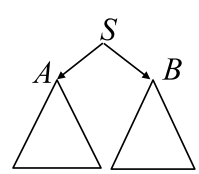
- Want to know if string *w* is in *L(G)*.
- Assume G is in CNF.
  - Or convert the given grammar to CNF.
  - w = λ is a special case, solved by testing if the start symbol is nullable.
- Cocke Younger Kashimi Algorithm (*CYK*) is a good example of dynamic programming and runs in time $O(n^3)$, where $n = |w|$.

7

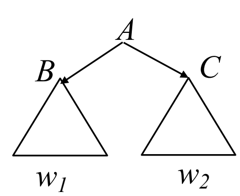
7

Observations (derivations in CNF grammar)

- CNF Grammar: suppose S derives string w
- Parse tree:



- Generalize to variable A derives a string $w = w_1 w_2$



8

Setting up our solution/algorithm: Notations

- Important: these notations are a bit different from notations in the book, but the end algorithm works in the same manner
- Input string w has length n – i.e, consists of n terminal symbols: $w = a_1 a_2 \ldots a_n$ where each $a_i \in T$
  - Ex: $w = abcaab$ $a_1=a$ $a_2=b$ $a_3= c, \ldots$
- Define a substring x_{ij} (of w) as the the substring starting at position i and having length j
  - Ex: $x_{13} = abc$ $x_{22}= bc$ $x_{33}= caa$ $x_{15}= abcaa$ $w=x_{16}=abcaab$
- For a substring x_{ij}, define V_{ij} to be set of variables that derive x_{ij}
  - $V_{ij} = \{ A \mid A =>^* x_{ij} \}$ *note* $1 <= i <= n\text{-}j$

9

Algorithm

- Claim is that we can construct V_{ij} interatively
- Basis: $V_{i1} = \{ A \mid A \rightarrow x_{i1}$ is a production $\}$
- Ind. $A =>^* x_{ij}$ iff $A \rightarrow BC$ and for some k, $1<= k <= j$,

  $B =>^* x_{ik}$ and $C =>^* x_{i+k,\, j-k}$
- Since k, $j\text{-}k < j$ the IH holds.
- w is in L(G) **iff** $S \in V_{1n}$ (since $w = x_{1n}$)

$$V_{ij} = \{ A \mid A \rightarrow BC, \text{ and for some } k, B \text{ is in } V_{ik} \text{ and } C \text{ is in } V_{i+k,j-k} \}$$

10

CYK Algorithm

Input: CFG G=(V,T,P,S) in CNF, Input string w of length n

1. for $i=1$ to n

   $V_{i1} = \{A \mid A \rightarrow a$ is in P and $x_{i1} = a\}$

2. for $j=2$ to n

   - For $i=1$ to $n-j+1$ {

     $V_{ij} = \emptyset$

     for $k=1$ to $j-1$ {

     $V_{ij} = V_{ij} \cup \{ A \mid A \rightarrow BC$ is a production in P,

     B is in V_{ik}

     C is in $V_{i+k,j-1}$ }

     }}

3. w is in L(G) if S is in V_{1n}

11

Time Complexity

- Step 1: takes $O(n)$ to examine each of the n symbols
  - Assume P is a constant.
- Step 2: $O(n^3)$
  - Outer j loop iterates $O(n)$
  - The i loop iterates $O(n)$
  - For each of the n^2 iterations, the k loop iterates $O(n)$
- Dynamic programming formulation
  - Construct solution for size n in terms of sizes $n-1$
    - Principle of optimality needs to hold

12

Example: Application of CYK Algorithm

- $S \to AB \mid BC$ $\quad A \to BA \mid a$ $\quad B \to CC \mid b$ $\quad C \to AB \mid a$
- $w = baaba$ (length 5), so i,j iterate from 1 to 5.

- Some sample V_{ij}
- To compute V_{31}, $x_{31} = a$. $V_{31} = \{ X \mid X \to a \text{ is in } P\}$
  - $V_{31} = \{ A, C\}$
- To compute V_{12}: $X \to YZ$ in P and
  - check if $Y \in V_{11}$ and $Z \in V_{21}$
- To compute V_{23} : $X \to YZ$ in P and
  - Check for Y in V_{21} and Z in V_{32}
  - Check for Y in V_{22} and Z in V_{41}

13

Example: Application of CYK Algorithm

- $S \to AB \mid BC$ $\quad A \to BA \mid a$ $\quad B \to CC \mid b$ $\quad C \to AB \mid a$
- w = baaba (length 5), so i,j iterate from 1 to 5.

i=1 j=2

i

j

B	A,C	A,C	B	A,C
S,A	B	S, C	A,S	

14

Example: Application of CYK Algorithm

- $S \rightarrow AB \mid BC$ $A \rightarrow BA \mid a$ $B \rightarrow CC \mid b$ $C \rightarrow AB \mid a$
- w = baaba (length 5), so i,j iterate from 1 to 5.

i →

j ↓

B	A, C	A, C	B	A, C
S, A	B	S, C	A, S	
{}	B	B		
{}	B			
S, A, C				

S is in V_{15} therefore w is in L(G)

15

Summary

- CFGs can be simplified and converted to CNF form
- CYK Algorithm provides a polynomial time $O(n^3)$ "parsing" algorithm
  - This is still time consuming if input is a large program
- Luckily syntax of most programming languages form a subset of CFGs known as Deterministic Context Free
  - Lend themselves to an O(n) parsing algorithm
- YACC: yet another compiler compiler
  - Standard tool in most Unix distributions
  - Generates a parser when given the grammar
    - Input is Grammar, and output is a parser
- Next: Then properties of CFLs…what languages are not CFL?
  - Equivalence of PDAs and CFGs

16

Exercise: CYK Algorithm

Grammar: S -> AB, A -> BC | a, B -> AC | b, C -> a | b

String w = ababa

i

j
