CS 3313
Foundations of Computing:

CFG Normal Formsand a Parsing Algorithm

http://gw-cs3313-2021.github.io

Simplification and Parsing

- 1. Simplification rules: transform a grammar such that:
  - Resulting grammar generates the same language
  - and has “more efficient” production rules in a specific format

- 2. Normal Forms: express all CFGs using a standard “format” for how the production rules are specified
  - Definition of CFGs places no restrictions on RHS of production
  - It is convenient (for parsing algorithms) to restrict to a standard form
    - Chomsky Normal Form (CNF) or Greiback Normal Form (GNF)
- 3. Parsing Algorithm: Design a parsing algorithm that takes a grammar in a standard form (CNF) to check if string w is generated by grammar G.

CFG Simplification (Cleanup) Algorithms

1. Remove productions
2. Remove Unit Productions
3. Remove Useless Symbols and Production
    1. Remove variables that do not derive terminal strings
    2. Remove variables that are not reachable from S

- After the simplification process, a CFG has productions where right hand side has length more than two or is a single terminal symbol

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 $A \to a$ or $A \to BC$ where a is a terminal and A, B, C are variables

- Greibach Normal Form (GNF): All productions are of the form $A \to a\alpha$ where a is a terminal and α is a string of variables (possibly empty)

Conversion to CNF

- **Theorem 6.6**: Any CFG $G = (V, T, S, P)$ with $\lambda \notin L(G)$ has an equivalent grammar $\hat{G} = (\hat{V}, \hat{T}, S, \hat{P})$ in CNF.
- **Step 1**: Constructing $G_1 = (V_1, T, S, P_1)$ from G by considering all productions P in the form

  $A \rightarrow x_1 x_2 \ldots x_n$ where each x_i is either in V or T.
  - Add variable V_a and production $V_a \rightarrow a$ for each terminal a
  - If x_i is a terminal a, replace with V_a
- **Step 2:** For rules with $A \rightarrow C_1 \ldots C_n, n > 2$, we introduce new variables $D_1, D_2, \ldots$ and put into $\hat{P}$ the productions
  - $A \rightarrow C_1 D_1$
  - $D_1 \rightarrow C_2 D_2 \ \ldots \ \ldots$
  - $D_{n-1} \rightarrow C_{n-1} C_n$, where each $A, D_1, \ldots, D_{n-1}$ is in CNF.

Testing for Membership – a Parsing Algorithm

- Simple algorithm: Convert CFG to a Greibach Normal Form (all productions are of the form $A \rightarrow a\alpha$)
  - 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

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 = ϵ 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|.

Observations (derivations in CNF grammar)

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

- Generalize to variable A derives a string $w = w_1w_2$

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 $a_i \epsilon T$
  - 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}
  - $V_{ij} = \{ A \mid A \Rightarrow^{*} x_{ij} \}$ note $1 <= i <= n-j$

Algorithm

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

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

V_{ij} = { A | A→ BC, and

for some k, B is in V_{ik} and C is in $V_{i+k,j-k}$ }

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 $V_{i+k,j-k}$ }
        }}
3. w is in L(G) if S is in $V_{1n}$
```

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

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.

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

Example: Application of CYK Algorithm

- S →AB | BC   A→ BA| a   B → CC |b   C→ AB| 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	

Example: Application of CYK Algorithm

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

$$\xrightarrow{\quad\quad\quad i \quad\quad\quad}$$

j ↓

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

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

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: Return to automaton models for CFLs
- Then properties of CFLs…what languages are not CFL?

Exercise: CYK Algorithm

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

String w = ababa