Formal Languages, Automata and Computation

Context Free Languages

Carnegie Mellon University in Qatar

SUMMARY

- Describing nonregular languages
- Grammars as finite descriptions of infinite sets
- Context-free Grammars and context-free languages
- Derivations and parse trees
- Ambiguity
- Writing grammars

Grammar Examples

- Consider $L = \{a^n b^n \mid n \geq 0\}$
- $S \rightarrow aSb$
  - a's and b's are generated in the right order and in equal numbers
- $S \rightarrow \epsilon$
  - get rid of any remaining S at the end.

Grammar Examples

- Consider $L = \{a^n b^m \mid m > n \geq 0\}$
- $S \rightarrow AB$
- $A \rightarrow aAb \mid \epsilon$
  - a's and b's are generated in the right order and in equal numbers, followed by B
- $B \rightarrow bB \mid b$
  - Generate 1 or more (additional) b's

Grammar Examples

- $L = \{a^n b^{2n} \mid n \geq 0\}$
  - $S \rightarrow aSbb \mid \epsilon$
- $L = \{a^{n+2} b^n \mid n \geq 1\}$
  - $S \rightarrow aaA$,
  - $A \rightarrow aAb \mid ab$

Grammar for Arithmetic Expressions

- $L \rightarrow a \mid b \mid \cdots \mid z$ (letters)
- $D \rightarrow 0 \mid \cdots \mid 9$ (digits)
- $V \rightarrow L \mid V\ L \mid V\ D$ (variables)
- $N \rightarrow D \mid N\ D$ (positive numbers)
- $F \rightarrow V \mid N \mid (E)$ (factors)
- $T \rightarrow F \mid T * F \mid T/F$ (terms)
- $E \rightarrow T \mid E + T \mid E - T$ (expressions)
- E is the start symbol.

Let us generate $(v23 + 456) * k23/(a - b * 34)$ as an exercise.

Ambiguity

- Remember a boy with a flower sees a girl with a telescope?
- We say that a grammar generates a string ambiguously, if the string has two different parse trees (not just two different derivations)
- A derivation of a string w in a grammar G is a leftmost derivation if at every step, the leftmost remaining variable is the one replaced.

AMBIGUITY

DEFINITION

A string w is derived ambiguously in context-free grammar G if it has two or more different leftmost derivations. Grammar G is ambiguous if it generates some string ambiguously.

- Sometimes an ambiguous grammar can be transformed into an unambiguous grammar for the same language.
- Some context-free grammars can be generated only by ambiguous grammars. These are known as inherently ambiguous languages.
  - $L = \{a^i b^j c^k \mid i = j \text{ or } j = k\}$

Grammar Transformations

- Some types of productions cause problems in some uses of grammars.
- ϵ-productions: $A \rightarrow \epsilon$
  - Intermediate sentential forms in a derivation get shorter and this has computational implications.
- Unit productions: $A \rightarrow B$.
  - Such a rule does not achieve much except for lengthening the derivation sequence.
  - There may be inadvertent "infinite loops": e.g., if $A \stackrel{*}{\Rightarrow} A$

Removing ϵ-productions

- If $\epsilon \in L$, then we can not do much. $S \to \epsilon$ is needed for this.
- For all rules of the type $A \to \epsilon$ and A is not the start symbol, we proceed as follows:
- For occurrence of an A on the right-hand side of a rule, we add a rule with that occurence deleted.
  - For a rule like $R \to uAv$, we add the rule $R \to uv$ (either u or v not ϵ)
  - For a rule like $R \to A$, we add $R \to \epsilon$, unless we removed $R \to \epsilon$ earlier.
  - For a rule with multiple occurences of A, we add one rule for each combination. $R \to uAvAw$ would add $R \to uvAw$, $R \to uAvw$, and $R \to uvw$.

Removing ϵ-productions

- Consider

$$\begin{aligned} S &\rightarrow ASA \mid aB \\ A &\rightarrow B \mid S \\ B &\rightarrow b \mid \epsilon \end{aligned}$$

- Add a new start symbol S_0

$$\begin{aligned} \mathbf{S_0} &\rightarrow \mathbf{S} \\ S &\rightarrow ASA \mid aB \\ A &\rightarrow B \mid S \\ B &\rightarrow b \mid \epsilon \end{aligned}$$

- Remove $B \rightarrow \epsilon$

$$\begin{aligned} \mathbf{S_0} &\rightarrow \mathbf{S} \\ S &\rightarrow ASA \mid aB \mid \mathbf{a} \\ A &\rightarrow B \mid S \mid \epsilon \\ B &\rightarrow b \end{aligned}$$

- Remove $A \rightarrow \epsilon$

$$\begin{aligned} \mathbf{S_0} &\rightarrow \mathbf{S} \\ S &\rightarrow ASA \mid aB \mid a \mid \\ & \quad \mathbf{SA} \mid \mathbf{AS} \mid \mathbf{S} \\ A &\rightarrow B \mid S \\ B &\rightarrow b \end{aligned}$$

Removing Unit Productions

- To remove a unit rule like $A \rightarrow B$,
  - We first add to the grammar a rule $A \rightarrow u$ whenever $B \rightarrow u$ is in the grammar, unless this is a unit rule previously removed.
  - We then delete $A \rightarrow B$, from the grammar.
- We repeat these until we eliminate all unit rules.

Removing Unit Productions

- After ϵ-rule removal

$$\begin{array}{lll} S_0 & \rightarrow & S \\ S & \rightarrow & ASA \mid aB \mid a \mid SA \mid AS \mid S \\ A & \rightarrow & B \mid S \\ B & \rightarrow & b \end{array}$$

- Remove $S \rightarrow S$

$$\begin{array}{lll} S_0 & \rightarrow & S \\ S & \rightarrow & ASA \mid aB \mid a \mid SA \mid AS \\ A & \rightarrow & B \mid S \\ B & \rightarrow & b \end{array}$$

- Remove $S_0 \rightarrow S$

$$\begin{array}{lll} S_0 & \rightarrow & ASA \mid aB \mid a \mid SA \mid AS \\ S & \rightarrow & ASA \mid aB \mid a \mid SA \mid AS \\ A & \rightarrow & B \mid S \\ B & \rightarrow & b \end{array}$$

Removing Unit Productions

- After $S_0 \rightarrow S$ removal

$$\begin{array}{rcl} S_0 & \rightarrow & ASA \mid aB \mid a \mid SA \mid AS \\ S & \rightarrow & ASA \mid aB \mid a \mid SA \mid AS \\ A & \rightarrow & B \mid S \\ B & \rightarrow & b \end{array}$$

- Remove $A \rightarrow B$

$$\begin{array}{rcl} S_0 & \rightarrow & ASA \mid aB \mid a \mid SA \mid AS \\ S & \rightarrow & ASA \mid aB \mid a \mid SA \mid AS \\ A & \rightarrow & b \mid S \\ B & \rightarrow & b \end{array}$$

- Remove $A \rightarrow S$

$$\begin{array}{rcl} S_0 & \rightarrow & ASA \mid aB \mid a \mid SA \mid AS \\ S & \rightarrow & ASA \mid aB \mid a \mid SA \mid AS \\ A & \rightarrow & b \mid ASA \mid aB \mid a \mid SA \mid AS \\ B & \rightarrow & b \end{array}$$

Chomsky Normal Form

- CFGs in certain standard forms are quite useful for some computational problems.

Chomsky Normal Form

A context-free grammar is in Chomsky normal form(CNF) if every rule is either of the form

$$A \rightarrow BC \text{ or } A \rightarrow a$$

where a is a terminal and A, B, C are variables – except B and C may not be the start variable. In addition, we allow the rule $S \rightarrow \epsilon$ if necessary.

Chomsky Normal Form

Theorem

Every context-free language can be generated by a context-free grammar in Chomksy normal form.

Proof Idea

- Add a new start variable and the production $S_0 \rightarrow S$.
- Remove all ϵ-productions
- Remove all unit productions.
- Add new variables and rules so that all rules have the right forms.

Proof

u_i below is either a terminal or a variable.

- Replace each rule like $A \rightarrow u_1 u_2 \cdots u_k$ where $k \geq 3$, with rules $A \rightarrow u_1 A_1$, $A_1 \rightarrow u_2 A_2$, $\cdots$ $A_{k-2} \rightarrow u_{k-1} u_k$
- After this stage, all rules have right-hand side of length either 2 or 1
- For each rule like $A \rightarrow u_1 u_2$ where either or both u_i is a terminal, replace u_i with the new variable U_i and add the rule $U_i \rightarrow u_i$ to the grammar.

Conversion to Chomsky Normal Form

- Grammar after ϵ and unit production removal

$$\begin{array}{rcl} S_0 & \rightarrow & ASA \mid aB \mid a \mid SA \mid AS \\ S & \rightarrow & ASA \mid aB \mid a \mid SA \mid AS \\ A & \rightarrow & b \mid ASA \mid aB \mid a \mid SA \mid AS \\ B & \rightarrow & b \end{array}$$

- Remove $S_0 \rightarrow ASA$ and add $S_0 \rightarrow AA_1$ and $A_1 \rightarrow SA$
- Remove $S \rightarrow ASA$ and add $S \rightarrow AA_1$ ($A_1 \rightarrow SA$ already added)
- Remove $A \rightarrow ASA$ and add $A \rightarrow AA_1$ ($A_1 \rightarrow SA$ already added)
- Replace $S_0 \rightarrow aB$ with $S_0 \rightarrow UB$ and $U \rightarrow a$
- Replace $S \rightarrow aB$ with $S \rightarrow UB$ ($U \rightarrow a$ already added)
- Replace $A \rightarrow aB$ with $A \rightarrow UB$ ($U \rightarrow a$ already added)

- Final grammar in Chomsky normal form

$$
\begin{aligned}
S_0 &\rightarrow AA_1 \mid UB \mid a \mid SA \mid AS \\
S &\rightarrow AA_1 \mid UB \mid a \mid SA \mid AS \\
A &\rightarrow b \mid AA_1 \mid UB \mid a \mid SA \mid AS \\
A_1 &\rightarrow SA \\
U &\rightarrow a \\
B &\rightarrow b
\end{aligned}
$$

Another Example

- Let's convert
  $R = \{S \rightarrow SS, S \rightarrow aSb, S \rightarrow bSa, S \rightarrow \epsilon\}$ to Chomsky Normal Form.

Other Interesting Forms for Grammars

- If all productions of a grammar are like $A \rightarrow bB$ or $A \rightarrow b$ where b is a terminal and B is a variable, then it is called a right-linear grammar.
- If all productions of a grammar are like $A \rightarrow Bb$ or $A \rightarrow b$ where b is a terminal and B is a variable, then it is called a left-linear grammar.
- Right-linear grammars generate regular languages.
- Left-linear grammars generate regular languages.

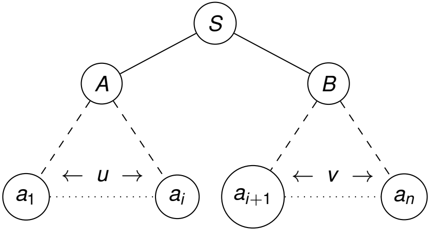
THE RECOGNITION PROBLEM FOR CFL'S

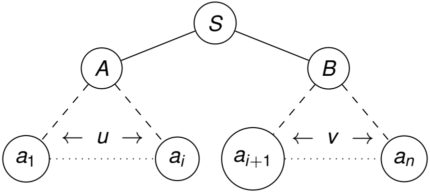
- Given a context-free grammar G and a string $w \in \Sigma^*$ how can we tell if $w \in L(G)$?
- If $w \in L(G)$, what are the possible structures assigned to w by G?
- Different grammars for the same language
  - will answer the first question the same, but
  - will assign possibly different structures to strings in the language.
  - Consider original and Chomsky Normal Form of some example grammars earlier!

The Cocke-Younger-Kasami (CYK) algorithm

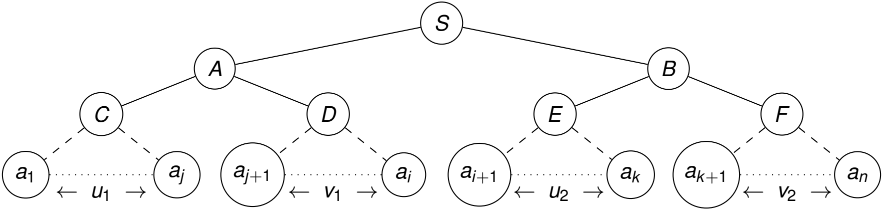
- The CYK parsing algorithm determines if $w \in L(G)$ for a grammar G in Chomsky Normal Form
  - with some extensions, it can also determine possible structures.
  - Assume $w \neq \epsilon$ (if so, check if the grammar has the rule $S \rightarrow \epsilon$)

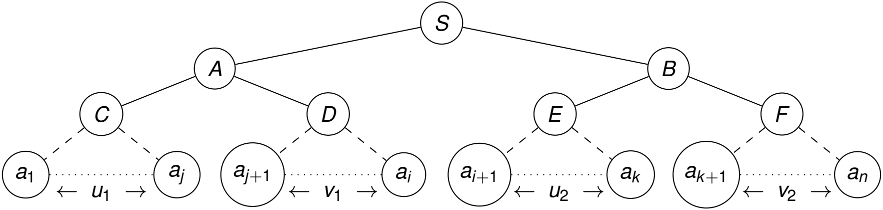
- Consider $w = a_1 a_2 \cdots a_n$, $a_i \in \Sigma$
- Suppose we could cut up the string into two parts $u = a_1 a_2 .. a_i$ and $v = a_{i+1} a_{i+2} \cdots a_n$
- Now suppose $A \stackrel{*}{\Rightarrow} u$ and $B \stackrel{*}{\Rightarrow} v$ and that $S \rightarrow AB$ is a rule.





- Now we apply the same idea to A and B recursively.





- What is the problem here?
- We do not know what i, j and k are!
- No Problem! We can try all possible i's, j's and $k's$.
- Dynamic programming to the rescue.

DIGRESSION - DYNAMIC PROGRAMMING

- An algorithmic paradigm
- Essentially like divide-and-conquer but subproblems overlap!
- Results of subproblem solutions are reusable.
- Subproblem results are computed once and then memoized
- Used in solutions to many problems
  - Length of longest common subsequence
  - Knapsack
  - Optimal matrix chain multiplication
  - Shortest paths in graphs with negative weights (Bellman-Ford Alg.)

(Back to) The CYK Algorithm

- Let $w = a_1 a_2 \cdots a_n$.
- We define
  - $w_{i,j} = a_i \cdots a_j$ (substring between positions i and j)
  - $V_{i,j} = \{A \in V \mid A \stackrel{*}{\Rightarrow} w_{i,j}\} (j \geq i)$ (all variables which derive w_{ij})
- $w \in L(G)$ iff $S \in V_{1,n}$
- How do we compute $V_{i,j} (j \geq i)$?

- How do we compute $V_{i,j}$?
- Observe that $A \in V_{i,i}$ if $A \rightarrow a_i$ is a rule.
  - So V_{ii} can easily be computed for $1 \leq i \leq n$ by an inspection of w and the grammar.
- $A \stackrel{*}{\Rightarrow} w_{ij}$ if
  - There is a production $A \rightarrow BC$, and
  - $B \stackrel{*}{\Rightarrow} w_{i,k}$ and $C \stackrel{*}{\Rightarrow} w_{k+1,j}$ for some k, $i \leq k < j$.
- So

$$V_{i,j} = \bigcup_{i \leq k < j} \{A :| A \rightarrow BC \text{ and } B \in V_{i,k} \text{ and } C \in V_{k+1,j}\}$$

The CYK Algorithm

$$V_{i,j} = \bigcup_{i \le k < j} \{A : A \rightarrow BC \text{ and } B \in V_{i,k} \text{ and } C \in V_{k+1,j}\}$$

- Compute in the following order:

$$\begin{array}{llllllll}
 & \rightarrow & & & & & & \\
\downarrow & V_{1,1} & V_{2,2} & V_{3,3} & \cdots & \cdots & \cdots & V_{n,n} \\
 & V_{1,2} & V_{2,3} & V_{3,4} & \cdots & \cdots & V_{n-1,n} & \\
 & V_{1,3} & V_{2,4} & V_{3,5} & \cdots & V_{n-2,n} & & \\
 & \cdots & & & & & & \\
 & V_{1,n-1} & V_{2,n} & & & & & \\
 & V_{1,n} & & & & & &
\end{array}$$

- For example to compute $V_{2,4}$ one needs $V_{2,2}$ and $V_{3,4}$, and then $V_{2,3}$ and $V_{4,4}$ all of which are computed earlier!

The CYK Algorithm

`for i=1 to n do // Initialization`
    $V_{i,i} = \{A \mid A \rightarrow a \text{ is a rule and } w_{i,i} = a]$
`for j=2 to n do`
  `for i=1 to n-j+1 do`
    `begin`
      $V_{i,j} = \{\}$; `// Set` $V_{i,j}$ `to empty set`
      `for k=1 to j-1 do`
        $V_{i,j} = V_{i,j} \cup \{A :\mid A \rightarrow BC \text{ is a rule and } B \in V_{i,k} \text{ and } C \in V_{k+1,j}\}$

- This algorithm has 3 nested loops with the bound for each being $O(n)$. So the overall time is $O(n^3)$.
- The size of the grammar factors in as a constant factor as it is independent of n – the length of the string.
- Certain special CFGs have subcubic recognition algorithms.

The CYK Algorithm in Action

- Consider the following grammar in CNF

$$\begin{array}{lcl} S & \rightarrow & AB \\ A & \rightarrow & BB \mid a \\ B & \rightarrow & AB \mid b \end{array}$$

- The input string is $w = aabbb$

$i \rightarrow$	1	2	3	4	5
	a	a	b	b	b
	$\{A\}$	$\{A\}$	$\{B\}$	$\{B\}$	$\{B\}$
	$\{\}$	$\{S,B\}$	$\{A\}$	$\{A\}$	
	$\{S,B\}$	$\{A\}$	$\{S,B\}$		
	$\{A\}$	$\{S,B\}$			
	$\{S,B\}$				

- Since $S \in V_{1,5}$, this string is in $L(G)$.

THE CYK ALGORITHM IN ACTION

- Consider the following grammar in CNF

$$\begin{array}{lcl} S & \rightarrow & AB \\ A & \rightarrow & BB \mid a \\ B & \rightarrow & AB \mid b \end{array}$$

- Let us see how we compute $V_{2,4}$
  - We need to look at $V_{2,2}$ and $V_{3,4}$
  - We need to look at $V_{2,3}$ and $V_{4,4}$

$i \rightarrow$	1	2	3	4	5
	a	a	b	b	b
	$\{A\}$	$\{A\}$	$\{B\}$	$\{B\}$	$\{B\}$
	$\{\}$	$\{S, B\}$	$\{A\}$	$\{A\}$	
	$\{S, B\}$	$\{A\}$	$\{S, B\}$		
	$\{A\}$	$\{S, B\}$			
	$\{S, B\}$				