Theory of Formal Languages and Automata
Lecture 11

Mahdi Dolati

Sharif University of Technology

Fall 2023

November 6, 2023

Membership

- If $w \in L(G)$, then the sequence

$$S \Rightarrow w_1 \Rightarrow \cdots \Rightarrow w_n \Rightarrow w,$$

  is a **derivation** of the sentence w.
- The strings $S, w_1, \ldots, w_n$, which contain variables as well as terminals, are called **sentential forms** of the derivation.

Membership

Example

Consider the grammar $G = (\{S\}, \{a, b\}, S, P)$ with P given by:

$$S \to aSb,$$
$$S \to \varepsilon.$$

Then,

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb,$$

so we can write

$$S \overset{*}{\Rightarrow} aabb.$$

The string aabb is a sentence in the language generated by G, while aaSbb is a sentential form.

Membership

- Given a grammar G and a string w:
  - Whether or not w is in $L(G)$,
    - A membership algorithm,
  - If $w \in L(G)$, find a derivation of w,
    - Parsing: find a sequence of rules by which $w \in L(G)$ is derived,

Exhaustive

- Exhaustive search parsing or brute force parsing:
  - Start by all rules of the form

$$S \to x,$$

  - Substitute the leftmost variable of every x using all applicable rules,
- If $w \in L(G)$, then it has a derivation of finite length,
- Thus the method will give a leftmost derivation.

Exhaustive

Example

Consider string $w = aabb$ and the grammar:

$$S \to SS \mid aSb \mid bSa \mid \varepsilon$$

Round one of the brute force parsing algorithm:

$$S \Rightarrow SS,$$
$$S \Rightarrow aSb,$$
$$S \Rightarrow bSa,$$
$$S \Rightarrow \varepsilon,$$

Last two sentential forms are not useful here,

Example (Cont.)

Consider string $w = aabb$ and the grammar: $S \to SS \mid aSb \mid bSa \mid \varepsilon$
Remove pointless ones and apply the second step of substitutions:

$$\begin{array}{lcl} & & S \Rightarrow SS \Rightarrow SSS, \\ & & S \Rightarrow SS \Rightarrow aSbS, \\ & & S \Rightarrow SS \Rightarrow bSaS, \\ S \Rightarrow SS, & \Rightarrow & S \Rightarrow SS \Rightarrow S, \\ S \Rightarrow aSb, & & S \Rightarrow aSb \Rightarrow aSSb, \\ & & S \Rightarrow aSb \Rightarrow aaSbb, \\ & & S \Rightarrow aSb \Rightarrow abSab, \\ & & S \Rightarrow aSb \Rightarrow ab, \end{array}$$

Next round we get to w.

Exhaustive

- Exhaustive search parsing or brute force parsing:
  - Is inefficient,
  - May never stop for a string not in the language.
    - E.g., $w = abb$,
  - The problem of nontermination comes from rules:
    - $A \to \varepsilon$
    - $A \to B$

Exhaustive

Example

A grammar for the language of the previous grammar (without the empty string) without unit and ε rules:

$$S \to SS \mid aSb \mid bSa \mid ab \mid ba$$

Given any $w \in \{a, b\}^+$, the exhaustive search parsing method will always terminate in no more than $|w|$ rounds. This is clear because the length of the sentential form grows by at least one symbol in each round. After $|w|$ rounds we have either produced a parsing or we know that $w \notin L(G)$.

Exhaustive

Theorem

Suppose that $G = (V, T, S, P)$ is a CFG that does not have any rules of the form $A \to \varepsilon$ or $A \to B$, where $A, B \in V$. Then the exhaustive search parsing method can be made into an algorithm that, for any $w \in \Sigma^*$, either produces a parsing of w or tells us that no parsing is possible.

Proof.

- Each step in the derivation increases either (or both) of length or the number of terminals in each sentential form.
- Neither the length of a sentential form nor the number of terminal symbols can exceed $|w|$.
- Thus, a derivation cannot involve more than $2|w|$ rounds.

□

Exhaustive

Number of sentential forms (restrict ourselves to leftmost derivations):

- No more than $|P|$ sentential forms after one round,
- No more than $|P|^2$ sentential forms after one round,
- We showed there are at most $2|w|$ rounds,
- Thus, number of sentential forms does not exceeds:

$$
\begin{aligned}
M &= |P| + |P|^2 + \cdots + |P|^{2|w|} \\
&= O(|P|^{2|w|+1}).
\end{aligned}
$$

- May grow exponentially with the length of the string,
- Practical observation shows that exhaustive search parsing is very inefficient in most cases.

CFL Parsing

Theorem

For every context-free grammar there exists an algorithm that parses any $w \in L(G)$ in a number of steps proportional to $|w|^3$.

- Still inefficient,
  - need an excessive amount of time to analyze even a moderately long program.
- We like a linear time parsing algorithm:
  - to takes time proportional to the length of the string.
- We do not know any linear time parsing methods for CFLs in general,
- There are linear time parsing algorithms for restricted, but important, special cases.

Simple Grammars

Definition

A CFG $G = (V, T, S, P)$ is said to be a simple grammar or s-grammar if all its productions are of the form

$$A \to ax,$$

where $A \in V$, $a \in T$, $x \in V^*$, and any pair (A, a) occurs at most once in P.

If G is an s-grammar, then any string $w \in L(G)$ can be parsed with an effort proportional to $|w|$.

Simple Grammars

Example

The grammar

$$S \to aS \mid bSS \mid c$$

is an s-grammar. The grammar

$$S \to aS \mid bSS \mid aSS \mid c$$

is not an s-grammar because the pair (S, a) occurs in the two productions $S \to aS$ and $S \to aSS$.

Simple Grammars

Exhaustive search method for s-grammars:

- String $w = a_1 a_2 \dots a_n$,
- There is one choice for $S \to a_1 x$, so the derivations starts with

$$S \Rightarrow a_1 A_1 \dots A_m.$$

- There is one choice for $A_1 \to a_2 x^{'}$, so the derivation continues with

$$S \Rightarrow a_1 A_1 \dots A_m \Rightarrow a_1 a_2 B_1 \dots A_2 \dots A_m.$$

- Each step produces one terminal symbol,
- The whole process must be completed in no more than $|w|$ steps.

Theorem

For every context-free grammar there exists an algorithm that parses any $w \in L(G)$ in a number of steps proportional to $|w|^3$.

- CYK algorithm,
  - Cocke-Younger-Kasami Algorithm,
- Works only if the grammar is in Chomsky normal form,
- Breaks one problem into a sequence of smaller ones.

Definition (Chomsky normal form (CNF))

A CFG is in CNF if every rule is of the form:

$$A \to BC$$
$$A \to a$$

- a is any terminal,
- A, B and C are variables,
- B and C may not be the start variable.

Following rule is also valid:

$$S \to \varepsilon$$

CYK

- Assume a grammar $G = (V, T, S, P)$ in Chomsky normal form a string

$$w = a_1 a_2 \dots a_n.$$

- Define substrings

$$w_{ij} = a_i \dots a_j,$$

and subsets of V

$$V_{ij} = \{A \in V \ : \ A \stackrel{*}{\Rightarrow} w_{ij}\}$$

- $w \in L(G)$ if and only if $S \in V_{1n}$.

CYK

- $A \in V_{ii}$ if and only if there is $A \to a_i$ in G,
- Thus, V_{ii} can be computed for all $1 \le i \le n$ by inspecting w and rules of the grammar.
- We can combine V_{ii} can compute V_{ij} for $j > i$ that derives w_{ij}:

$$V_{ij} = \bigcup_{k \in \{i, i+1, \ldots, j-1\}} \{A \ : \ A \to BC, \text{ with } B \in V_{ik}, C \in V_{k+1,j}\}.$$

  - Compute $V_{11}, V_{22}, \ldots, V_{nn}$,
  - Compute $V_{12}, V_{23}, \ldots, V_{n-1,n}$,
  - Compute $V_{13}, V_{24}, \ldots, V_{n-2,n}$,
  - ...
- Keep track of how the elements of V_{ij} are derived, it can be converted into a parsing method.

Example

The string $w = aabbb$ and the grammar:

$$\begin{aligned} S &\to AB, \\ A &\to BB \mid a, \\ B &\to AB \mid b, \end{aligned}$$

$w_{11} = a$	$w_{22} = a$	$w_{33} = b$	$w_{44} = b$	$w_{55} = b$
$V_{11} = \{A\}$	$V_{22} = \{A\}$	$V_{33} = \{B\}$	$V_{44} = \{B\}$	$V_{44} = \{B\}$
$w_{12} = aa$	$w_{23} = ab$	$w_{34} = bb$	$w_{45} = bb$	
$V_{12} = \emptyset$	$V_{23} = \{S, B\}$	$V_{34} = \{A\}$	$V_{34} = \{A\}$	
$w_{13} = aab$	$w_{24} = abb$	$w_{35} = bbb$		
$V_{13} = \{S, B\}$	$V_{24} = \{A\}$	$V_{35} = \{S, B\}$		
$w_{14} = aabb$	$w_{25} = abbb$			
$V_{14} = \{A\}$	$V_{25} = \{S, B\}$			
$w_{15} = aabbb$				
$V_{15} = \{S, B\}$				

CYK

To see that the CYK membership algorithm requires $O(n^3)$, notice that exactly $\frac{n(n+1)}{2}$ sets of V_{ij} have to be computed. Each involves the evaluation of at most n terms in

$$V_{ij} = \bigcup_{k \in \{i, i+1, \ldots, j-1\}} \{A \ : \ A \to BC, \text{ with } B \in V_{ik}, C \in V_{k+1,j}\}.$$

so the claimed result follows.