Parsing - Introduction

Dr G Sudha Sadasivam

Agenda

- Role of parsers
- Categorisation
- Error recovery
- CFG
- Derivations
- Parse trees
- ambiguity

Role of parser

- syntax analyzer verifies if the tokens are properly sequenced (with the grammar of the language).
- report syntactical errors
- Error recovery

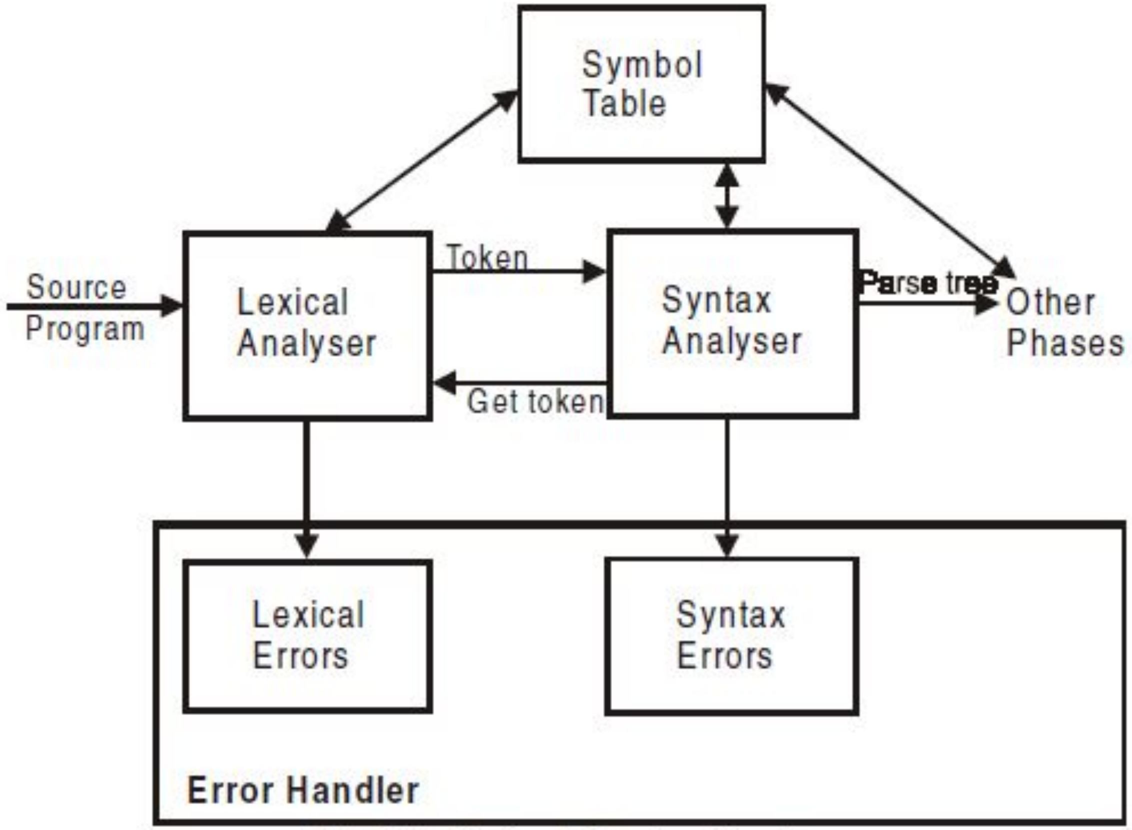


Fig.3.1 Role of Syntax Analyser

Categorisation

Universal parsers: Cocke-Kasami-Younger (CKY) - Chomsky Normal Form (CNF) of the CFG.

- Inefficient- not used in commercial compilers.

Top down parsers, build parse trees from the root node to leaves.

- satisfy LL grammars - scan left to right and use Left-most derivation.
- Top down parsers categorized as that which use / not use backtracking
- backtracking – if erroneous expansions then rollback to initial position
- Non-backtracking – using current NT to be expanded and next input symbol the parser decides next production - Predictive parsing
  - Procedure driven – recursive descent
  - Table driven

- **Bottom up parsers** build parse trees starting from the leaves and work up to the root node.(input is reduced to the start symbol)
  - satisfy **LR grammars** (scan left to right & right-most derivation.
- **Shift reduce parsers-** parsers are implemented using stacks
- Classified as operator precedence parsers and LR parsers.
- In **operator precedence** parsers, the choice of whether to shift the next symbol or reduce the processed input is decided using a precedence relation table.
- LR parsers use parsing table derived out of the grammar for deciding whether to shift the next input symbol or reduce the processed input.
  - **Simple LR (SLR), Canonical LR (CLR) and Look Ahead LR (LALR) parsers**

Syntax error

- No legal move from its current configuration determined by its state, stack and current input symbol.
- Error recovery: The parser should locate the position of error, correct the error, revise its configuration and resume parsing.
- Examples of syntactic phase errors are errors in structure, missing operators, unbalanced parenthesis.
- **Missing parenthesis:** a=(b+c)+d+e).
- **Extraneous-insertion error:** for(i=0,;i!=10;i++) , an extra ',' is inserted.
- **Replacement error:**

  i=0:

  j=9;
- **Transcription errors:** misspelled keywords , eg**.** "man" is typed instead of "main"
- **Insertion** error: When extra blank are inserted

  /*---------* /

Error recovery

- Error correction involves a minimum number of insertions, deletions and corrections to transform an incorrect syntactic structure to a correct one – minimum hamming distance
- **Panic Mode**: discard symbols from point of error till the synch token like (; or })
- **Exhaustive method**: examine each possibility, find the cause, and then decide recovery (error recovery routines for each type of error) – table driven parsers
- **Systematic methods for error recovery:**

  i. Suspend normal parsing on an error configuration,

  ii. Change the error configuration by modifying either the stack contents or the input buffer, to arrive at a different configuration,

  iii. Resume normal parsing from the new configuration.

- **Time of detection** – valid prefix property
- **Phrase-level error recovery**: On discovering an error, the parser performs local correction of the remaining input. It replaces the prefix of the remaining input by some string that allows the parser to continue.
- **Error Productions**: to augment the grammar for the language with error messages for erroneous constructs
- **Global Correction**: Compiler makes as few changes as possible in processing the input string - minimum distance error correction

CFGs: Definition

$$G = (V, \Sigma, P, S)$$

V	= variables	a finite set
Σ	= alphabet or terminals	a finite set
P	= productions	a finite set
S	= start variable	$S \in V$

Productions' form, where $A \in V$, $\alpha \in (V \cup \Sigma)^*$:

- $A \rightarrow \alpha$

Definition. v is **derivable** from u, written u $\Rightarrow^*$ v, if:
There is a chain of one-derivations of the form:

$$u \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \ldots \Rightarrow v$$

S $\rightarrow$ A | A B
A $\rightarrow$ ε | **a** | A **b** | A A
B $\rightarrow$ **b** | **b c** | B **c** | **b** B

Sample derivations:
S ⇒ AB ⇒ AAB ⇒ **a**AB ⇒ **aa**B ⇒ **aab**B ⇒ **aabb**
S ⇒ AB ⇒ A**b**B ⇒ A**bb** ⇒ AA**bb** ⇒ A**abb** ⇒ **aabb**

Definition CFL. Given a context-free grammar G = (∑, NT, R, S), the **language generated** or derived from G is the set:

$$L(G) = \{w : S \Rightarrow^* w\}$$

Definition. A language L is context-free if there is a context-free grammar G = (∑, NT, R, S), such that L is generated from G

Example

1) L:{$\mathbf{a}^n\ \mathbf{b}^n$ | n≥0}

P: S → ε | **a** S **b**

G = ({S}, {**a**,**b**}, {S → ε, S → **a** S **b**}, S)

2) Balanced Parenthesis eg (), (())()

P → ε | **(** P **)** | P P

P□ PP □(P)P□(())P□(())()

3) {$\mathbf{a}^m\ \mathbf{b}^n\ \mathbf{c}^{m+n}$ | m,n≥0}

$\mathbf{a}^m\ \mathbf{b}^n\ \mathbf{c}^{m+n}$ can ve rewritten as $\mathbf{a}^m\ \mathbf{b}^n\ \mathbf{c}^n\ \mathbf{c}^{\mathbf{m}}$

S□ aSc | S’

S’ □ bS’c|ϵ

Parse Tree

A parse tree of a derivation is a tree in which:

- Each internal node is labeled with a non-terminal

- If a rule A □ $A_1 A_2 \ldots A_n$ occurs in the derivation then A is a parent node of nodes labeled $A_1, A_2, \ldots, A_n$

$$\begin{tikzcd}
 & & S \arrow[dl] \arrow[d] & & \\
 & a & S \arrow[d] \arrow[dr] & & \\
 & & a & S \arrow[d] \arrow[dr] & \\
 & & & S \arrow[d] & b \\
 & & & e &
\end{tikzcd}$$

Parse Trees

S → A | A B
A → ε | **a** | A **b** | A A
B → **b** | **b c** | B **c** | **b** B

Sample derivations:

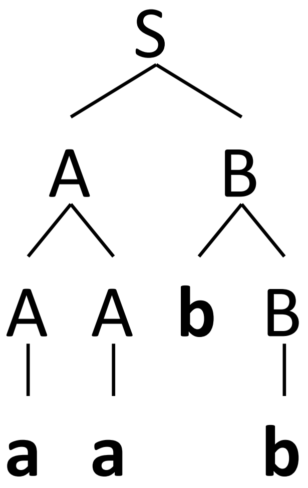
S ⇒ AB ⇒ AAB ⇒ **a**AB ⇒ **aa**B ⇒ **aab**B ⇒ **aabb**

S ⇒ AB ⇒ A**b**B ⇒ A**bb** ⇒ AA**bb** ⇒ A**abb** ⇒ **aabb**

These two derivations use same productions, but in different orders.

This ordering difference is often uninteresting.

Derivation trees give way to abstract away ordering differences.



Root label = start node.

Each interior label = variable.

Each parent/child relation = derivation step.

Each leaf label = terminal or ε.

All leaf labels together = derived string = *yield*.

Left most and Right most derivations

Definition. A **left-most derivation** of a sentential form is one in which rules transforming the left-most nonterminal are always applied

S ⇒ AB ⇒ AAB ⇒ **a**AB ⇒ **aa**B ⇒ **aab**B ⇒ **aabb**

S → A | A B
A → ε | **a** | A **b** | A A
B → **b** | **b c** | B **c** | **b** B

Definition. A **right-most derivation** of a sentential form is one in which rules transforming the right-most nonterminal are always applied

S ⇒ AB ⇒ A**b**B ⇒ A**bb** ⇒ AA**bb** ⇒ A**abb** ⇒ **aabb**

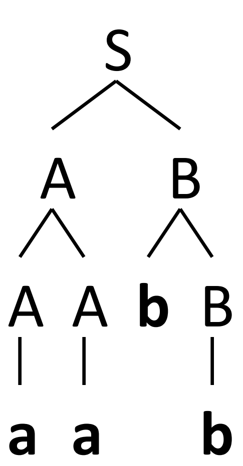
Derivation Trees

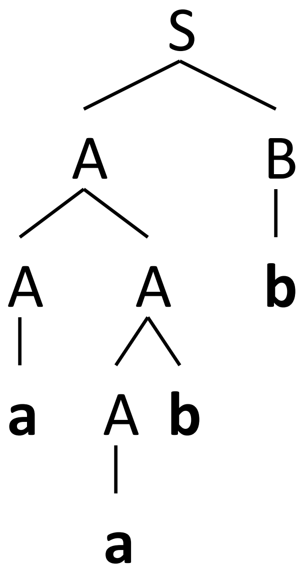
S $\rightarrow$ A | A B

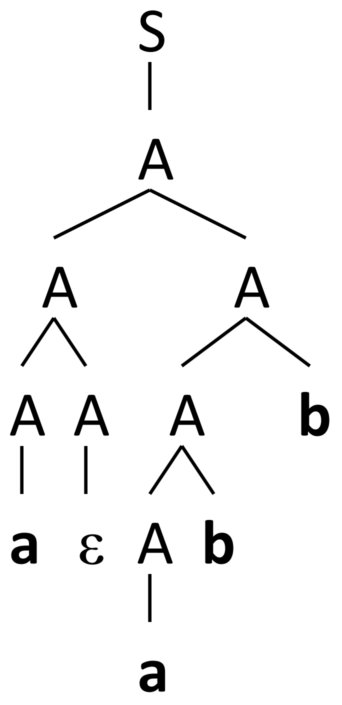
A $\rightarrow$ ε | **a** | A **b** | A A

B $\rightarrow$ **b** | **b c** | B **c** | **b** B

w = **aabb**







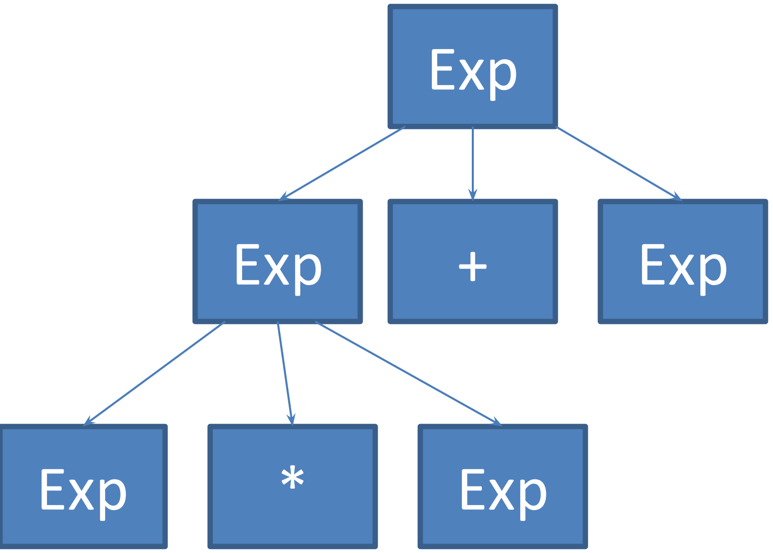
Ambiguity

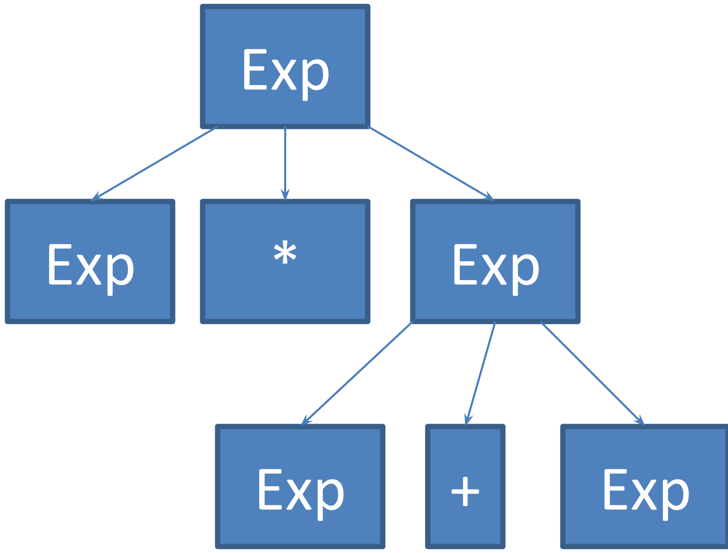
CFG *ambiguous* ⇔ any of following equivalent statements:

- ∃ string w with multiple derivation trees.
- ∃ string w with multiple leftmost derivations.
- ∃ string w with multiple rightmost derivations.

Defining ambiguity of <u>grammar</u>, not language.

Ambiguity: Exp*Exp+Exp





Disambiguation: Example 1

$$
\begin{array}{lcl}
\text{Exp} & \rightarrow & \mathbf{n} \\
 & | & \text{Exp} + \text{Exp} \\
 & | & \text{Exp} \times \text{Exp}
\end{array}
$$

What is an equivalent unambiguous grammar?

$$
\begin{array}{lcl}
\text{Exp} & \rightarrow & \text{Term} \\
 & | & \text{Term} + \text{Exp} \\
\text{Term} & \rightarrow & \mathbf{n} \\
 & | & \mathbf{n} \times \text{Term}
\end{array}
$$

Uses

- operator precedence
- left-associativity

Home exercises

- Write CFG for even length and odd length palindromes
- Write CFG for arithmetic expressions
- Consider the following grammar

  S□ aS | Sa | ϵ

- Construct parse tree for a^4
- Write the left most derivation for a^4
- Write the right most derivation for a^4
- Is the grammar ambiguous. Justify your answer