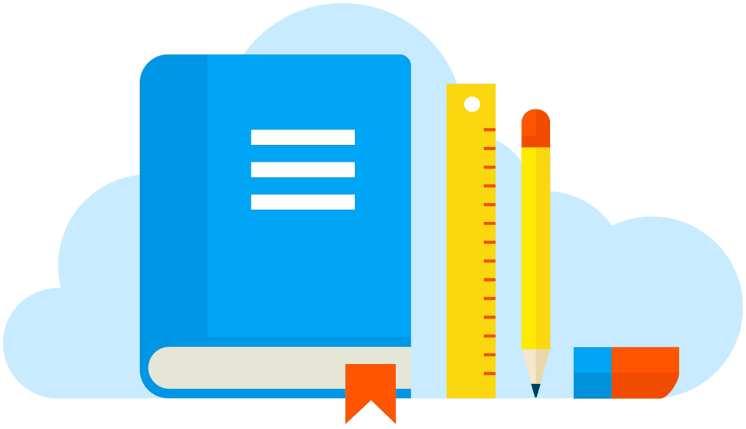


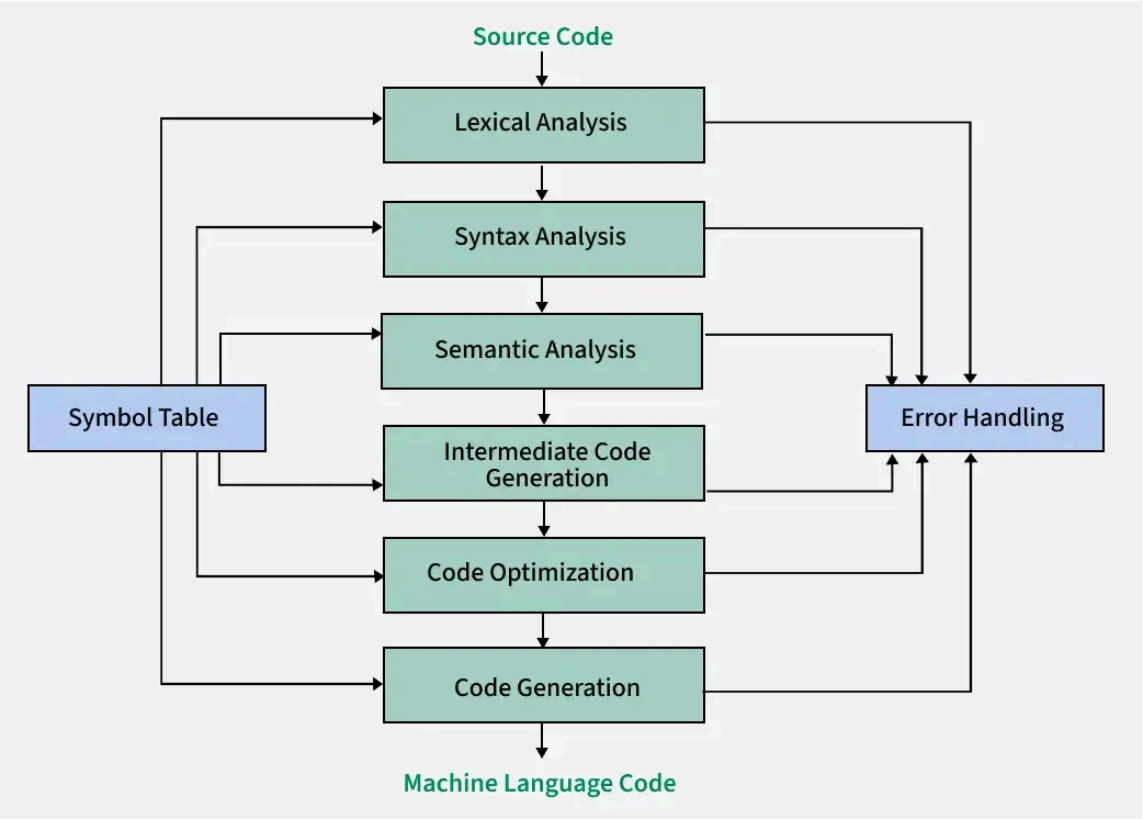
Lecture Notes

Date: 10/08/2025

LR(0) Parser

Prerequisites :

Phases of Compiler



1. What is a Parser ?

A **parser** is a component of the compiler that checks whether the code follows the **grammar rules** of the programming language.
It takes **tokens** from the lexical analyzer and builds a **parse tree** to represent the program's structure.

- Example: For `a + b`, the parser checks if `+` is allowed between two identifiers.

2. Why Do We Need It in Compilers?

The parser is needed to:

- **Validate syntax** of the code (detect errors).
- **Build structure** (parse tree) for further analysis.
- Provide input for later stages like **semantic analysis** and **code generation**.

- Without a parser, the compiler wouldn't understand the code's structure.

3. Types of Parsers

Parsers are mainly of two types :

Type	Direction	Builds From	Example
Top-down	Left to right	Root to leaves	LL(1), Recursive Descent
Bottom-up	Left to right	Leaves to root	LR(0), LR(1), SLR

- Top-down: predicts what to parse.
- Bottom-up: reduces input to grammar start symbol.

4. What is an LR(0) Parser ?

The **LR parser** is an efficient **bottom-up syntax** analysis technique that can be used for a large class of **context-free grammar**. This technique is also called **LR(0) parsing**.

- **L** stands for the **left to right scanning.**
- **R** stands for **rightmost derivation in reverse.**
- **0** stands for **no. of input symbols** of lookahead.

5. Augmented Grammar

An augmented grammar is a modified version of a grammar where we add a new start symbol and rule to help with parsing.

- **Add a new start symbol (S')**
- **Create a new rule (S' → S)**, where S is the original start symbol.
- **Keep all other rules the same.**

Original Grammar :

S → aA

A → b

Augmented Grammar :

S' → S

S → aA

A → b

Q. Construct an LR parsing table for the given context-free grammar:

E → T+E / T

T → id

Steps to follow :

- **STEP 1** - Find the augmented grammar
- **STEP 2** - Find LR(0) collection of items
- **STEP 3** - defining 2 functions: goto[list of non-terminals] and action[list of terminals] in the parsing table

Special Note :

What is the Dot Operator in LR Parsing?

In LR parsing, the **dot (·) operator** is used to **indicate how much of a production has been seen (parsed)** so far and what remains to be parsed.

LR(0) Parsing Table :

State	id	+	$	E	T
I_0					
I_1					
I_2					
I_3					
I_4					
I_5					