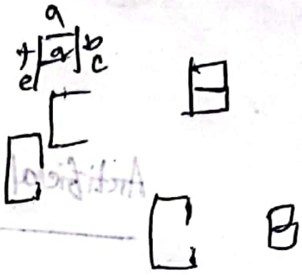
20.08.23
arisha 1823

Compiler design

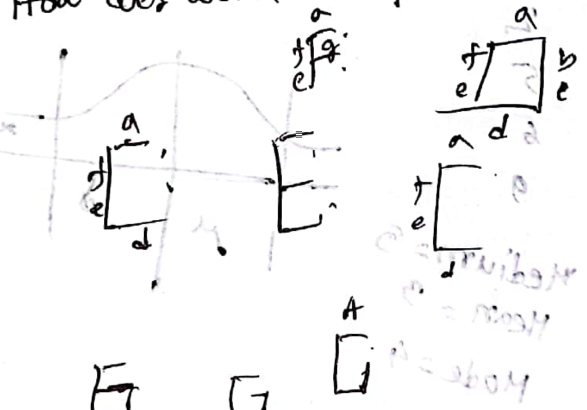
Introduction to compilers — Sunday - 10 am

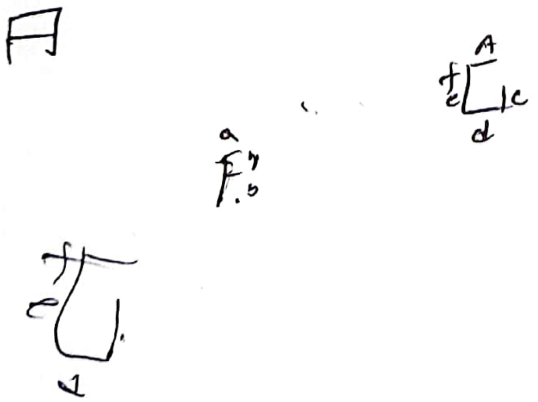
→ Introduction
→ programming languages
→ What are the problems here?

DM167



→ Compiler
Interpreter ~~and co~~ Vs compiler
Hybrid compiler
program → Machine code
preprocessors   How does work a compiler?
Assemblers
Linkers
Loaders





Compiler design and automata theory

21.08.23

The structure of a compiler

Wednesday CT-01

9:20 am

Slide-01 8,02

- Phases of compilation
- Analysis- synthesis Phases
- Phases of a compiler
- Lexical analysis / scanning
- Parsing / Syntax analysis
- Semantic Ananlysis
- Intermediate Code Generation
- code optimization
- code generation

8920471024

26969

Slide-03

Compiler Design & Automata Theory

Date- 24.08.23

Evolution

- Machine language
- Assembly language.
- Macro instructions in assembly language
- Classification ~~of~~ by generations
- Impact of compilers

Compiler Design & Automata Theory

Date- 26.08.23

Lexical Analyzer

[illegible]

- Lexical Analyzer
  - scanning
  - parser
  - Tokens, patterns and Lexems
  - Example of Tokens

Attributes for tokens (self study)

Ex - 3.2

Compiler Design & Automata Theory Date – 28.08.23

Lexical Analysis

- common Token Types
- Attributes for tokens
- Example – 3.2

Error Recovery / Handling

- Types of errores
- More Lexical error examples
- Error Recovery/handling
- Advantages of error Recovery in Lexical Analysis phase
- Disadvantages of error Recovery in Lexical Analysis phase

Input Buffering

- Input Buffering
- What is input buffering? → Low Level Language to High Level Language
- Advantage
- Disadvantage
  - How it works?
- Buffer Pairs
- Sentinels
- can we run out of Buffer space?

Compiler Design & Automata Theory   Date - 29.08.23

Notation -

→ Alphabet -

→ String -

→ Language -

— Term for parts of strings

→ concatenation

→ Exponentiation

Language operators.

- Union ($L \cup M$)
- concatenation (LM)
- closure
  - → Kleene closure → (L^*)
  - → positive closure → (L^+)

~~Lin~~ formal definition

Fig - 3.6 Definitions of operators

Ex - 3.3

. Regular expressions

Ex - 3.4

Algebraic laws for regular expression → Fig - 3.7

3.3.4 Regular Definitions

Extensions of regular expression

Exercise 3.3.2

Ex - 3.3.4

CSE - 3105

Date- 04.09.23

Compiler Design & Automata Theory

→ Token recognition

→ Transition Diagram

→ Example

→ Keywords / ID ~~reg~~ recognition

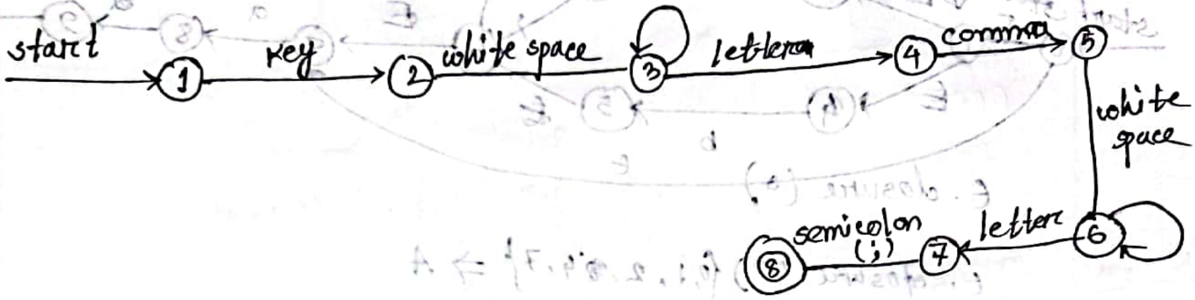
→ Unsigned number & white ~~recognition~~ space recognition → Fig - 3.17

peripheral and Interfacing Date - 05.09.23

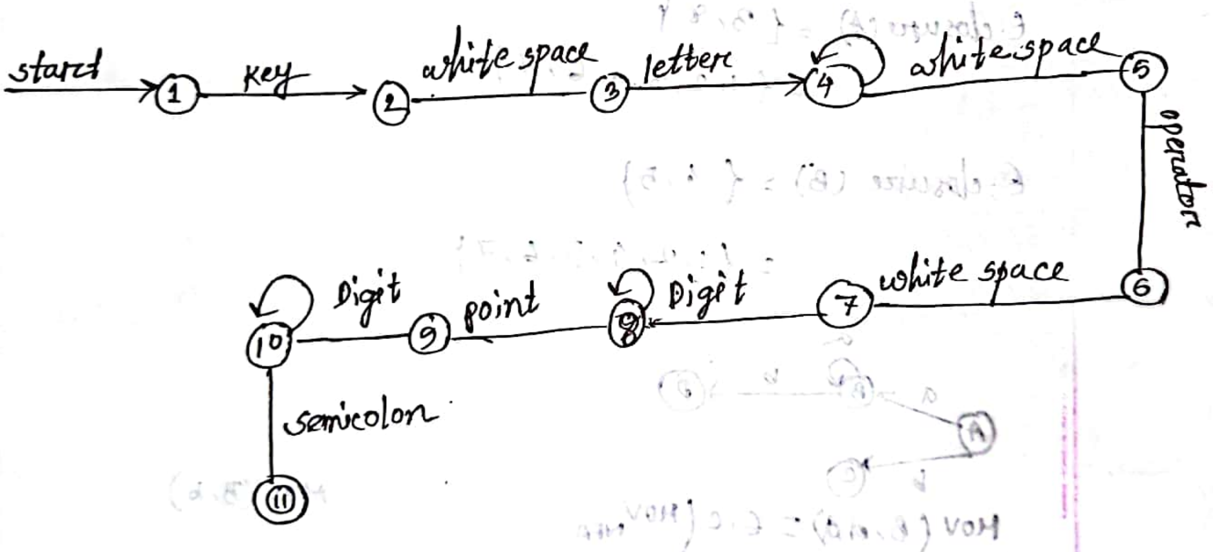
ID: 190135
CSE-3206

Transition Diagram

① int ~~xyz~~ x, yz;



(ii) float abc = 5.5;



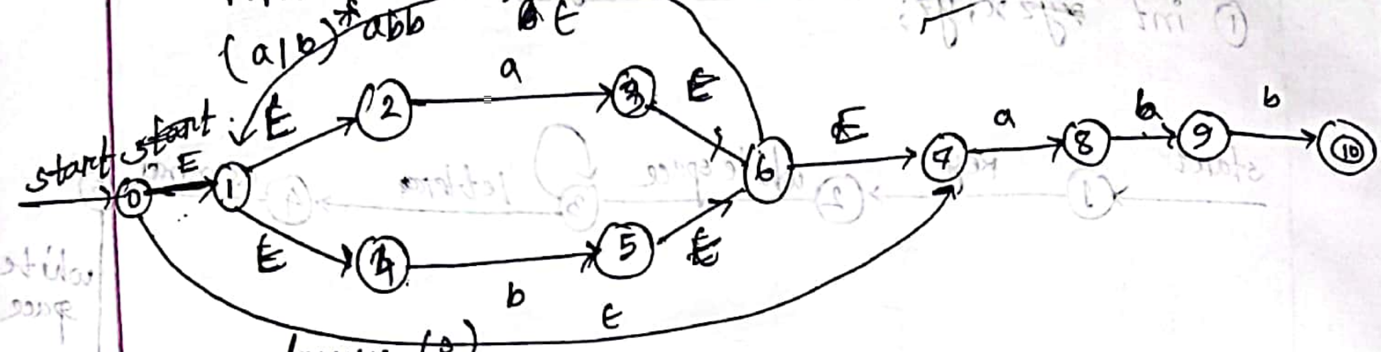
Compiler Design & Automata Theory

Date – 12/09/23

conversion NFA to DFA

NFA

(a|b)*abb



E. closure (0)

E. closure (0) {0,1,2,4,7} ⇒ A

MOV (A, b)

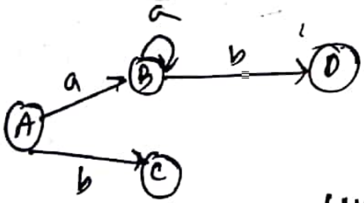
MOV (A, a)

E.closure (A) = {3, 8}

= {1, 2, 3, 4, 6, 7}

E.closure (B) = {4, 5}

= {1, 2, 4, 5, 6, 7}



MOV (B, b)

MOV (B, A) = E.C (MOV$_{NFA}$

~~MoV (C, a)~~

MoV (C, a) . MoV (C, b)

compiler Design

Date- 16/09/23

Syntax Analysis (slide)

Regular expression
NFA → DFA
DFA
→ CT-02

CT-3

Error handling
common syntax errors
Error recovery strategies
Syntax Errors
Semantic Errors

Parser
The role of parser
Grammer
Benefits of grammer
Example of grammer
context-free grammer
Example - 4.5

Slide - Syntax Analyser

Compiler Design & Automata Theory

Date - 26/09/23

→ Derivations

→ Derivation types

→ Derivation in detail

→ Example

→ Excercise - 4.2.1 :

$S \rightarrow SS+ \mid SS* \mid a$ and the string aa+a*

a:

~~$S \Rightarrow S+S$~~

$$
\begin{aligned}
S &\Rightarrow SS* \\
&\Rightarrow SS+S* \\
&\Rightarrow aS+S* \\
&\Rightarrow aa+S* \\
&\Rightarrow aa+a*
\end{aligned}
$$

Buzzer,
transmitor,
resistor,
copper wire.

→ Ambiguity

→ Example of ambiguity

→ Ambiguity again - dangling Else

→ How to fix dangling else problem

Nanotechnology, Peripheral and interfacing Date – 01/10/23

SCSI - introduction

~~SCSI sy~~

SCSI system

SCSI

SCSI controller

Termination

Compiler Design (Slide – Syntax Analysis)

Date – 02/10/23

→ Left Recursion Example

→ Left Factoring

→ Top Down Parsing

→ Top down Parsing Example

→

$E \rightarrow TE'$

$E' \rightarrow TE' | \epsilon$

$T \rightarrow FT'$

$T' \rightarrow \cdot FT' | \epsilon$

$F' \rightarrow (E) | id$

→ First & ~~Flow~~ Follow

→ How to compute First (x)

→ Example of First set generation

→ How to compute Follow (x)

Left (2) → H.W.

CS CamScanner

Compiler Design & Automata Theory Date - 7/10/23

Example (4.32) (Slide - syntax Analysis)

$E \rightarrow T E'$

$E' \rightarrow + T E' \mid \epsilon$

$T \rightarrow F T'$

$T' \rightarrow * F T' \mid \epsilon$

$F \rightarrow (E) \mid id$

* —— *

(Slide - syntax Analysis)

| Example (4.30) of follow set generation

In previous class we learnt:

- Various kind of parsing
- problem according to grammer
- Recursive grammer
- First follow to parse table
- Predictive parser
- constructing a predictive parsing table

  Method : $A \rightarrow \alpha$ rule
- Example 4.32
- Fig 4.2

Compiler Design Date - 09/10/23

Slide - Bottom up parsing

→ Bottom up parsing

→ Top down parsing

→ 4.5.2 Handle pruning

→ 4.5.3 shift reduce parsing

→ conflict in shift reduce parsing

→ 4.6 Introduction to LR parsing : simple

→ 4.6.1 why LR parses ?

→ 4.6.2 Items and the LR(0) Automation

→ Example 4.40

Algorithm : C 4.5 : Introduction

Algorithm : C 4.5 : Gain Ratio

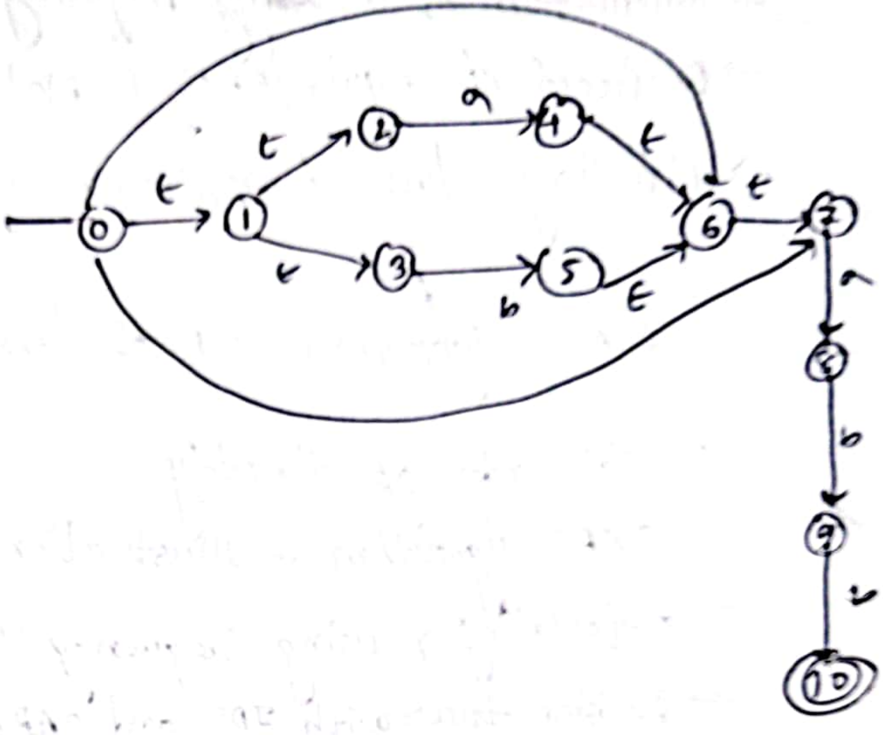
physical Interpretation

compiler

Date - 28/10/23

① → Regular Expression

$(a|b)^* abb$



② NFA → DFA

closure → ε(0)

③ parsing

④ Transition Diagram
Lexem ←

⑤ DFA

⑥ Syntax Analyzer
- → Notational convention
- → Formal Grammer
- → Parse Trees

Compiler

- Parsing, FIRST(), FOLLOW()

Table ↓ (4)

Parsing ↓ (3)

FIRST() ↓ (3)

FOLLOW() ↓ (3)