AIN SHAMS UNIVERSITY

FACULTY OF ENGINEERING

i-CREDIT HOURS ENGINEERING PROGRAMS

CSE 247 – DESIGN OF COMPILERS

MILESTONE 1

TINY LANGUAGE

SUBMITTED BY:

Nouran Elsayed	23P0006
Reetaj Ahmed	23P0114
Carol Kamal	23P0328
Adham Walid	23P0024
Hassan Ismail	23P0152
Eslam Fawzy	23P0052

Contents

1.0 INTRODUCTION

The Tiny Language is a simplified programming language designed to facilitate the understanding and application of compiler design concepts. In this phase of the project, our focus is to define the lexical structure of Tiny Language by identifying and formalizing a set of regular expressions (RE rules) and constructing their corresponding Deterministic Finite Automata (DFAs).

To streamline the lexical analysis process, we carefully selected eight fundamental regular expressions (REs) that serve as the core building blocks of the language. These eight REs are sufficient to derive and construct the remaining language constructs, making them essential for the scanner. The remaining patterns (function calls, assignments, and control structures) are compositions of these eight primary components. This modular approach not only simplifies DFA construction but also promotes reusability and extensibility in later stages of the compiler design.

2.0 REGULAR EXPRESSIONS AND DFA DIAGRAMS

The following are the eight essential regular expressions used to tokenize the Tiny Language:

2.1 Arithmetic Operator

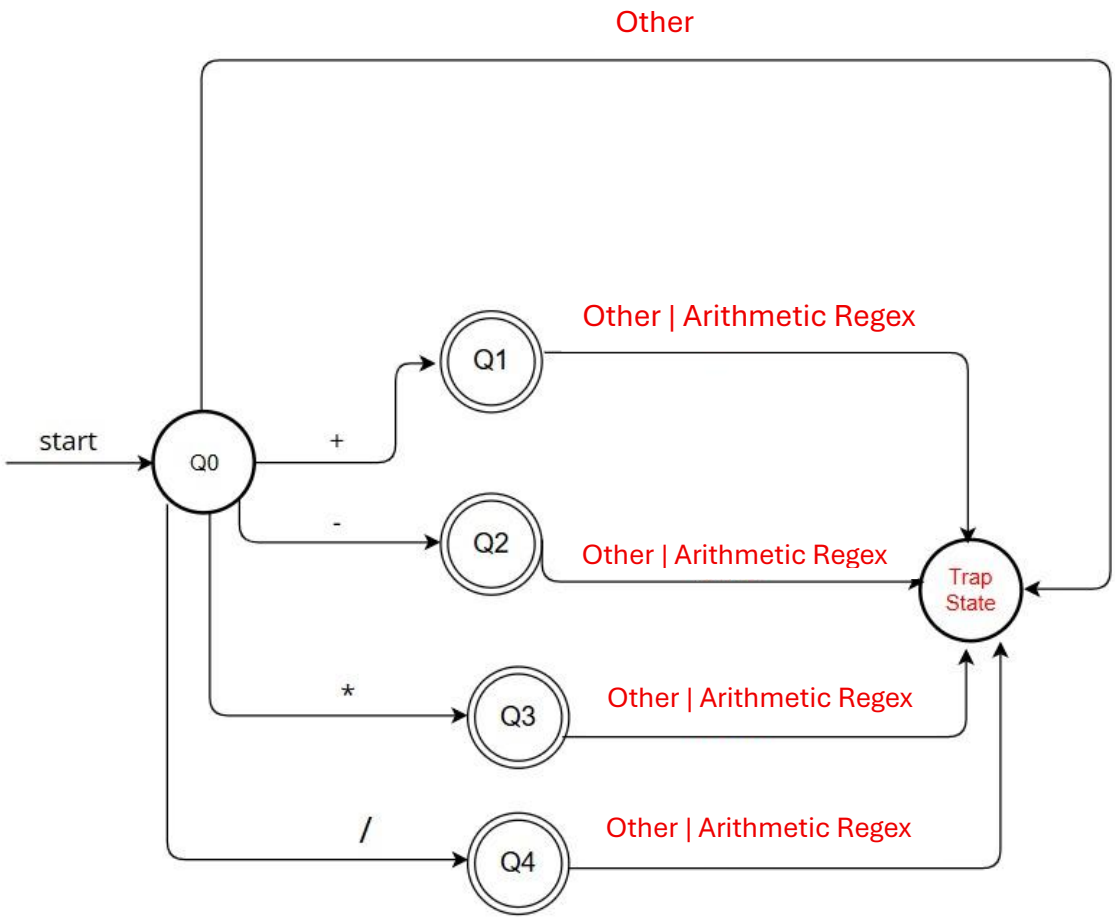
2.1.1 Regular Expression

Arithmetic Regex: -|+|/|*

Description: Matches any arithmetic operator: +, -, *, or /

Other: [A-Z] | [a-z] | [0-9] | : | < | > | = | & | | | (|) | { | } | ; | , | . | "

2.1.2 DFA Diagram



2.2 Condition Operator

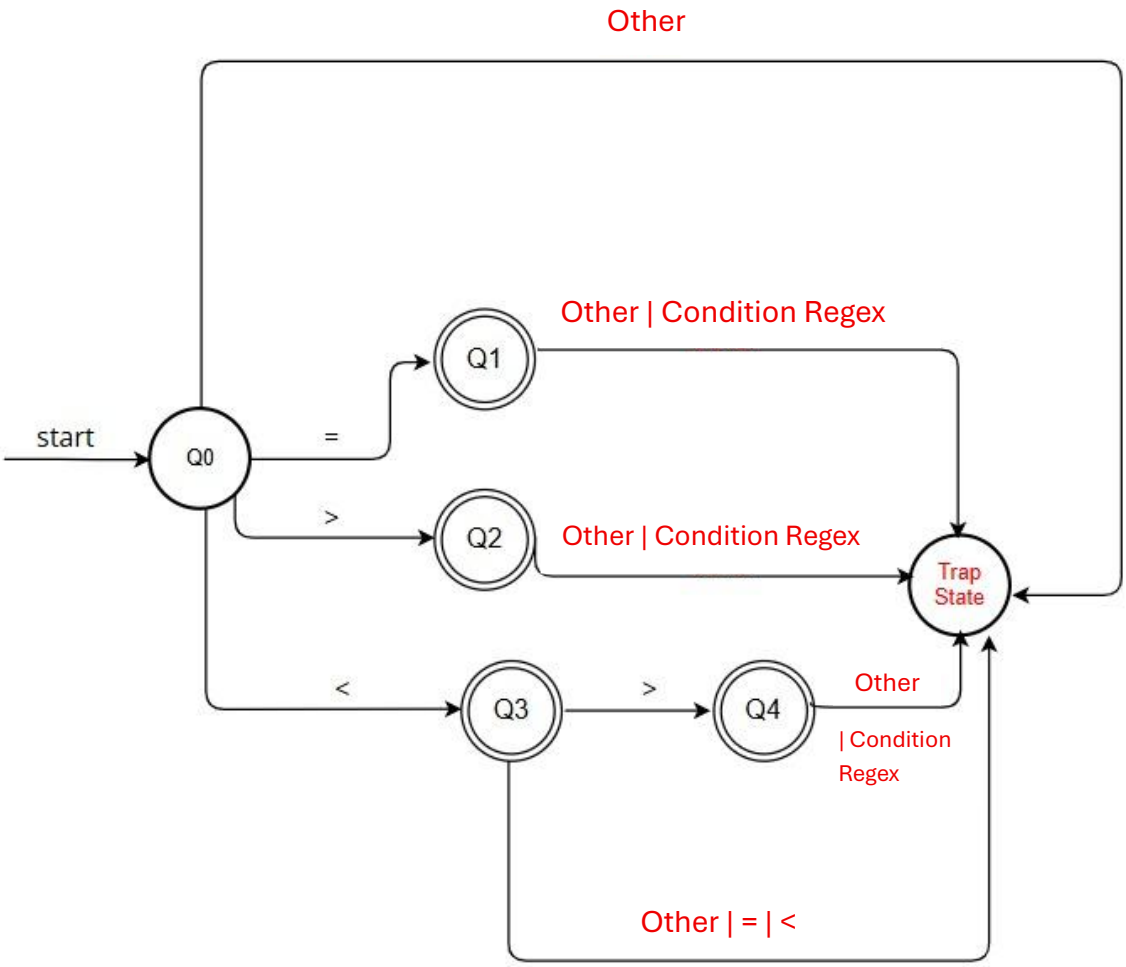
2.2.1 Regular Expression

Condition Regex: =|<|>|<>

Description: Matches comparison operators: , =, or <>.

Other: [A-Z] | [a-z] | [0-9] | : | & | | | (|) | { | } | ; | , | . | "| - | + | / | *

2.2.2 DFA Diagram



2.3 Boolean Operator

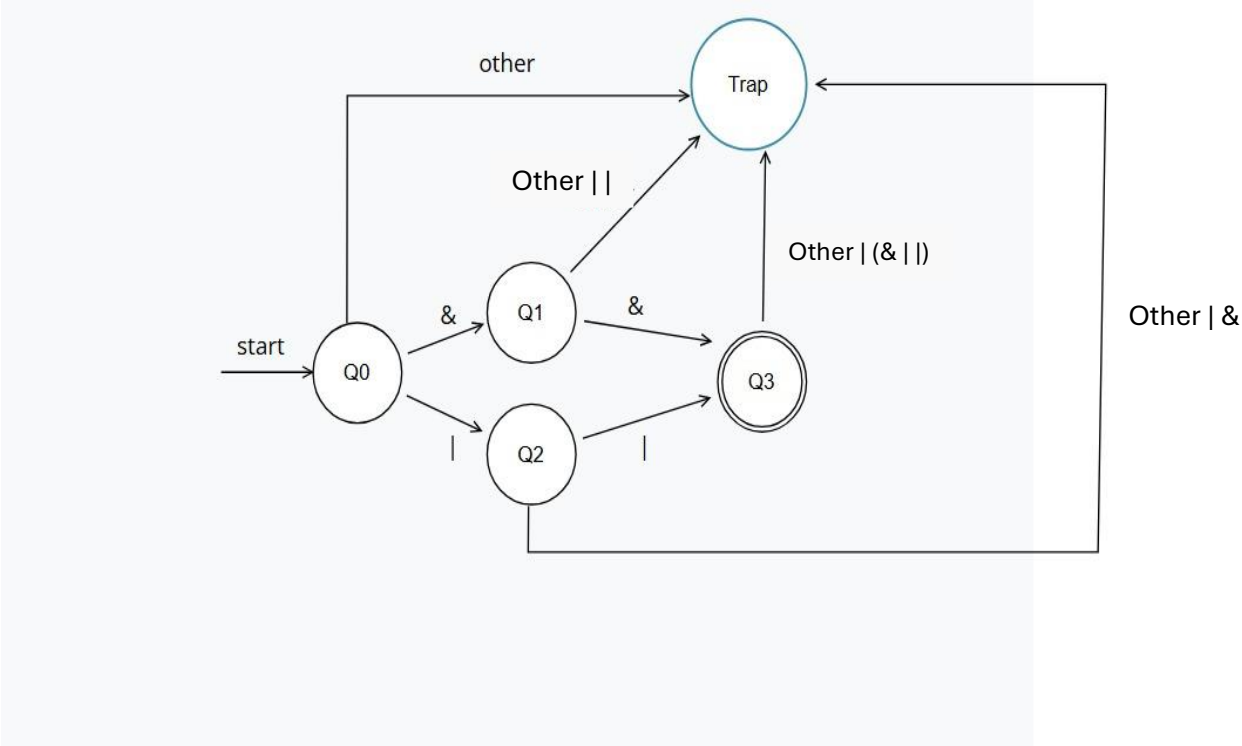
2.3.1 Regular Expression

Boolean Regex: && | ||

Description: Matches comparison operators: , =, or <>.

Other: [A-Z] | [a-z] | [0-9] | : | (|) | { | } | ; | , | . | "| - | + | / | *| = | < | >

2.3.2 DFA Diagram



2.4 Assignment Operator

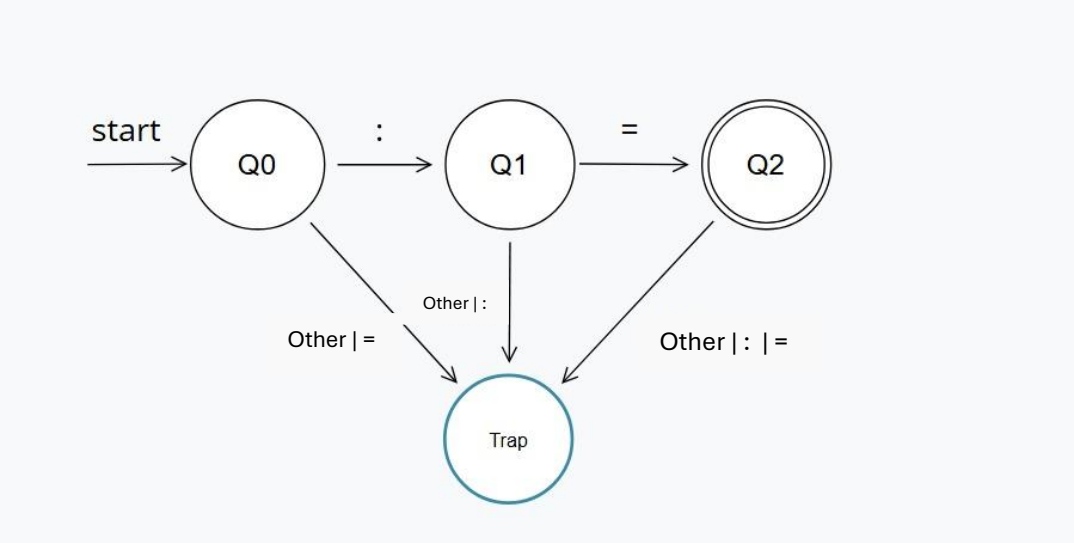
2.4.1 Regular Expression

Assignment Regex: :=

Description: Matches the assignment operator := used to assign values.

Other: [A-Z] | [a-z] | [0-9] | (|) | { | } | ; | , | . | "| - | + | / | * | < | > | & | |

2.4.2 DFA Diagram



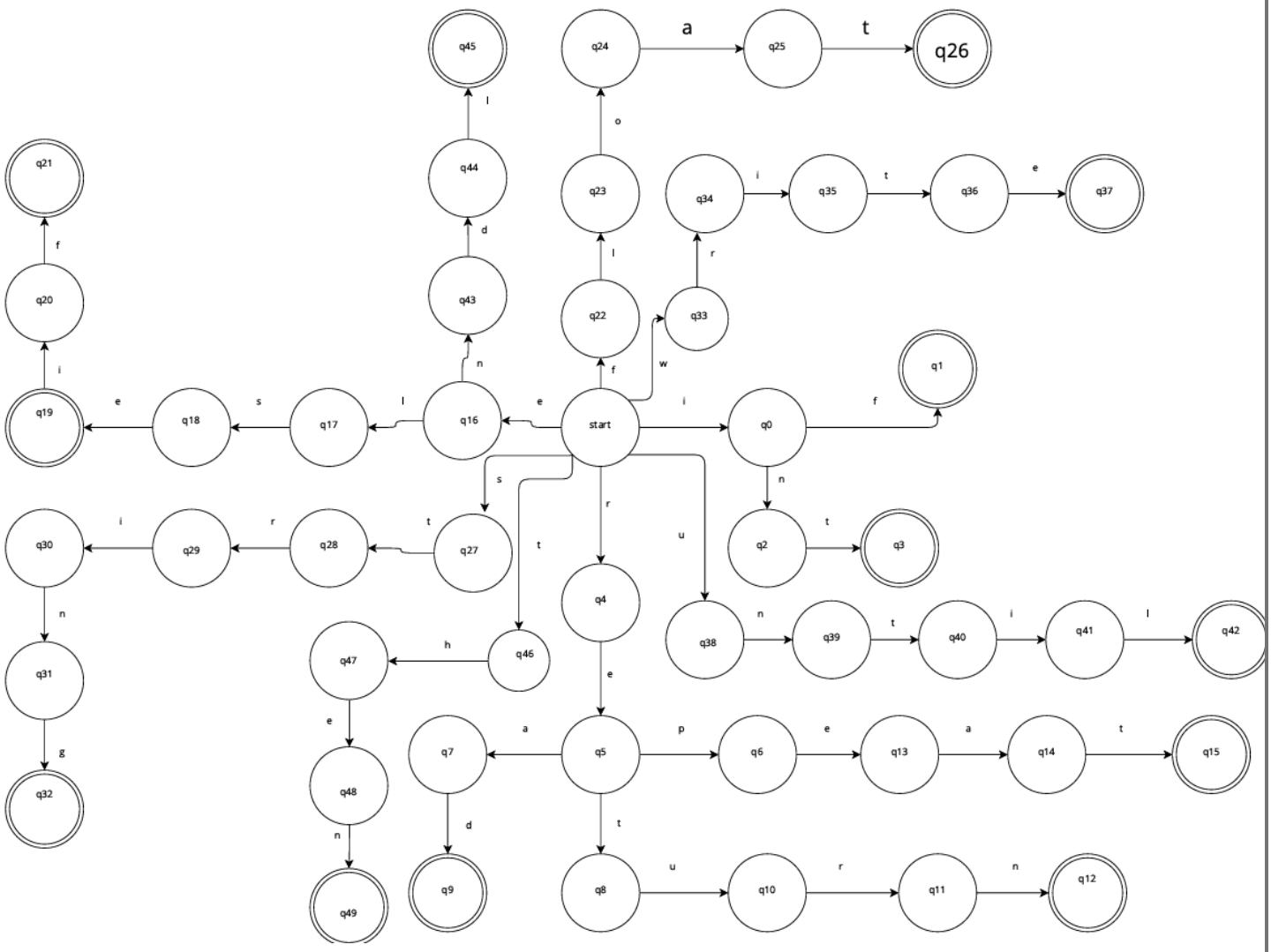
2.5 Reserved Keywords

2.5.1 Regular Expression

(int|float|string|read|write|repeat|until|if|elseif|else|then|return|endl)

Description: Matches reserved keywords used by Tiny Language.

2.5.2 DFA diagram



2.6 Number

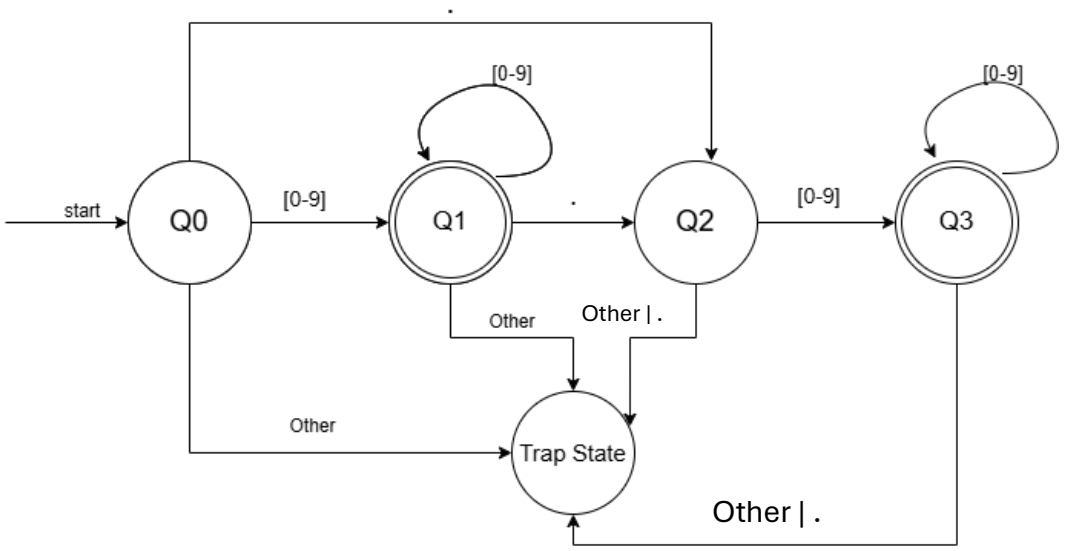
2.6.1 Regular Expression

Number Regex: ([0-9]+(\.[0-9]+)?|\.[0-9]+)

Description: Matches integers and floating-point numbers.
Examples: 42, 0.56, 1000.

Other: [A-Z] | [a-z] | (|) | { | } | ; | , | "| - | + | / | * | < | > | & | | | : | =

2.6.2 DFA Diagram



2.7 Identifier

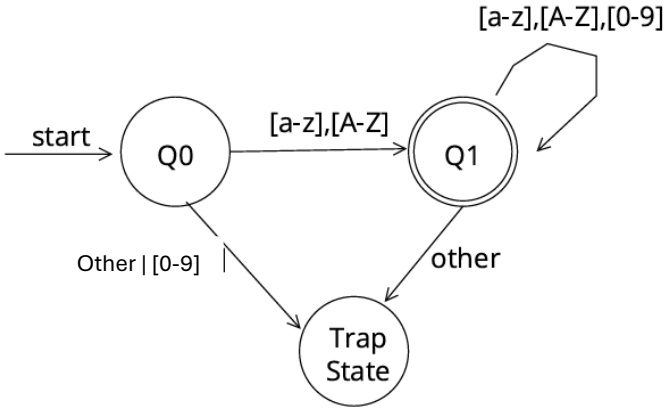
2.7.1 Regular Expression

Identifier Regex: ([a-z] | [A-Z]) ([a-z] | [A-Z] | [0-9])*

Description: Matches variable and function names starting with a letter and followed by alphanumeric characters.

Other: (|) | { | } | ; | , | "| - | + | / | * | < | > | & | | | : | =

2.7.2 DFA Diagram



2.8 String

2.8.1 Regular Expression

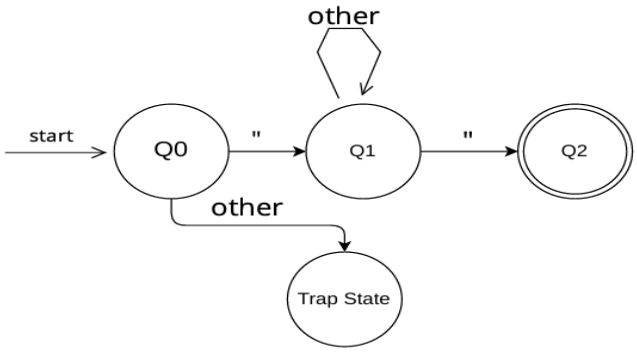
String Regex: ".*"

Description: Matches string literals enclosed in double quotes.

Examples: "Hello", "var123", "2 + 3 = 5".

Other: [A-Z] | [a-z] | [0-9] | (|) | { | } | ; | , | - | + | / | * | < | > | & | | | : | =

2.8.2 DFA Diagram



2.9 Comment Handling

Comments are not considered tokens in the lexical analysis phase, as they are ignored by the compiler and have no impact on the program's execution. Therefore, the comment pattern (starting with /* and ending with */) is not defined as a token in the lexical specification. Instead, comments are handled directly in the code, where any text between /* and */ is detected and skipped during scanning.

3.0 SCANNER OVERVIEW

3.1 Purpose

The scanner (lexical analyzer) will read Tiny Language source code and segment it into a series of tokens based on the above eight regular expressions. These tokens form the input for the syntax analysis (parser) phase.

3.2 How It Works

I. Reads source code character by character.
II. Matches substrings against the defined DFAs.
III. Produces token types (e.g., identifier, number, keyword, etc.).
IV. Skips irrelevant characters (e.g., whitespace, newlines) unless significant to the grammar.

3.3 Example Output

Input	**Token Type**
int	Reserved Keyword
Counter1	Identifier
:=	Assignment Op
5	Number
“Hello”	String
+	Arithmetic Op

4.0 SELECTION CRITERIA FOR THE 8 CORE REGULAR EXPRESSIONS

- They are the atomic patterns for lexical analysis.
- Higher-level language elements (e.g., assignment Statement, Repeat Statement, Function Call) are composed of these building blocks.
- Simplifies DFA creation by reducing redundancy and complexity.
- Makes the scanner lightweight and easier to extend in later project phases.

5.0 CONCLUSION

This report establishes the lexical foundation of the Tiny Language through eight critical regular expressions and their DFAs. These elements will drive the construction of an efficient scanner capable of tokenizing the Tiny Language for further parsing and code generation stages.