A Report on

LEXER IMPLEMENTATION USING ANTLR4 (SCANNER PART)

Submitted By:

Tanisha Majumder
Vaishak Balachandra

Under the Guidance of

Hairong Zhao, PhD
Professor of Computer Science; Graduate Advisor

Purdue University Northwest
Department of Computer Science
2025-26

LEXER TRANSITION FUNCTIONS (Optional Work)
(GROUP – 4)

This document presents the transition functions for all token types from our lexer grammar. Both transition diagrams and transition tables are provided for each token type to demonstrate how the lexer processes input characters.

Table of Contents (Categories):

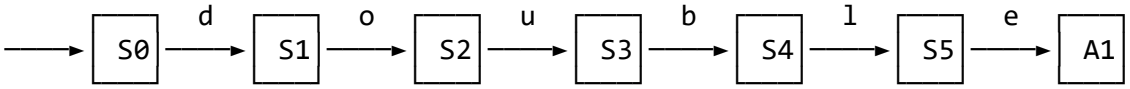
1. *KEYWORDS*

Regular Expression

```
'double'|'int'|'long'|'char'|'bool'|'fun'|'if'|'then'|'else'|'true'|'false'|'
                                orelse'|'andalso'
```

Transition Diagram

For keyword example: 'double':



(Similar diagrams exist for each keyword)

Transition Table

State	‘d’	‘o’	‘u’	‘b’	‘l’	‘e’	…	Other
S0	S1	-	-	-	-	-	…	-
S1	-	S2	-	-	-	-	…	-
…	…	…	…	…	…	…	…	…
A1*	-	-	-	-	-	-	…	-

- States marked with * are accepting states

- “-” indicates transition to an error state

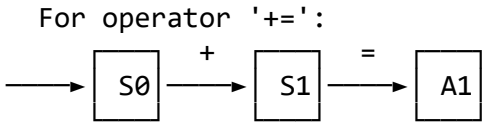
Description: - Each keyword is recognized as a specific sequence of characters. - The lexer compares the input against each keyword pattern. - Keywords have higher precedence than identifiers, ensuring that words like “if” and “else” are recognized as keywords rather than identifiers. - All keywords are reserved and cannot be used as identifiers. - The transition diagram shown is simplified for the keyword ‘double’; similar diagrams exist for each keyword.

2. *OPERATORS*

Regular Expression

```
'='|'+='|'-='|'*='|'/='|'+'|'-'|'*'|'/'|'//'|'>'|'<'|'=='|'!='|'!'
```

Transition Diagram



(Similar diagrams exist for each operator)

Transition Table

State	‘+’	‘=’	‘-’	’*’	‘/’	‘>’	‘<’	‘!’	Other
S0	S1	A2	S3	S4	S5	A6	A7	S8	-
S1	-	A9	-	-	-	-	-	-	-
S3	-	A10	-	-	-	-	-	-	-
…	…	…	…	…	…	…	…	…	…

- States marked with A* are accepting states
- “-” indicates transition to an error state
-

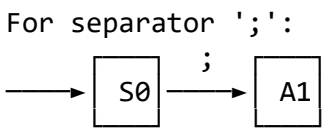
Description: - Operators are recognized as specific sequences of one or more characters. - Single-character operators (like ‘+’, ‘-’, ’*‘,’/‘) are directly accepted. - Multi-character operators (like’+=‘,’-=‘,’==‘,’!=‘) require additional transitions. - The lexer handles ambiguity by greedily matching the longest possible operator. - For example, when encountering’=‘, it checks if the next character forms part of a longer operator like’==’.

3. *SEPARATORS*

Regular Expression

```
';'|','|'('|')'|'{'|'}'
```

Transition Diagram



(Similar diagrams exist for each operator)

Transition Table

State	';'	','	'('	')'	'{'	'}'	Other
S0	A1	A2	A3	A4	A5	A6	-

- States marked with A* are accepting states
- "-" indicates transition to an error state

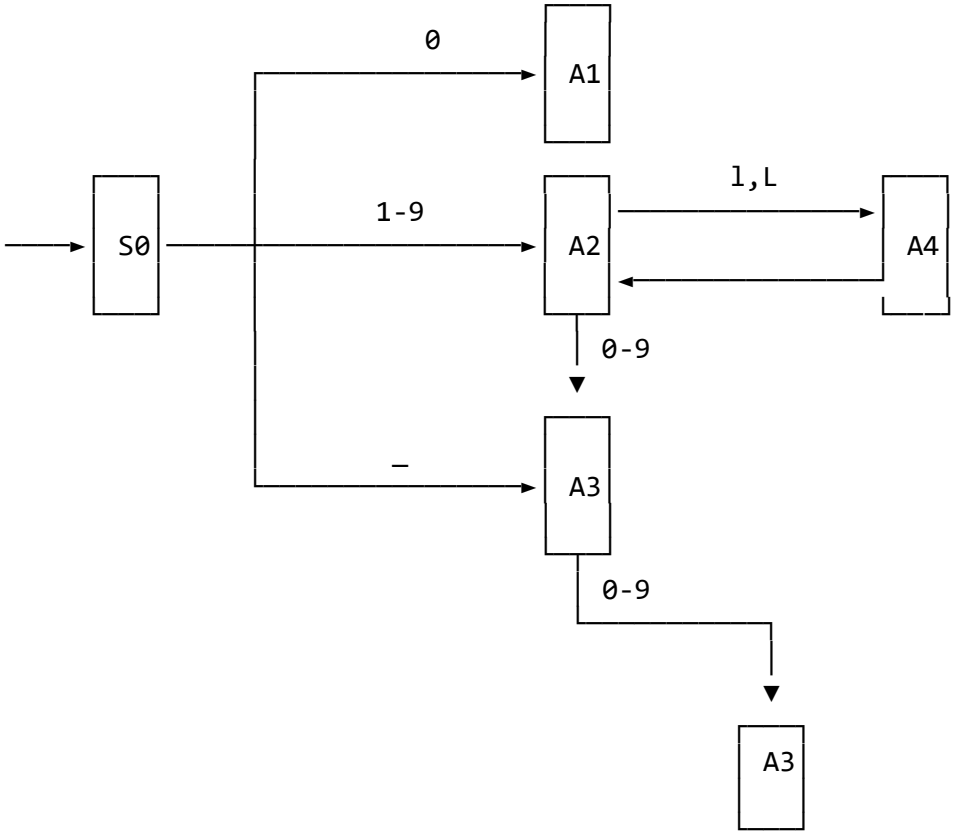
Description: - Separators are recognized as single-character tokens. - Each separator has a direct transition from the initial state to an accepting state. - Separators help structure the syntax by marking statement ends, parameter separation, and code blocks. - The lexer identifies these characters without considering surrounding context.

4. *INTEGER_LITERAL*

Regular Expression

```
'0'|[1-9]('_'*[0-9])*|[0-9]('_'*[0-9])*[lL]
```

Transition Diagram



Transition Table

State	0	1-9	_	l,L	Other
S0	A1	A2	-	-	-
A1*	-	-	-	-	-
A2*	A2	A2	A3	A4	-
A3	A2	A2	A3	-	-
A4*	-	-	-	-	-

- States marked with * are accepting states
- "-" indicates transition to an error state

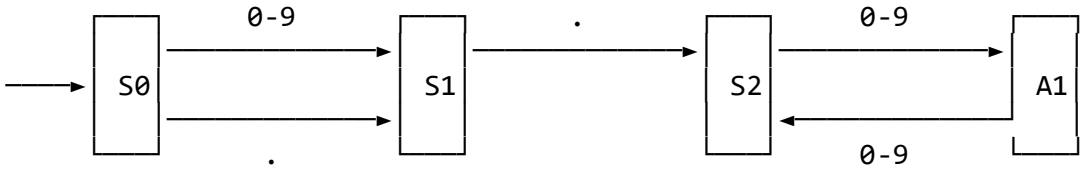
Description: - The INTEGER_LITERAL automaton handles valid integer patterns according to the grammar. - It allows decimal integers starting with 0 (only 0) or any non-zero digit. - It permits underscores between digits for readability. - It allows an optional suffix 'l' or 'L' for long integers. - The automaton rejects patterns with leading zeros followed by other digits. - It also rejects patterns starting or ending with underscores.

5. *DOUBLE_LITERAL*

Regular Expression

```
[0-9]+'.'[0-9]+|'.'[0-9]+
```

Transition Diagram



Transition Table

State	0-9	.	Other
S0	S1	S2	-
S1	S1	S2	-
S2	A1	-	-
A1*	A1	-	-

- States marked with * are accepting states
- "-" indicates transition to an error state

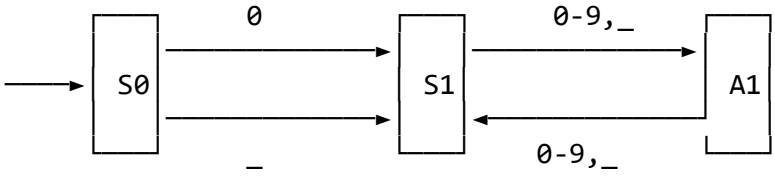
Description: - The DOUBLE_LITERAL automaton handles valid floating-point patterns. - It accepts patterns like "123.456" (digits, decimal point, digits). - It also accepts patterns like ".456" (decimal point followed by digits). - The automaton requires at least one digit after the decimal point. - It transitions to an error state for patterns like "123." (no digits after decimal).

6. *INVALID_INTEGER_LITERAL*

Regular Expression

```
'0'[0-9_]+|'_'[0-9_]+|[0-9]('_'*[0-9])*'_'
```

Transition Diagram



Transition Table

State	0	1-9	_	Other
S0	S1	S2	S3	-
S1	A1	A1	A1	-
S2	S2	S2	S4	-
S3	A1	A1	A1	-
S4	S2	S2	S4	A1
A1*	-	-	-	-

- States marked with * are accepting states
- "-" indicates transition to an error state

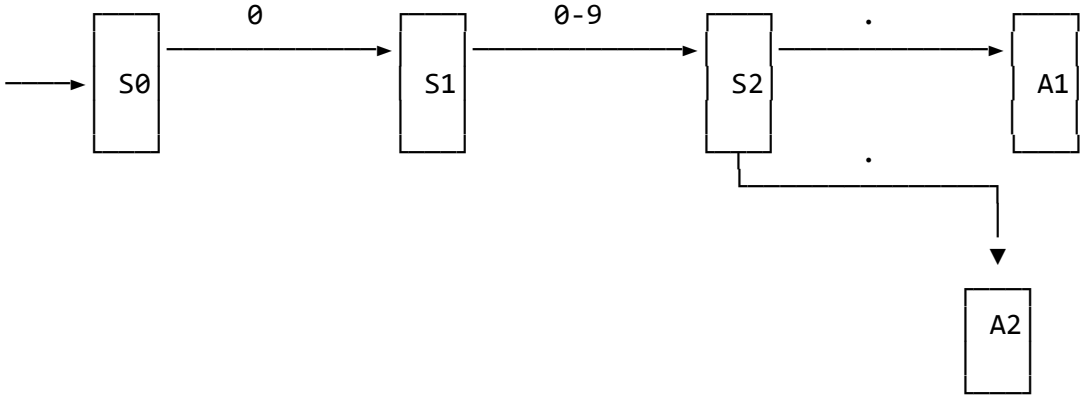
Description: - The INVALID_INTEGER_LITERAL automaton captures common invalid integer patterns. - It detects integers with leading zeros (e.g., "01234"). - It identifies integers starting with underscores (e.g., "*123"). - It catches integers ending with underscores (e.g., "123*"). - These patterns are flagged as errors to help catch potential coding mistakes. - The lexer provides specific error tokens to help with better error reporting.

7. *INVALID_DOUBLE_LITERAL*

Regular Expression

```
'0'[0-9]*'.'[0-9]*|[0-9]+'.'
```

Transition Diagram



Transition Table

State	0	1-9	.	Other
S0	S1	S2	-	-
S1	S1	S1	A1	-
S2	S2	S2	A2	-
A1*	A3	A3	-	-
A2*	-	-	-	-
A3*	A3	A3	-	-

- States marked with * are accepting states
- "-" indicates transition to an error state

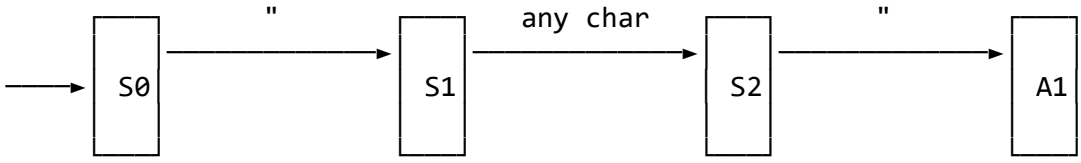
Description: - The INVALID_DOUBLE_LITERAL automaton captures common invalid floating-point patterns. - It detects doubles with leading zeros for the whole number part (e.g., "01.23"). - It catches doubles without any digits after the decimal point (e.g., "123."). - These patterns are flagged as errors to catch potential coding mistakes. - The separate handling of invalid double literals helps generate better error messages.

8. *CHAR_LITERAL*

Regular Expression

```
'"'.'\"'
```

Transition Diagram



Transition Table

State	”	Any Char	Other
S0	S1	-	-
S1	-	S2	-
S2	A1	-	-
A1*	-	-	-

- States marked with * are accepting states
- "-" indicates transition to an error state

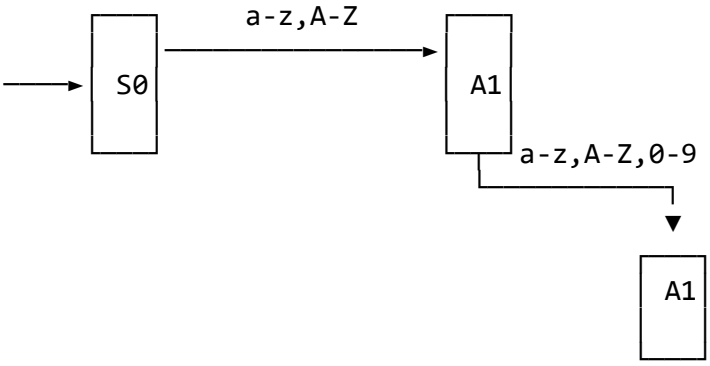
Description: - The CHAR_LITERAL automaton handles character literals enclosed in double quotes. - It requires exactly one character between the opening and closing double quotes. - The automaton recognizes patterns like "a", "1", or " " (a space character). - It rejects empty character literals ("") or multi-character literals. - This simple automaton does not handle escape sequences, which would require additional states.

9. *IDENTIFIER*

Regular Expression

```
[a-zA-Z][a-zA-Z0-9]*
```

Transition Diagram



Transition Table

State	a-z,A-Z	0-9	Other
S0	A1	-	-
A1*	A1	A1	-

- States marked with * are accepting states
- "-" indicates transition to an error state

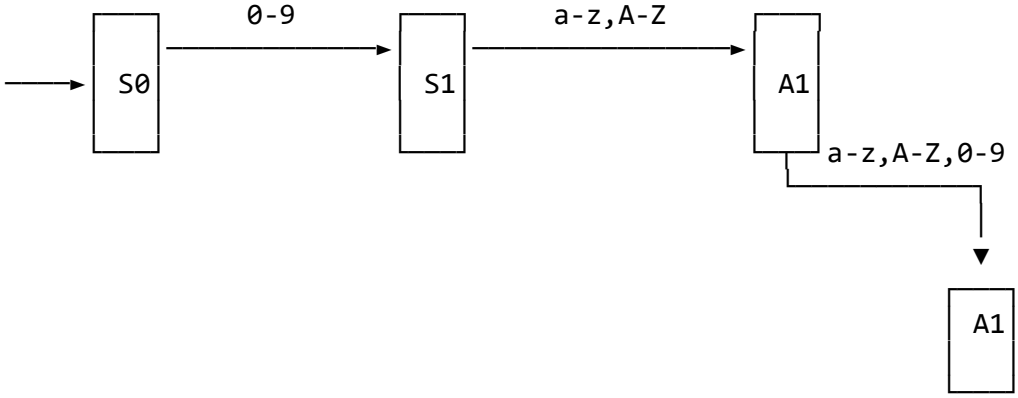
Description: - The IDENTIFIER automaton handles valid identifiers. - It requires identifiers to start with a letter (a-z or A-Z). - After the first character, it allows any combination of letters and digits. - The automaton rejects identifiers starting with digits or special characters. - Keywords are handled separately in the lexer to distinguish them from regular identifiers.

10. *INVALID_IDENTIFIER*

Regular Expression

```
[0-9]+[a-zA-Z][a-zA-Z0-9]*
```

Transition Diagram



Transition Table

State	0-9	a-z,A-Z	Other
S0	S1	-	-
S1	S1	A1	-
A1*	A1	A1	-

- States marked with * are accepting states
- "-" indicates transition to an error state

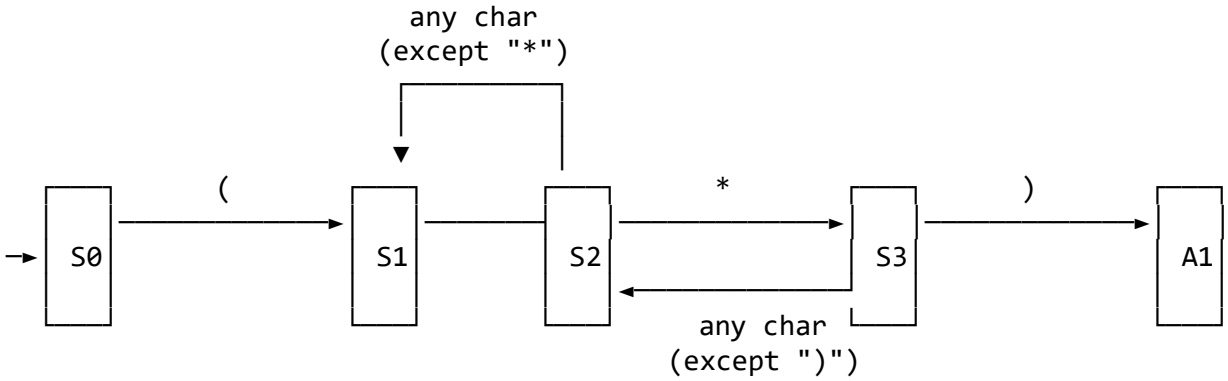
Description: - The INVALID_IDENTIFIER automaton specifically captures identifiers that start with digits. - It first consumes one or more digits. - Then it requires at least one letter to distinguish from INTEGER_LITERAL. - After that, it allows any combination of letters and digits. - These tokens are marked as invalid by the lexer since identifiers must start with a letter in the language.

11. *COMMENT*

Regular Expression

```
'(*'.*?'*)'
```

Transition Diagram



Transition Table

State	(	*	)	Other
S0	S1	-	-	-
S1	-	S2	-	S1
S2	-	S3	-	S1
S3	-	S3	A1	S1
A1*	-	-	-	-

- States marked with * are accepting states
- "-" indicates transition to an error state

Description: - The COMMENT automaton handles the (* ... *) comment style. - It starts by recognizing an opening "(" followed by "*". - *It then consumes all characters until it encounters a "*"* followed by ")". - The automaton handles nested comment delimiters by treating them as regular

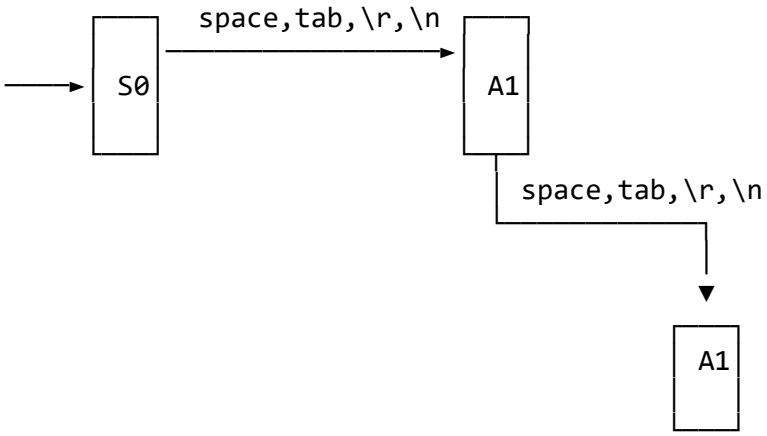
characters. - Once a complete comment is recognized, it is skipped by the lexer and not returned as a token.

12. *WHITESPACE*

Regular Expression

```
[ \t\r\n]+
```

Transition Diagram



Transition Table

State	space	tab	\r	\n	Other
S0	A1	A1	A1	A1	-
A1*	A1	A1	A1	A1	-

- States marked with * are accepting states
- "-" indicates transition to an error state

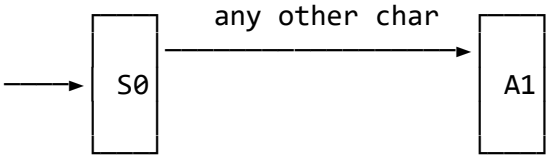
Description: - The WHITESPACE automaton handles any sequence of whitespace characters. - It recognizes spaces, tabs, carriage returns, and newlines. - It requires at least one whitespace character to match. - The automaton allows any combination and length of whitespace characters. - Once whitespace is recognized, it is skipped by the lexer and not returned as a token.

13. *ERROR*

Regular Expression

.

Transition Diagram



Transition Table

State	Any Char
S0	A1
A1*	-

- States marked with * are accepting states
- "-" indicates transition to an error state

Description: - The ERROR automaton is a catch-all for any character not recognized by other rules. - It matches any single character that doesn't form part of a valid token. - This ensures that lexical analysis doesn't fail on unexpected input. - It helps identify invalid characters or symbols in the source code. - The error token allows the parser to provide meaningful error messages about invalid syntax.