Regular Grammars for Lexical Analyzer

COSC 3127 Programming Languages - Assignment 1

Overview: This document defines the regular grammars for the Mini language lexical analyzer. The language supports two data types (integer and real), the assignment operator (:=), and five arithmetic operators (+, -, *, /, ^).

Grammar Notation Legend

S, A, B = Non-terminal symbols (states)

'a', '0', '+' = Terminal symbols (characters)

→ = Production rule (derives to)

ε = Epsilon (empty string)

1. Identifier Grammar

Definition: An identifier must start with a letter (a-z, A-Z) or underscore (_), followed by zero or more letters, digits (0-9), or underscores.

```
Regular Expression: [a-zA-Z_][a-zA-Z0-9_]*
```

Right-Linear Grammar:

```
S → a A | b A | ... | z A

S → A A | B A | ... | Z A

S → _ A

A → a A | b A | ... | z A

A → A A | B A | ... | Z A

A → 0 A | 1 A | ... | 9 A

A → _ A

A → ε
```

Valid Examples: x, var1, total_sum, _count, myVariable

```
   a-z, A-Z, _         a-z, A-Z, 0-9, _
         ┌─────┐        ┌─────┐
───►│  S  │────────►│  A  │◄──────┐
     └─────┘        └──┬──┘       │
                       │          │
                       │ ε (accept)
                       ▼          │
                    ACCEPT        │
                                  │
                     └────────────┘
```

2. Integer Literal Grammar

Definition: An integer is a sequence of one or more digits.

`Regular Expression: [0-9]+`

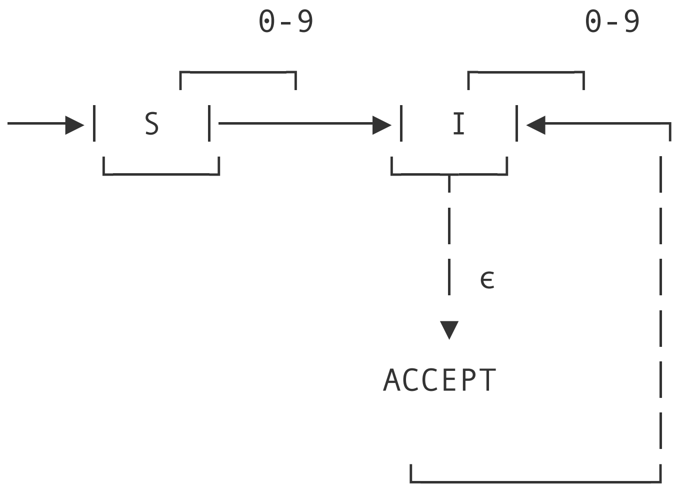
Right-Linear Grammar:

S → 0 I | 1 I | 2 I | ... | 9 I

I → 0 I | 1 I | 2 I | ... | 9 I

I → ε

Valid Examples: 0, 42, 123, 9999



3. Real Number Literal Grammar

Definition: A real number consists of one or more digits, followed by a decimal point, followed by one or more digits.

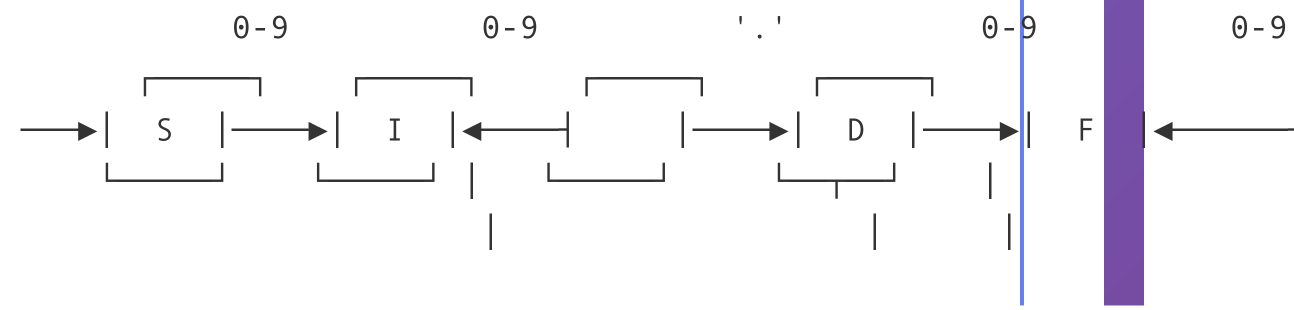
`Regular Expression: [0-9]+\.[0-9]+`

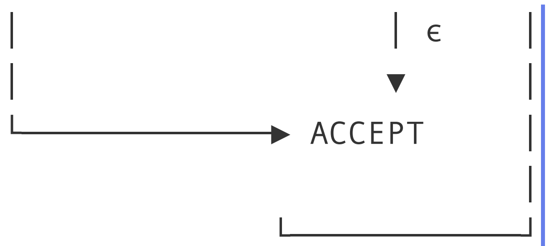
Right-Linear Grammar:

```
S  →  0 I | 1 I | 2 I | ... | 9 I

I  →  0 I | 1 I | 2 I | ... | 9 I

I  →  . D

D  →  0 F | 1 F | 2 F | ... | 9 F

F  →  0 F | 1 F | 2 F | ... | 9 F

F  →  ε
```

Valid Examples: 3.14, 0.5, 123.456, 99.99

Note: The grammar requires at least one digit before and after the decimal point. Inputs like ".5" or "3." are rejected.





4. Assignment Operator Grammar

Definition: The assignment operator is the two-character sequence ":=".

`Regular Expression: :=`

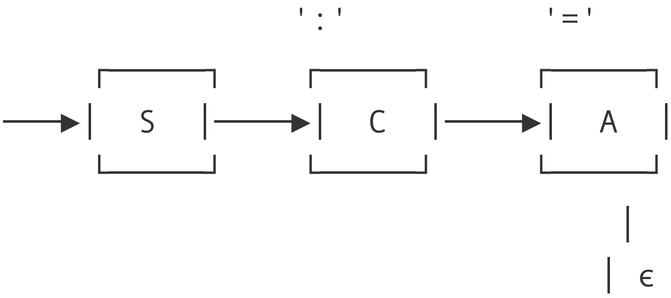
Right-Linear Grammar:

S → : C

C → = A

A → *ε*

Valid Example: :=



```
                    ▼
                 ACCEPT
```

5. Arithmetic Operators Grammar

Definition: Single-character arithmetic operators: +, -, *, /, ^

Regular Expression: `[+\-*/^]`

Right-Linear Grammar:

```
S  →  + O
S  →  - O
S  →  * O
S  →  / O
S  →  ^ O
O  →  ε
```

Valid Examples: +, -, *, /, ^

```
                     +, -, *, /, ^
          ┌─────┐              ┌─────┐
    ────►│  S  │────────────►│  O  │
          └─────┘              └─────┘
                                   |
                                        │ ϵ
                                        ▼
                                     ACCEPT
```

6. Complete Token Summary

Token Type	Regular Expression	Grammar Type	Example
IDENTIFIER	`[a-zA-Z_][a-zA-Z0-9_]*`	Right-Linear	myVar, _temp, x1
INTEGER	`[0-9]+`	Right-Linear	42, 0, 1234
REAL	`[0-9]+\.[0-9]+`	Right-Linear	3.14, 0.5, 99.99
ASSIGNMENT	`:=`	Right-Linear	:=
OPERATOR	`[+\-*/^]`	Right-Linear	+, -, *, /, ^

7. Grammar Properties

Right-Linear Grammar Properties:

- All productions are of the form: A → aB or A → a or A → ε
- Non-terminal appears only on the right side of productions
- Generates regular languages (Type 3 in Chomsky hierarchy)
- Can be directly converted to Deterministic Finite Automata (DFA)
- Equivalent to regular expressions in expressive power

Implementation Note: Each regular grammar in this document has been implemented as a DFA in the Java lexical analyzer. The DFA states correspond to the non-terminal symbols in the grammars, with transitions representing the production rules.

8. Lexical Analysis Process

The lexer applies these grammars in the following order:

1. **Identifier** - Checked first to capture variable names
2. **Real** - Checked before Integer to capture decimal numbers
3. **Integer** - Checked for whole numbers
4. **Assignment** - Checked for := operator
5. **Operator** - Checked for arithmetic operators

The lexer uses the **maximal munch** principle: it always selects the longest possible match for each token.

COSC 3127 Programming Languages - Assignment 1

Lexical Analyzer Regular Grammars Documentation

Author: Toluwa Fayemi

Group: Ayush, Khusi, Sai

Date: November, 2025