Compiler Assignment Report: Traffic Light Control System

CT3 - Syntax Analysis and Parse Tree Generation

Team Information

Members:

Saifuddin Shawon - 202214092

Md. Saif Ahmed - 202314008

Md. Ragib Hossain Rifat - 202314022

Mohammad Arafat Hasan Sarker - 202314069

Shihab Ahmed - 202314049

Project Overview

This compiler assignment implements a domain-specific language (DSL) for traffic light control systems. The project demonstrates the fundamental phases of compilation: lexical analysis, syntax analysis, and semantic analysis through parse tree generation with semantic annotations.

Domain Context

The language is designed to specify traffic light state transitions, where each state defines:

- Current light color (RED, GREEN, YELLOW)
- Timing duration (numeric value)
- Next state transition
- Associated action (STOP, GO, PREPARE)

Grammar Specification

The formal grammar defines the structure of traffic light control statements using context-free grammar rules:

Production Rules

S → StateList

StateList → State StateList | State

State → "state" ID NUM "->" ID "{" Action "}" ";"

ID → RED | GREEN | YELLOW

NUM → DIGITS

DIGITS → DIGIT DIGITS | DIGIT

DIGIT → 0|1|2|3|4|5|6|7|8|9

Action → "STOP" | "GO" | "PREPARE"

Grammar Analysis

- **Start Symbol:** S (represents the complete program)
- **Non-terminals:** S, StateList, State, ID, NUM, DIGITS, DIGIT, Action

- **Terminals:** "state", "->", "{", "}", ";", RED, GREEN, YELLOW, STOP, GO, PREPARE, digits 0-9
- **Grammar Type:** Context-free grammar suitable for LL parsing
- **Recursion:** Left recursion in StateList allows multiple state definitions

---

Semantic Rules

The semantic rules define how meaning is propagated through the parse tree using synthesized attributes:

Attribute Grammar Rules

S.transitions = StateList.transitions

StateList.transitions = {State.transition} ∪ $StateList_1$.transitions

StateList.transitions = {State.transition}

NUM.value = integer(DIGITS.value)

DIGITS.value = DIGIT.value ∘ $DIGITS_1$.value

DIGITS.value = DIGIT.value

DIGIT.value = tokenValue

ID.name = tokenValue

Action.value = tokenValue

Semantic Analysis Features

- **Transition Collection:** Aggregates all state transitions into a set
- **Type Conversion:** Converts digit sequences to integer values
- **String Concatenation:** Builds multi-digit numbers through concatenation
- **Symbol Table:** Maps tokens to their semantic values

Lexical Analysis

Token Classification

The lexical analyzer recognizes the following token types:

Token	Type	Description
`state`	KEYWORD	Reserved word for state declaration
`RED/GREEN/YELLOW`	ID	Traffic light color identifiers

Token	Type	Description
`10`	NUM	Numeric duration value
`->`	ARROW	State transition operator
`{`	LBRACE	Opening brace
`}`	RBRACE	Closing brace
`;`	SEMICOLON	Statement terminator
`STOP/GO/PREPARE`	ACTION	Traffic control actions

Example Tokenization

Input: `state RED 10 -> YELLOW {STOP};`

Token Stream:

(state, KEYWORD)

(RED, ID)

(10, NUM)

(->, ARROW)

(YELLOW, ID)

({, LBRACE)

(STOP, ACTION)

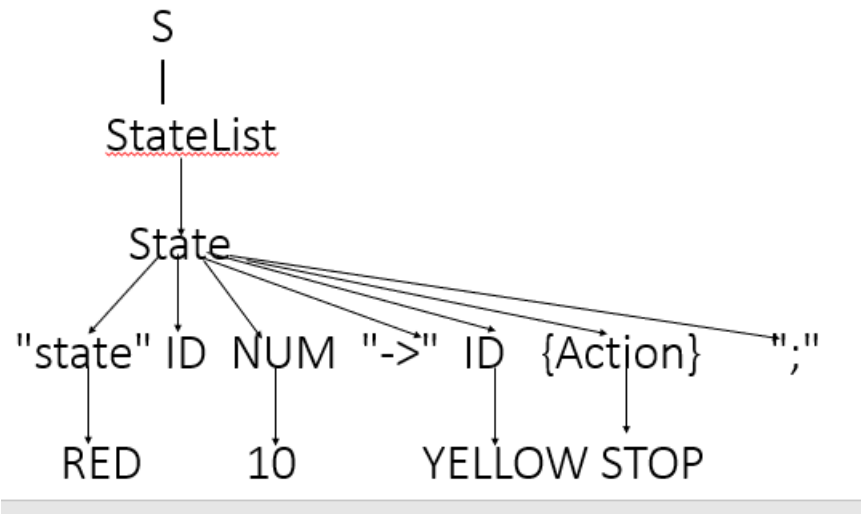
(}, RBRACE)

(;, SEMICOLON)

Syntax Analysis

Parse Tree Structure

The syntax analyzer constructs a hierarchical parse tree following the grammar rules:



Parsing Strategy

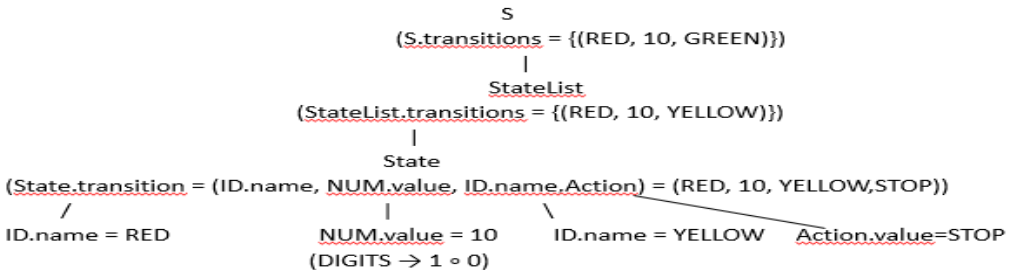
- **Top-down parsing** approach following grammar production rules
- **Leftmost derivation** used for parse tree construction
- **Syntax validation** ensures input conforms to grammar rules

---

Semantic Analysis with Parse Tree Annotation

Annotated Parse Tree

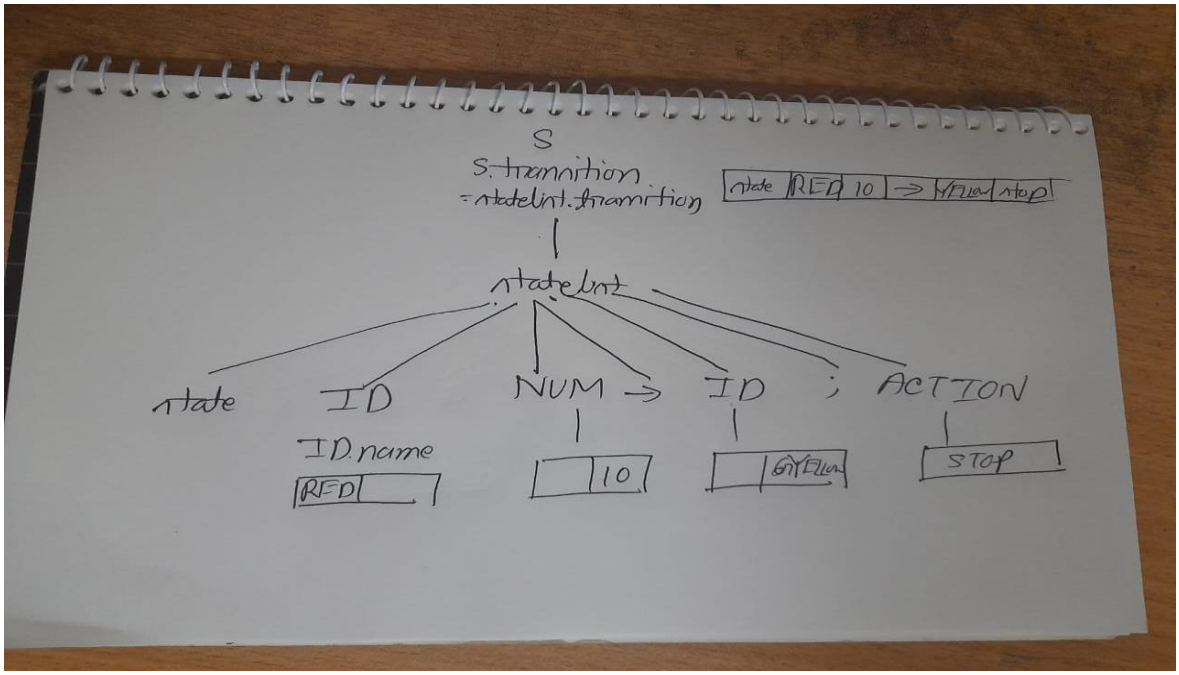
The parse tree is enhanced with semantic attributes showing value propagation:



Semantic Attributes

- **State Transition Tuple:** (current_state, duration, next_state, action)
- **Numeric Value Synthesis:** Combines individual digits into complete numbers
- **Symbol Resolution:** Maps identifiers to their semantic meanings

Syntax tree



Technical Implementation Details

Compiler Architecture

1. **Lexical Analyzer:** Tokenizes input stream and classifies tokens
2. **Syntax Analyzer:** Builds parse tree using recursive descent or table-driven parsing
3. **Semantic Analyzer:** Annotates parse tree with attribute values
4. **Symbol Table:** Manages identifier bindings and type information

Key Features

- **Error Handling:** Detects and reports lexical and syntax errors
- **Attribute Evaluation:** Implements synthesized attribute grammar
- **Tree Construction:** Builds explicit parse tree data structure
- **Semantic Validation:** Ensures type consistency and semantic correctness

Language Characteristics

- **Domain-Specific:** Tailored for traffic light control applications
- **Declarative Syntax:** States are declared with clear transition rules
- **Type Safety:** Enforces correct usage of colors, numbers, and actions
- **Extensibility:** Grammar can be extended for additional features

---

Sample Program Analysis

Input Program

```
state RED 10 -> YELLOW {STOP};
```

Compilation Phases

1. **Lexical:** Tokenizes into 9 distinct tokens
2. **Syntactic:** Validates grammar compliance and builds parse tree
3. **Semantic:** Extracts transition (RED, 10, YELLOW, STOP) with type checking

Output

- **Transition Set:** {(RED, 10, YELLOW, STOP)}
- **Parse Tree:** Complete syntactic structure
- **Semantic Annotations:** Type-checked attribute values

Conclusion

This compiler assignment successfully demonstrates the core phases of compilation for a domain-specific traffic light control language. The implementation covers:

- **Complete lexical analysis** with proper token classification
- **Syntax analysis** following formal grammar rules
- **Semantic analysis** with attribute grammar evaluation
- **Parse tree generation** with semantic annotations

The project provides a solid foundation for understanding compiler construction principles and can be extended with additional features such as code generation, optimization, and more complex semantic analysis.

Future Enhancements

- Code generation for target traffic control systems
- Optimization phases for efficient execution

- Extended grammar for complex traffic scenarios
- Error recovery mechanisms
- Integration with real traffic control hardware

Project Status: Complete
Deliverables: Grammar specification, lexical analyzer, syntax analyzer, semantic rules, annotated parse tree