Domain Specific Language with IDE for Commision Plan Designers

19z325 - Kavin Manikandan
20Z302 - Abinay
20Z345 - Shruthi Uday
20Z357 - Anukul Singh
20Z362 - Irfan

19Z620 - INNOVATION PRACTICES

BACHELOR OF ENGINEERING

BRANCH: COMPUTER SCIENCE AND ENGINEERING

MARCH 2023

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PSG COLLEGE OF TECHNOLOGY

(Autonomous Institution)

COIMBATORE – 641 004

PSG COLLEGE OF TECHNOLOGY

(Autonomous Institution)

COIMBATORE – 641 004

Domain Specific Language with IDE for Commision Plan Designers

Bonafide record of work done by

19z325 - Kavin Manikandan
20Z302 - Abinay
20Z345 - Shruthi Uday
20Z357 - Anukul Singh
20Z362 - Irfan

Dissertation submitted in partial fulfillment of the requirements for the degree of

BACHELOR OF ENGINEERING

BRANCH: COMPUTER SCIENCE AND ENGINEERING

of Anna University

March 2023

_____

Dr. Kavitha C
Faculty Guide
Assistant Professor (Sl.Gr.)
Department of Computer Science & Engineering

Certified that the candidate was examined in the viva voce examination held on _____

_____ Internal Examiner

_____ External Examiner

CONTENTS

ACKNOWLEDGEMENTS

We wish to express our sincere gratitude to our beloved Principal Dr.Prakasan for providing an opportunity and necessary facilities in carrying out this dissertation work.

We also wish to express our sincere thanks to Dr G Sudha Sadasivam, Head of Computer Science and Engineering Department, for her encouragement and support that she extended towards this dissertation work.

We also wish to express our sincere thanks to Mr Vivek, Mr Dg and Ms Nivedha, mentors from the company Everstage, for their encouragement and support that they extended towards this dissertation work.

Our sincere thanks are due to Dr Kavitha C, Assistant Professor (Sl.Gr.), Department of Computer Science & Engineering, project guide whose valuable guidance and continuous encouragement throughout the course made it possible to complete this dissertation work well in advance.

Finally, we would like to thank all of our student colleagues, staff members in the Computer Science Engineering department without whom this dissertation work would not have been completed successfully.

SYNOPSIS

A domain specific language(DSL) with an integrated development environment(IDE) for a commission plan designer would provide a way to create and modify commission plans using a specialized language that is tailored to the specific needs of the commission plan designer. The IDE would include tools such as a syntax highlighting editor, code completion, and debugging capabilities to make the process of designing commission plans are more efficient and user-friendly

The DSL would provide a way to specify various components of a commission plan, such as commission rate, the eligible products or services, and the conditions under which a commission is paid out. This would allow designers to easily create and modify complex commission plans without needing to have a deep understanding of programming.

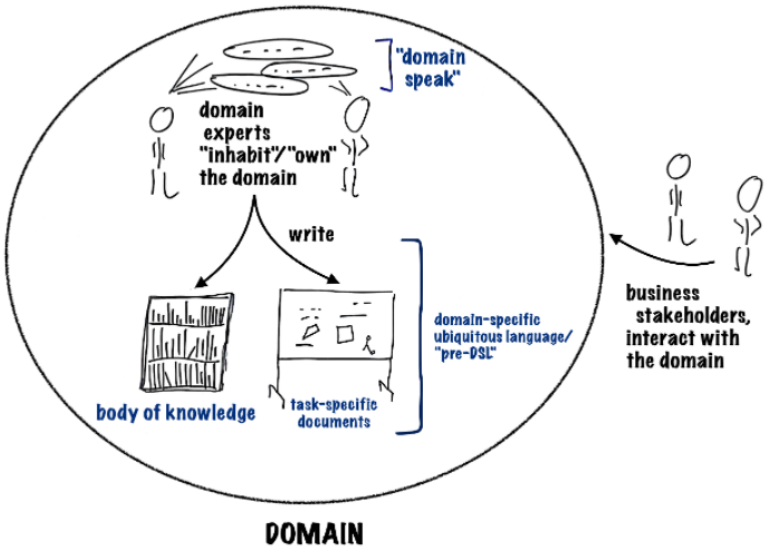


Fig.1.1: DSL Diagram

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Introduction

A commission plan designer is a tool that allows sales organizations to create and manage commission plans for their sales representatives. In this project, we are creating a domain-specific language (DSL) to simplify the process of designing commission plans.

To implement the DSL, we are using ANTLR, a powerful parser generator that allows us to define a grammar for the language. With this grammar, ANTLR generates a parser that can be used to validate and interpret the DSL code.

To provide a more user-friendly experience, we are also developing a VS Code extension that includes code completion, syntax highlighting, and debugging capabilities for the DSL. This extension will allow users to write commission plans more efficiently and with fewer errors.

The DSL will include features such as variable assignment, arithmetic operations, and conditional statements, which are commonly used in commission plan design. With the help of ANTLR and the VS Code extension, we hope to make commission plan design more accessible to sales organizations of all sizes.

Overall, this project aims to streamline the commission plan design process and improve the accuracy of commission calculations.

1.2 Motivation

The motivation for this project is to improve the efficiency and accuracy of commission plan design.

Commission plans are commonly used to incentivize sales teams to meet specific goals, and they can be quite complex, involving multiple variables and calculations. Creating commission plans can be a time-consuming and error-prone process, especially if done manually. By creating a DSL and VS Code extension specifically for commission plan designers, the goal is to provide a more streamlined and user-friendly process for creating and managing commission plans.

A DSL is a programming language designed to solve problems in a specific domain, such as commission plan design. By creating a DSL for commission plan design, designers can focus on the high-level concepts and rules of the plan rather than the low-level syntax of a general-purpose programming language.

The VS Code extension with code completion, syntax highlighting, and debugging capabilities further improves the efficiency of the commission plan design process. Code completion and syntax highlighting help designers write code faster and with fewer errors, while debugging capabilities allow them to identify and fix errors more quickly.

Overall, the motivation behind this project is to provide a more efficient and accurate process for commission plan design, allowing businesses to incentivize and reward their sales teams more effectively.

1.3 Problem Statement

To develop a domain specific language(DSL) with an integrated development environment (IDE) for a commission plan designer

1. The language should have an intuitive syntax that is easy for non-programmers like subject matter experts to understand.

2. IDE should have syntax highlighting, code completion and debugging facilities.

1.4 Objectives

The objectives of this project are:

Design a DSL for commission plan design: The first objective is to design a DSL that is specifically tailored to the needs of commission plan designers. This involves identifying the key concepts and rules of commission plan design and creating a syntax and grammar that is easy to read and write.

Implement a VS Code extension: The second objective is to implement a VS Code extension that supports the DSL and provides code completion, syntax highlighting, and debugging capabilities. This involves developing the necessary tools and integrating them into VS Code to make commission plan design as seamless and intuitive as possible.

Test and refine the DSL and extension: The third objective is to test the DSL and extension thoroughly to ensure that they are working as expected and meeting the needs of commission plan designers. Any issues or bugs that are identified during testing should be addressed promptly, and the DSL and extension should be refined as needed to improve their usability and effectiveness.

CHAPTER 2

LITERATURE SURVEY

"Domain-Specific Languages" by Martin Fowler and Rebecca Parsons: This book provides an introduction to domain-specific languages (DSLs) and describes their benefits in solving specific problems. It also discusses the process of designing and implementing DSLs.

"The Definitive ANTLR 4 Reference" by Terence Parr: This book provides a comprehensive guide to using ANTLR (ANother Tool for Language Recognition) to build DSLs. It covers the basics of ANTLR, including lexer and parser generation, tree construction, and error handling.

"Domain-Specific Languages Made Easy with MPS" by Markus Voelter: This book provides an introduction to the JetBrains Meta Programming System (MPS) and its use in building DSLs. It covers topics such as DSL design, implementation, and integration.

CHAPTER 3

SYSTEM REQUIREMENTS

3.1 Hardware Requirements

- A pc/laptop with stable internet connection which has the following specifications:
- Processor: A fast, multi-core processor would be recommended to support the computational demands of the IDE and DSL
- Memory: A minimum of 4 GB of RAM would be required, with 8 GB or more recommended for larger, more complex commission plan designs.
- Storage: A solid-state drive (SSD) or hard disk drive (HDD) with a minimum of 250 GB of storage capacity would be required, with more storage recommended for larger projects or if the IDE includes version control functionality.
- Display: A high-resolution display with a minimum resolution of 1920x1080
- Peripherals: A keyboard and mouse or trackpad would be required for input.

3.2 Software Requirements

- Operating System: A modern operating system such as Windows 10, macOS, or Linux would be required
- Development Environment: IDE will be built using a Language Workbench called MPS
- Dependent Libraries: The implementation may require several third-party libraries and frameworks, such as a lexer/parser generator library for the DSL, a GUI library for the IDE, and a database driver library to interface with the DBMS.

3.3 Functional Requirements

3.3.1 Commission Plan Expression Syntax

- The DSL should allow users to define commission plans using infix expressions. The syntax should include the following:
- Basic arithmetic operations (e.g., addition, subtraction, multiplication, division)
- Parentheses for grouping expressions
- Variables for referring to sales data (e.g., sales amount, sales quantity)
- Comparison operators (e.g., greater than, less than, equal to)
- Logical operators (e.g., AND, OR, NOT)

3.3.2 If-Else Condition Syntax

The DSL should allow users to define if-else conditions. The syntax should include the following:

- The 'if' keyword
- Comparison expressions using the syntax defined
- The 'then' keyword
- The commission plan expression to be executed if the condition is true
- The 'else' keyword (optional)
- The commission plan expression to be executed if the condition is false (optional)

3.3.3 Lexer and Parser

- The lexer will tokenize the input commission plan expression into a stream of tokens.
- The parser will generate a parse tree from the stream of tokens, based on the commission plan language grammar.
- The lexer and parser will work together to ensure that the commission plan expression is syntactically valid and can be parsed correctly.
- The parse tree generated by the parser will be used as input for the infix to prefix converter component.

3.3.4 JSON Converter

- The JSON converter will convert the commission plan expression and structure to JSON format.
- The JSON format will include the commission plan expression in prefix notation, the commission plan structure, and the variables used in the expression.
- The output of the JSON converter will be used by the IDE component for displaying the commission plan structure and variables and for evaluation by the backend.

3.3.5 IDE Features

The IDE should provide the following features to the users:

- Syntax highlighting and code completion for commission plans
- The IDE will provide a graphical user interface for writing, testing, and debugging commission plans.
- The IDE will communicate with the lexer, parser, infix to prefix converter, and JSON converter components to provide real-time syntax highlighting, code completion, and error highlighting.
- When a user enters a commission plan expression, the IDE will send it to the lexer component for tokenization.
- The lexer will then send the token stream to the parser component for generating a parse tree.
- The parse tree will be used by the infix to prefix converter component to generate the equivalent expression in prefix notation.
- The output of the JSON converter will be displayed in the IDE.
- The IDE will allow users to set breakpoints, inspect variables, and step through commission plan expressions during debugging.
- The IDE will integrate with source control systems, such as Git, for versioning and collaboration.
- The IDE will allow users to export and import commission plans in a variety of formats, such as CSV, JSON, and XML.

3.3.6 Error Handling

- The DSL should provide informative error messages when users provide invalid input. The error messages should help users identify and correct errors in their commission plans.

3.4 Non-functional Requirements

3.4.1 Performance

The DSL and IDE should be designed to handle large datasets of sales data efficiently. The time taken to evaluate a commission plan expression or if-else condition should be reasonable even for large datasets.

3.4.2 Performance

The DSL and IDE should be designed with security in mind. They should not allow users to inject malicious code or access data that they should not have access to.

3.4.3 Usability

The DSL and IDE should be user-friendly and easy to learn. The syntax should be intuitive, and the error messages should be informative.

3.5 Conclusion

The DSL built using ANTLR lexer and parser for commission plans with infix expressions and if-else conditions should meet the functional and non-functional requirements defined in this software requirement specification. Additionally, the IDE should provide a user-friendly interface for writing, testing, and debugging

CHAPTER 4

SYSTEM DESIGN

4.1 Proposed System

The purpose of this system design is to provide an overview of the architecture for a domain-specific language (DSL) built using ANTLR lexer and parser for commission plans, and an integrated development environment (IDE) for writing, testing, and debugging commission plans. The DSL should support infix to prefix conversion and JSON conversion.

4.2 Architecture Overview

The architecture for the system will consist of the following components:

- Lexer and Parser: ANTLR lexer and parser will be used to parse commission plan expressions and if-else conditions defined by the users.
- Compiler: The compiler will compile the commission plan expressions and conditions into an intermediate representation, which can then be evaluated by the evaluator.
- Infix to Prefix Converter: The infix to prefix converter will convert infix commission plan expressions to prefix notation.
- JSON Converter: The JSON converter will convert commission plans to JSON format.
- IDE: The IDE will provide a user-friendly interface for writing, testing, and debugging commission plans. It will communicate with the Lexer and Parser, Compiler, Evaluator, Infix to Prefix Converter, and JSON Converter as needed.

4.2 Data Workflow

The data flows through several stages, each stage processing the input data in a specific way to produce the output data. Here is a high-level overview of the flow of data through a compiler:

- Lexical Analysis:

  The first stage of the compiler is the Lexical Analysis phase, where the input code is broken down into a stream of tokens. A token is a sequence of characters that represents a single entity, such as a keyword, identifier, or operator.

- Syntax Analysis:

  The second stage is the Syntax Analysis phase, where the tokens generated by the Lexical Analysis phase are parsed into a tree-like structure called the Abstract Syntax Tree (AST). The AST represents the structure of the input code in a way that is easier to process than the original source code.

- Semantic Analysis:

  The third stage is the Semantic Analysis phase, where the AST is checked for semantic errors. This includes checking for correct variable declarations and usage, type checking, and other language-specific checks.

- Intermediate Code Generation:

  The fourth stage is the Intermediate Code Generation phase, where the AST is translated into an intermediate representation that is closer to the machine language. The intermediate code may be in the form of bytecode, assembly code, or some other form that is easier to execute.

- Code Generation:

  The final stage is the Code Generation phase, where the optimized intermediate code is translated into machine code that can be executed by the target processor. The machine code may be in the form of object files or executable files, depending on the specific compiler and target platform.

CHAPTER 5

SYSTEM IMPLEMENTATION

5.1 Module 1 : Lexer and Parser

The Lexer and Parser will be implemented using the ANTLR lexer and parser generator tool. ANTLR will be used to generate lexer and parser code in Java, which will be used to tokenize the input commission plan expression and generate the parse tree respectively.

The commission plan language grammar will be defined using a Context-Free Grammar (CFG) in ANTLR. The CFG will specify the syntax rules for the commission plan language and will be used by ANTLR to generate the parser code.

5.2 Module 2 : Infix to Prefix Converter

The Infix to Prefix Converter will be implemented as a Java class that takes in a parse tree as input and generates the equivalent commission plan expression in prefix notation.

The Converter will use a stack-based algorithm to traverse the parse tree in infix notation and generate the equivalent expression in prefix notation.

5.3 Module 3 : JSON Converter

The JSON Converter will be implemented as a Java class that takes in the commission plan expression in prefix notation as input and generates the commission plan structure and variables in JSON format.

5.4 Module 4 : IDE

The IDE will be implemented as a Visual Studio Code (VSC) plugin using the VSC extension API.

The plugin will provide an editor interface for users to enter commission plan expressions and debug them.

The plugin will use ANTLR to generate the lexer and parser code and the CFG for the commission plan language grammar.

The plugin will use the Infix to Prefix Converter and JSON Converter classes to convert the commission plan expression to prefix notation and generate the commission plan structure and variables in JSON format.

The plugin will use the VSC debugging API to provide debugging functionality, such as setting breakpoints, stepping through expressions, and inspecting variables.

The plugin will also integrate with source control systems, such as Git, for versioning and collaboration.

Finally, the plugin will allow users to export and import commission plans in a variety of formats, such as CSV, JSON, and XML.

5.5 Diagrams

5.5.1 Data Flow Diagram

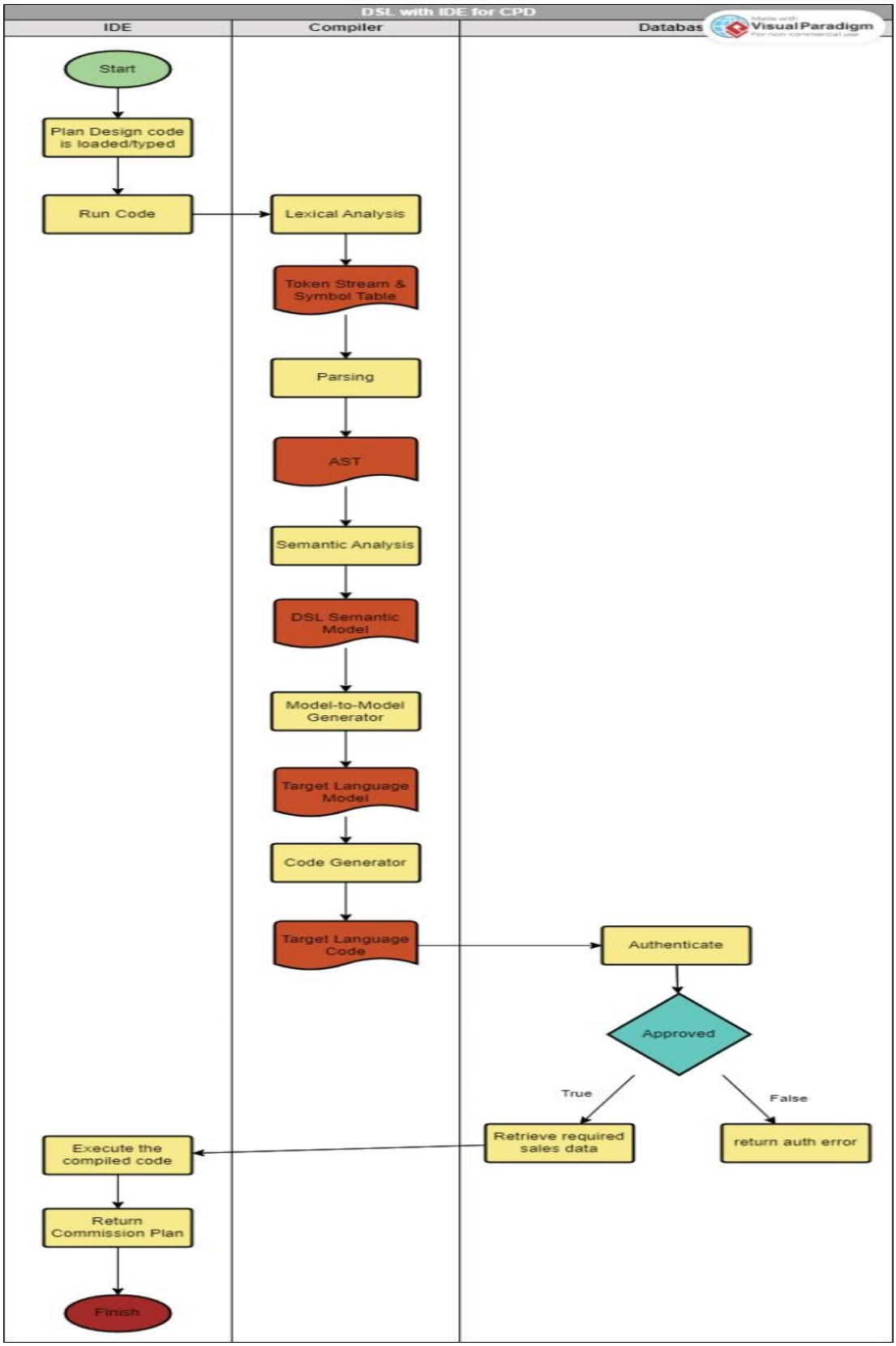


Fig.5.1: Dataflow Diagram

5.5.2 Component Diagram

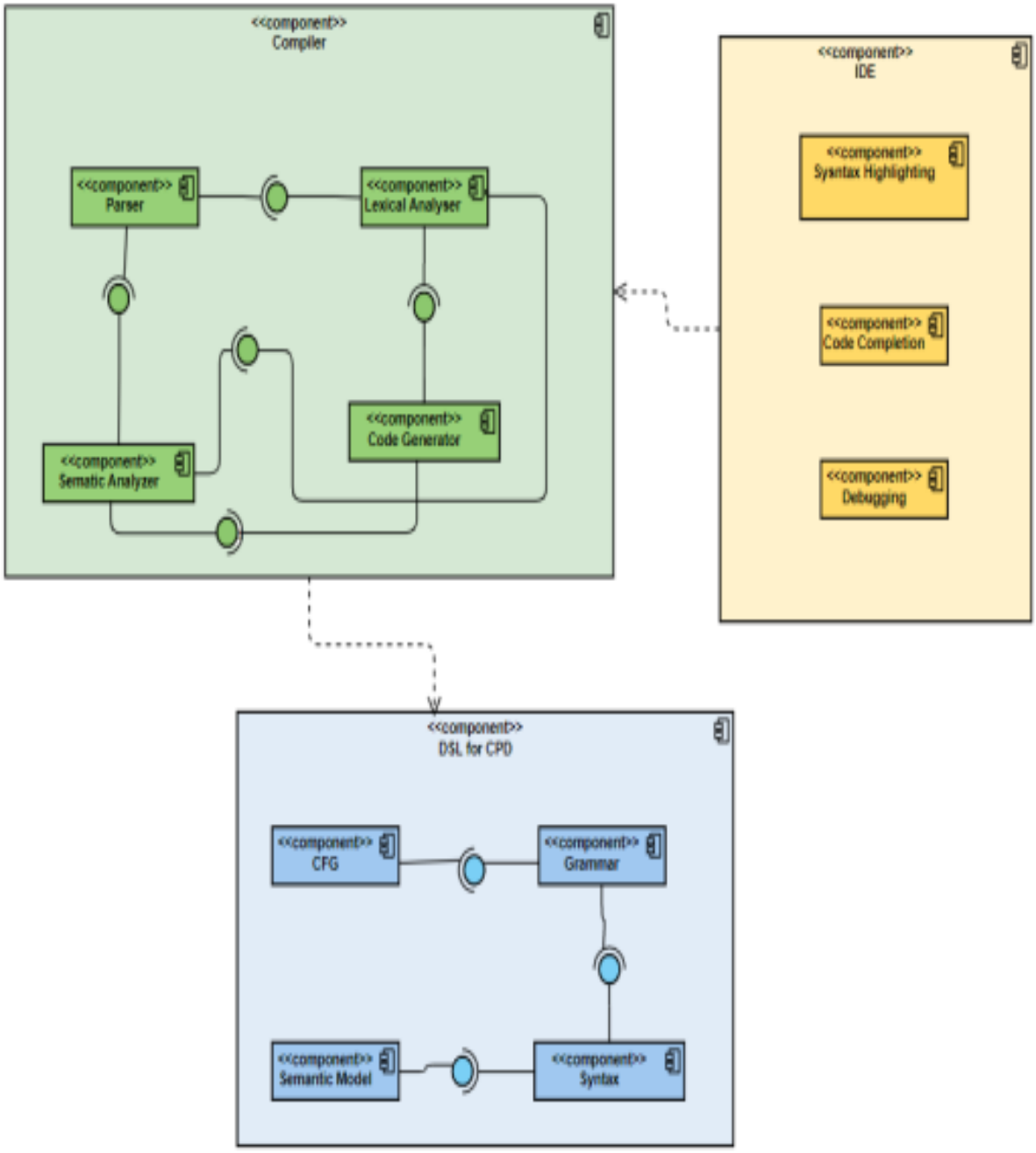


Fig.5.2: Component Diagram

5.5.3 Entity-Relationship Diagram

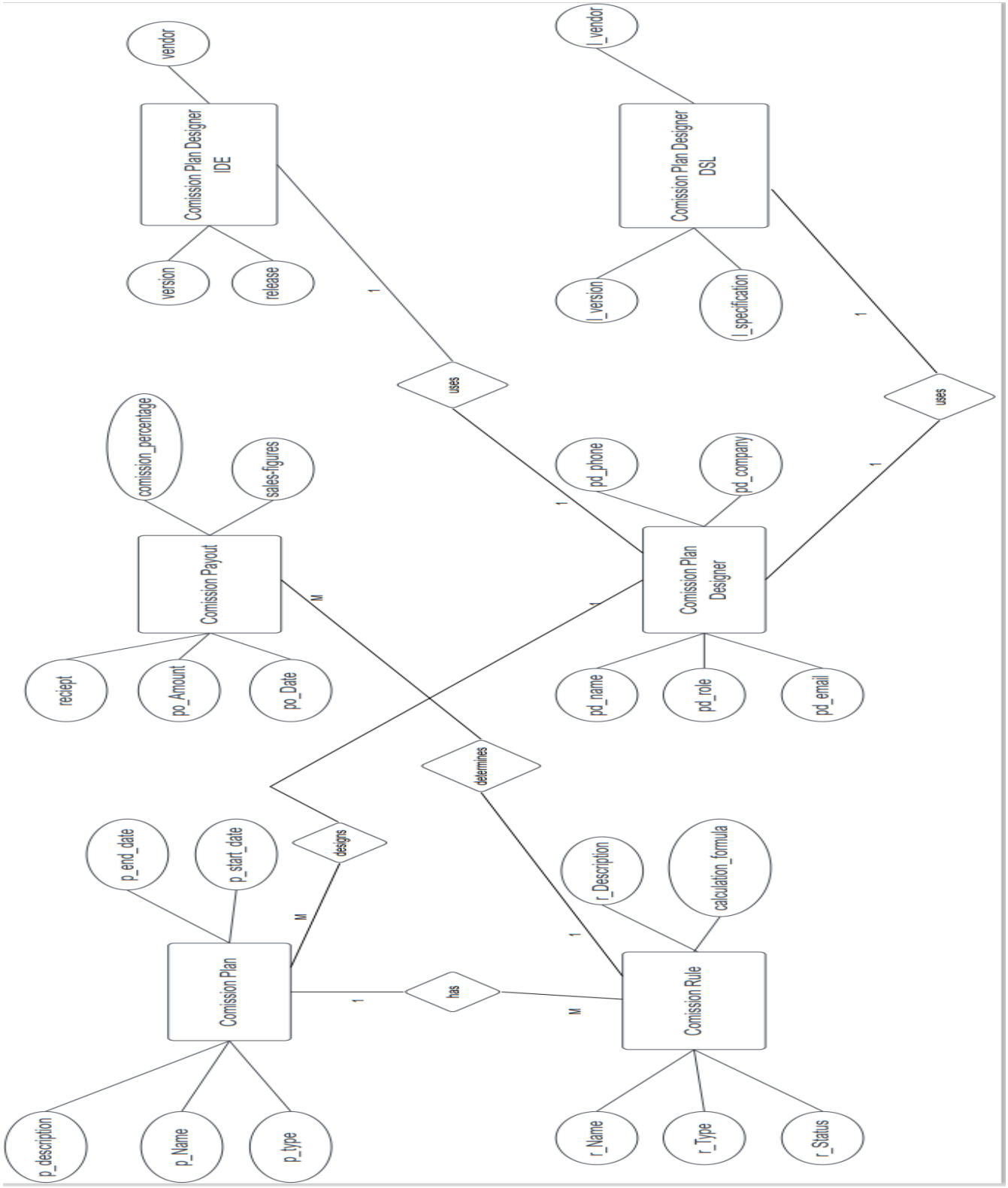


Fig.5.3: ER Diagram

CHAPTER 6

PERFORMANCE ANALYSIS AND TESTING

6.1 Performance Evaluation

This focuses on the process of analyzing the performance of a domain-specific language. It would cover techniques such as profiling, benchmarking, and tracing, which can be used to identify bottlenecks and areas for optimization. It would also cover different performance metrics and how they can be measured and analyzed to gain insights into the performance of the language.

6.2 Testing

This covers the process of testing a domain-specific language. It would include designing and implementing test cases, creating test scripts or programs, and executing tests to validate the correctness and functionality of the language. It would also cover different types of testing, such as unit testing, integration testing, and system testing, and how they can be used to ensure the quality of the language.

6.3 Testing Evaluation

This focuses on analyzing the results of tests performed on a domain-specific language. It would cover techniques such as statistical analysis, data visualization, and data mining, which can be used to identify patterns and trends in test results. It would also cover how testing analysis can be used to identify areas for improvement in the language, as well as to validate the performance and functionality of the language.

CHAPTER 7

CONCLUSION

In conclusion, the creation of a DSL and VS Code extension with code completion, syntax highlighting, and debugging capabilities for commission plan designers is a powerful tool that simplifies the process of creating and managing commission plans. By focusing on the specific needs of commission plan designers, this project aims to increase productivity, accuracy, and ultimately, better business outcomes. The DSL and VS Code extension provide a streamlined, user-friendly experience that allows designers to focus on high-level concepts and rules, rather than low-level syntax. With thorough testing, refinement, and comprehensive documentation, this project has the potential to revolutionize commission plan design, making it more efficient and effective for businesses worldwide.

CHAPTER 8

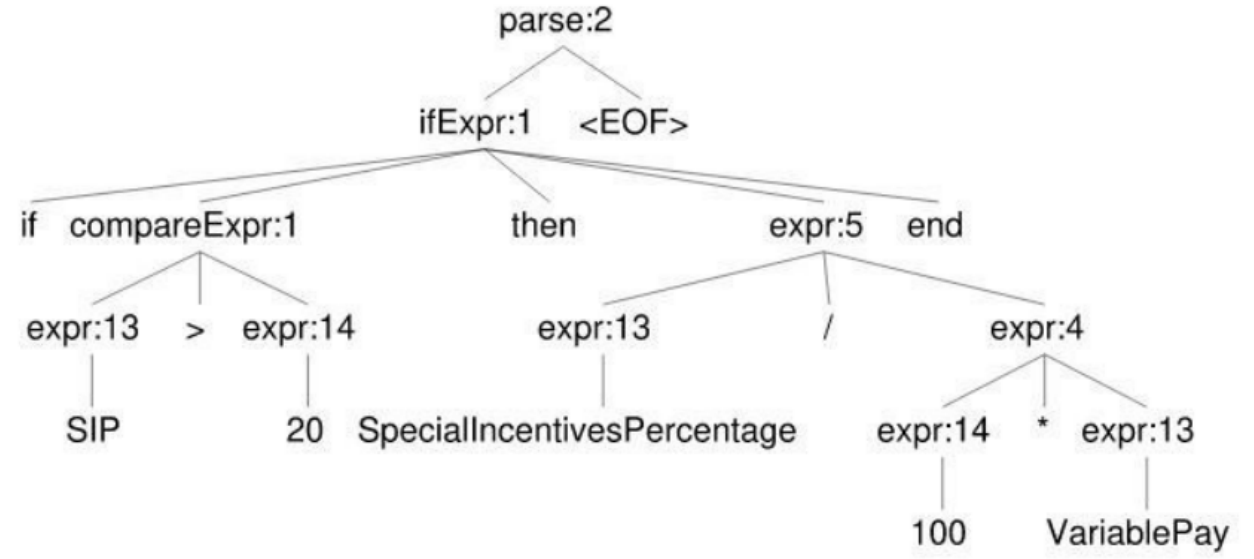
APPENDICES

8.1 Output Screenshots

```
RULES
    rulestart

        if SIP > 20 then
            SpecialIncentivesPercentage / 100 * VariablePay
        end

    ruleend
RULESEND
```


{
"if": [
{
"type": 1,
"value": "if"
},
{
">": [
{
"args": [
{
"type": "VARIABLE",
"value": "SIP"
}
]
},
{
"args": [
{
"type": "INTEGER",
"value": "20"
}
]
}
]
},
{
"type": 2,
"value": "then"
},
{
"/": [
{
"args": [
{
"type": "VARIABLE",


"args": [
{
"type": "VARIABLE",
"value": "SpecialIncentivesPercentage"
}
]
},
{
"*": [
{
"args": [
{
"type": "INTEGER",
"value": "100"
}
]
},
{
"args": [
{
"type": "VARIABLE",
"value": "VariablePay"
}
]
}
]
}
]
}
]
}

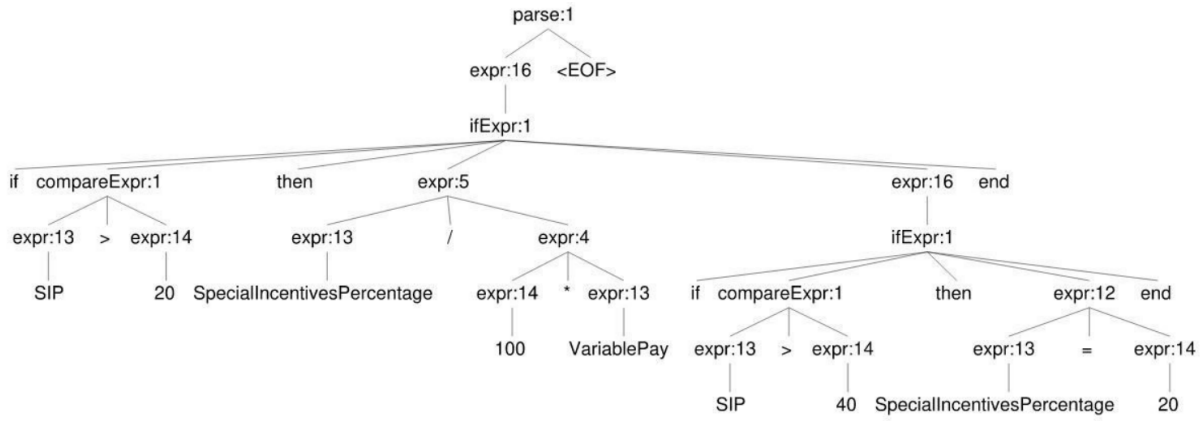

parse:2
ifExpr:1
<EOF>
if
compareExpr:1
then
expr:5
end
expr:13
>
expr:14
expr:13
/
expr:4
SIP
20
SpecialIncentivesPercentage
expr:14
*
expr:13
100
VariablePay

```
RULES
    rulestart

        if SIP > 20 then
            SpecialIncentivesPercentage / 100 * VariablePay
            if SIP > 40 then
                SpecialIncentivesPercentage = 20
            end
        end

    ruleend
RULESEND
```


"/": [
{
"args": [
{
"type": "VARIABLE",
"value": "SpecialIncentivesPercentage"
}
]
},
{
"*": [
{
"args": [
{
"type": "INTEGER",
"value": "100"
}
]
},
{
"args": [
{
"type": "VARIABLE",
"value": "VariablePay"
}
]
}
]
}
]
},

```
"ifContext": [
  {
    "if": [
      {
        "type": 1,
        "value": "if"
      },
      {
        ">": [
          {
            "args": [
              {
                "type": "VARIABLE",
                "value": "SIP"
              }
            ]
          },
          {
            "args": [
              {
                "type": "INTEGER",
                "value": "40"
              }
            ]
          }
        ]
      },
      {
        "type": 2,
        "value": "then"
      },
```

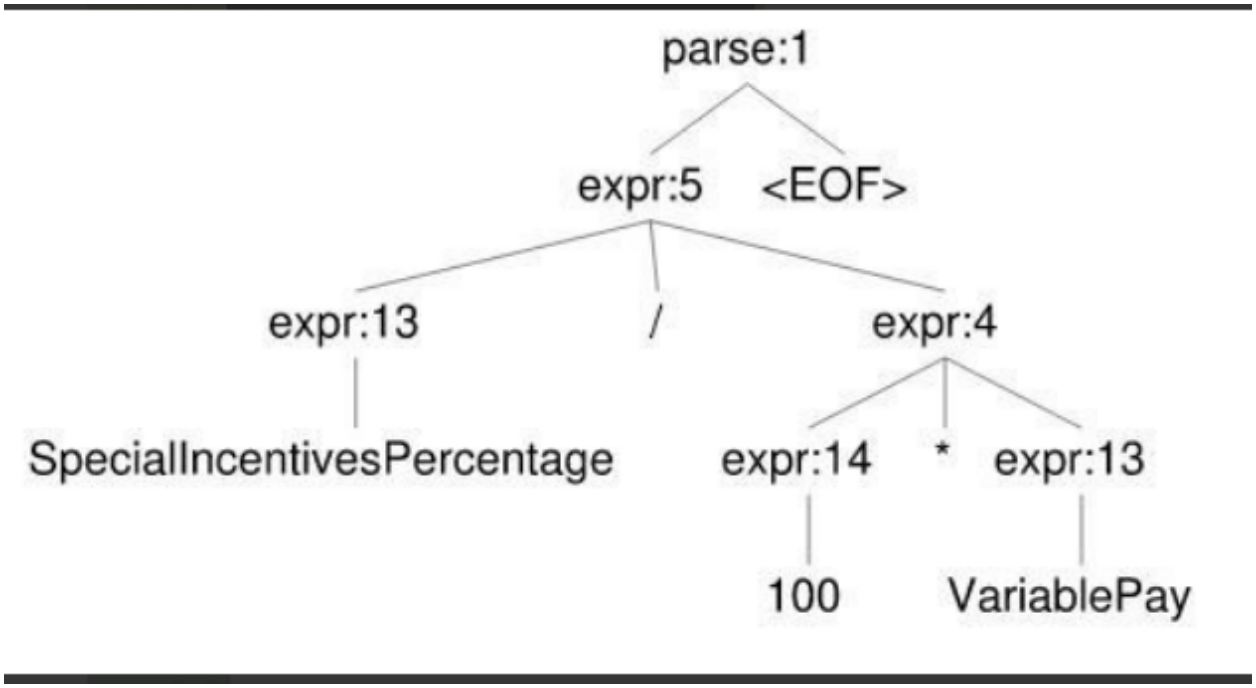
```
{
  "=": [
    {
      "args": [
        {
          "type": "VARIABLE",
          "value": "SpecialIncentivesPercentage"
        }
      ]
    },
    {
      "args": [
        {
          "type": "INTEGER",
          "value": "20"
        }
      ]
    }
```

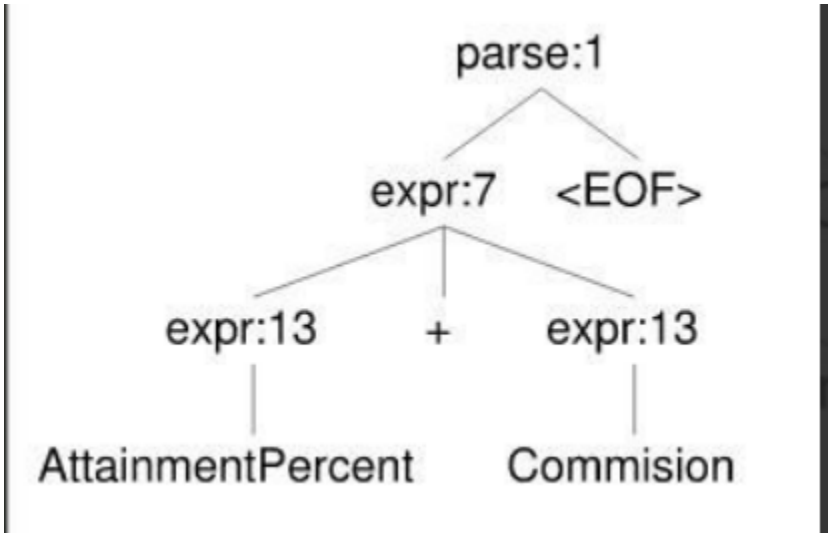



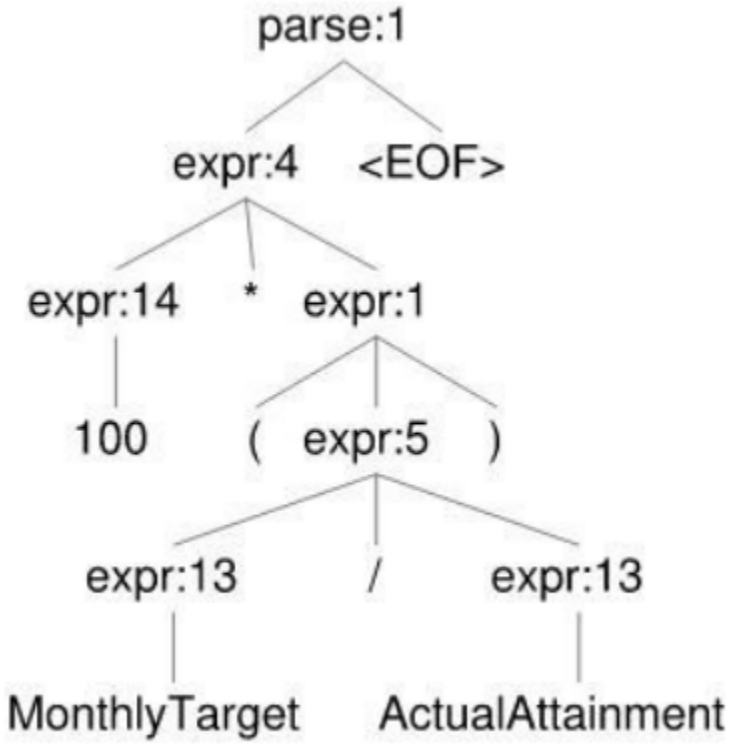
```
RULES
    rulestart
        SpecialIncentivesPercentage / 100 * VariablePay
    ruleend

    rulestart
        100 * (MonthlyTarget / ActualAttainment)
    ruleend

    rulestart
        AttainmentPercent + Commision
    ruleend
RULESEND
```


{
"/": [
{
"args": [
{
"type": "VARIABLE",
"value": "SpecialIncentivesPercentage"
}
]
},
{
"*": [
{
"args": [
{
"type": "INTEGER",
"value": "100"
}
]
},
{
"args": [
{
"type": "VARIABLE",
"value": "VariablePay"
}
]
}
]
}
]
}


{
"*": [
{
"args": [
{
"type": "INTEGER",
"value": "100"
}
]
},
{
"nested": [
{
"type": 11,
"value": "("
},
{
"/": [
{
"args": [
{
"type": "VARIABLE",
"value": "MonthlyTarget"
}
]
},
{
"args": [
{
"type": "VARIABLE",
"value": "ActualAttainment"
}
]
}
]
},
{
"type": 12,
"value": ")"
}
]
}
]
}


parse:1
expr:5
<EOF>
expr:13
/
expr:4
SpecialIncentivesPercentage
expr:14
*
expr:13
100
VariablePay


parse:1
expr:7
<EOF>
expr:13
+
expr:13
AttainmentPercent
Commision


parse:1
expr:4
<EOF>
expr:14
*
expr:1
100
(
expr:5
)
expr:13
/
expr:13
MonthlyTarget
ActualAttainment