Project Part 5: Generation & Final Submission

Amr Ghazzali, Baghdad Jamai Ghali, Ihab Kassimi, Ayman ElAkkaoui

Introduction

The objective is to study the design and implementation of a small domain-specific programming language, called Cleaning World Language. It allows a virtual cleaning agent to move within a world, cleaning it, by providing control structures like sequencing, conditionals, and loops.

Part 5 of the project represents the last phase of the compiler pipeline. The idea is to perform partial compilation in order to get the language executable and, at the same time, integrate all previously developed parts of the assignment. This submission combines lexical analysis, parsing, static semantics, AST construction, and execution, while improvements based on feedback given for Parts 1–4 have also been taken into account, for this we chose direct translation from the Abstract Syntax Tree, as the project specification allows.

Language Overview

The Cleaning World Language is designed to be simple and expressive.

It focuses on:

*A single agent acting in a world

*Action commands - movement, cleaning, etc.

*Control flow constructs-conditional statements and loops

The language was designed small so that clarity of syntax and semantics were emphasized over complexity in the language.

Chosen Approach: Direct Interpretation from the AST

Interpret and execute directly from the AST created in Part 4 using a higher-level language. Instead of producing pseudo-code, the program is executed by Python's traversing the AST. Every node in the AST represents one semantic construct of the language and contains the logic needed for the execution of that node.

This approach allows the compilation process to be completed in a clear and direct way, closely following the execution model discussed in class.

Execution Model and Function Calls

The interpreter follows a stack-based execution model, inspired from the function call mechanism studied in CSC 3315.

When execution enters a new executable block or function:

*A new activation record is conceptually created

*Local scope of the executed context is stored

*Control is transferred to the block body

When execution is complete:

*The activation record is popped

*Control returns to the calling context

To avoid making things more complicated than necessary, the Cleaning World Language eschews:

*Recursive function calls

*Non-local references

This design decision evades the necessity for complicated chains of environments while remaining loyal to the theoretical model of stack-based execution.

The built-in commands for movement and cleaning are treated as primitive operations-similar to macros-and are executed directly without generating additional stack frames.

Compilation and Execution Pipeline

The entire pipeline of the project includes the following steps:

Lexical Analysis

The lexical analyzer reads a source file written in the Cleaning World Language and converts it into a stream of tokens. Some of the tokens include:

*Key words
*Identifiers
*Numeric literals
*Symbols and delimiters

Lexical errors are detected here and reported before parsing.

Parsing and Grammar Validation

The parser processes the token stream according to the grammar of the language. It checks if the program obeys the syntactical rules, and creates:
*A concrete syntax tree reflecting the grammar structure
*An Abstract Syntax Tree (AST) representing the meaning of the program
The purpose of this step is to make sure that only syntactically valid programs go to execution.

Static Semantics

During parsing and AST construction, the following basic static semantic checks are performed:
*Correct structure of control statements
*Valid sequencing of commands
*Proper usage of language constructs
These checks ensure that the programs which are invalid do not get executed.

Construction of AST

Each syntactic construct has its AST node that represents it. Some examples include:
*Program nodes
*Statement nodes
*Control structure nodes - if-statements and loops
*Action nodes include moving and cleaning actions.
The AST is the intermediate representation that the interpreter uses.

Direct Interpretation and Execution

The interpreter walks down the AST and carries out each node in consequence with its semantic definition. In execution the position of the agent is updated and the world state is changed then Control flow decisions are evaluated.
This stage is the final stage in the elaboration process and source code behaves identically.

Examples and Testing

We have different example programs that are included with the project.

All these examples cover all control structures of the Cleaning World Language:
*Sequential execution
*Conditional statements

*Iteration using loops
Each example program was tested through the complete compilation and execution pipeline:
*Source Code
*Lexical Analysis (Lexer)
*Parsing and Grammar Validation (Parser)
*Abstract Syntax Tree Construction (AST)
*Direct Interpretation and Execution

Testing Table

sample.clean

Feature tested: Basic actions
Lexer: works
Parser: works
AST: works
Interpreter: works

test_simple.clean

Feature tested: Control flow
Lexer: works
Parser: works
AST: works
Interpreter: works

test_world_agent.clean

Feature tested: Full execution
Lexer: works
Parser: works
AST: works
Interpreter: works

The generated CST and AST outputs confirm that grammar rules and semantic structures are correctly applied.

The Upgrades and fixes from previous parts

Some modifications were introduced based on the responses received for Parts 1 and 2 and what we think was wrong in part 3 and 4:

*Rules of grammar were simplified and explained

*The AST structure was improved for a clearer separation of concerns.

*Interpreter logic became more robust

*Documentation and test cases were increased

These changes allowed for a more consistent and reliable system.

Putting it all Together

This final submission combines all phases of the compilation process:

*Lexical Analysis

*Parsing and grammar validation

*AST construction

*Immediate interpretation and implementation

Conclusion

We successfully implemented the Cleaning World Language by providing the ability to run the program from the AST. The interpreter captures important runtime concepts covered in the class by simulating the execution process performed on the stack while being relevant to the project scope.

As a whole, this project helped gain hands-on experience in language design, parsing, semantics, and interpreting, as well as supported understanding of the theoretical basis of compilers and interpreters.