Mansions of Madness (MoM) Adventure Language

Compiler Design

Prepared by: Marco A. Peyrot (A00815262)
mpeyrotc.github.io

Elias Arif Mera Ávila (A01280762)
a01280762@itesm.mx

Delivery Date: Wednesday November 22th, 2017

Contents

1 The Project

As part of course TC3048, Compiler Design, students must design and develop a compiler and virtual machine for a custom language of their creation. This text documents the entirety of the project specification, development, and results. This document is also accompanied by the source code, examples and scripts created for the purpose of completing the assignment, as well as with a reference to an online user's guide.

1.1 Purpose

The purpose of this project is to develop an Object-Oriented language to facilitate the creation of scenarios and components for the board game Mansions of Madness Second Edition [1]. Through the use of this language, owners will have the opportunity of creating additional content for the game and getting a bit into the field of computer programming. Therefore, a key emphasis of the language is to make it simple and comprehensible to promote its use and to give it more utility.

1.2 Objectives

The language's main area is education since it aims to promote learning general programming through an easy to use object-oriented language. The language accomplishes this by letting users develop and play scenarios and stories for the board game Mansions of Madness Second Edition. However, to keep the project manageable enough for a two-month project, the execution of a program will be similar to that of playing an old text-based game from the 1980's.

Because the language was made to promote an easy and engaging way to promote the art of computer programming, its basic form is just the beginning of what could be a major project than what the scope of this academic project holds. As a result, it is hoped that in the future, this project will have both graphical input and output to facilitate game creation and language use.

1.3 Project Scope

This project encompasses the creation of a limited, general purpose, object oriented programming language. By creation we mean the definition of the language, a parser for the source code, and a virtual machine capable of interpreting and running the intermediate code generated by the parser.

The compiler is designed to run via the terminal and requires the directory of the file that contains the source code as a supplement. To facilitate the compiler design and implementation, all the required source code must be present in the same file. The compilation of a program created in this language is similar to that of a `C++` program. Lexical, syntactic, and semantic analysis are made by the parser and this process generates a temporary file that holds intermediate code. This file is always named `temp.obj`. The file is then fed to the virtual machine in a similar fashion.

The language has three main structures: classes, specifications, and *enums*. A class is the template for a real-world abstraction, it can have methods and properties (fields). Classes are the building blocks of the language, in fact, the MoM Adventure Language is a pure object oriented language like Java, from which it is largely based on. As a result, every `.mom` program must have at least a single class and a main method, which is the entry point for the program execution. Specifications act as Java interfaces, enabling a limited type of multi-inheritance. Classes can implement any number of interfaces, but this is not mandatory. Finally, *enums* are syntactic-sugar for magic numbers with an 'extra' meaning. They are based on and act as classic `C` *enums*.

1.4 Requirement Analysis and Use Cases

The project requirements can be classified into two groups; one regarding the compiler as the project for course TC3048, and the second corresponding to what the user of the language can do with it. On such terms, the requirements for the compiler as a project are:

- Lexical validation of the source code during scanning.
- Syntactic validation of the source code during parsing.
- Semantic validation of the source code during parsing.
- Variable and constant table management.
- Quadrupole creation for intermediate code generation.
- Intermediate code interpretation and execution.
- Final delivery of working product with corresponding documentation.

On the other hand, the functional requirements of the language are defined below:

- Declaration of variables for both primitives and classes (discussed in section 2.2).
- Use of constants for the four primitive types.
- Printing any of the four primitives via the terminal.
- Reading any of the four primitives via the terminal and assigning them to an appropriate variable.
- Creation of cycles through the use of the `while`.
- Use of arithmetic an logical expressions.
- Conditional branching through the use of the `if` and `if-else` statements.
- Class instantiations through constructor calls.

- Method calls defined within a class, obtained through the parent, or through another class via composition.
- Single inheritance for new classes.
- Specification creation and implementation.
- Error reporting during compilation and runtime phases.
- Creation and use of *enums*.
- Method execution interruption through the use of the `return` statement.
- Declaration and use of arrays.
- Passing of constants and variables (primitive and complex) as method arguments (except arrays).

1.5 Test cases

Find	array_find.mom	Creates an array and searches for two numbers, one is present.
Sort	array_qsort.mom	Creates an array, prints it. implements quicksort and reprints the sorted array.
Complex class	complex_class.mom	Mix of several specific tests like two classes, specification, enumeration, etc.
Conditions	conditions.mom	Tests nested if's, else's.
Constructors	constructors.mom	Creates 2 objects of different classes and tests constructor calling and initialization.
Enumeration	enumeration.mom	Tests implementation and use of enumeration.
Expressions	expressions.mom	Tests simple expressions.
Factorial	factorial.mom	Implements recursive and iterative Factorial of a number. You give a number N and returns its factorial.
Fibonacci	fibonacci.mom	Implements recursive and iterative Fibonacci function. You give a number N and returns the n^{th} Fibonacci number.
Functions	functions.mom	Tests different type of functions with and without return value.
Implicit constructors	implicit_constructors.mom	Implements implicit constructors in three classes (default constructor).
Matrix Product	matrix_multiplication.mom	Creates 2 matrices and implements matrix multiplication on a third one and displays the result.
Returns	returns.mom	Tests correct generated returns Quadruples.
Single Class	single_class.mom	Tests functionality of a single class and including other specific tests like: functions, returns, etc.
Specification	specification.mom	Just tests the syntax of a specification without implementing it.
Specifications	specifications_in_class.mom	Implements a class Character with specification of Card and ComplexNumber.
Interfaces	test_interfaces.mom	Tests functionality of interfaces creating an object of type ComplexClass but implementing class “AClass” .
Two classes	two_classes.mom	Tests a file containing two classes.

Table 1: MoM Adventure Language Tests.

1.6 Development Log

The project was developed during the course of two months, from September 12^{th} to November 22^{nd}, according to the development plan passed to us by the course instructor. For the vast majority of the project, this schedule was followed to the letter and no additional tasks than those proposed were needed. The general schedule used is shown in table 2. Work was divided between the the members of the team. The source code can be separated into six disjoint parts. These are discussed below:

1. Grammar: the grammar was defined, *refactored* and improved by both teammates according to the task at hand.

2. Listener: the listener is a conglomeration of methods designed to execute custom code as the parser reads through the source code. Main development was done by Marco A. Peyrot during the initial phases of the project. Then, Elias Mera became more involved with and by the end he was the responsible person for the file.

3. Structures: represent the various helper classes and functions design to aid in the parsing of the source code, controlling the parsing process, internal representations for things like variables, methods, quadrupoles, to name a few, and validation of the correctness of the source code. Marco A. Peyrot was the main responsible for this part.

4. Semantic Table: the creation and maintenance of the semantic table was Marco A. Peyrot's responsibility.

5. Virtual Machine: the design and development of the virtual machine was made by both teammates during the same period of time. By the end of the project, this task fell on Elias Mera.

6. Unit tests: since development was done on different operating systems, each team member was in charge of conducting their tests. However, they had more or less the same phases. At first, they tested for correct parsing and obtaining of data; then, they started to check only for correct compilation as quadrupoles were constantly being added or modified. Unit tests for execution were done manually.

The source code was managed through Git and stored on Github. No branches were used, so every change and commit is on the master branch. To have more control over the development process and not have to carry a separate log too, commits followed a specific format.

Commits were divided into three sections, which are described below. The second and third sections are identified by having a line with just `Summary:` or `Issues:` respectively; These sections must be followed by their content and written in that order.

1. Title: a short summarized description of the commit.

2. Summary: a more detailed explanation of what was done for the commit (optional).

Week	M	W	F	Milestone	Expected Delivery Content
September	18	**20**	22	#1	Scanner and parser.
September	25	**27**	29	—	Semana i.
October	2	**4**	6	#2	Basic variable semantics: procedure directory and variable table.
October	9	**11**	13	#3	Basic expression semantics: semantic cube, intermediate code generation for arithmetic expressions and sequential statements.
October	16	**18**	20	#4	Intermediate code generation for cycles and conditional statements.
October	23	**25**	27	#5	Function calls intermediate code generation.
Oct - Nov	30	**1**	3	#6	Memory map for virtual machine and program execution. Execution of arithmetic expressions and sequential statements.
November	6	**8**	10	#7	Conditional statement execution and intermediate code generation for arrays.
November	13	**15**	17	#8	Documentation start. Final details for each language in particular.
November	20	**22**	24	FINAL	Project delivery as a whole at 12:00 p.m.

Table 2: Semantic rules for expression types.

3. Issues: bugs encountered, tasks that need to be done, issues that must be changed or fixed, etc. go in this part.

The main reason for this format is that it enables the script shown in listing 1 to generate a prettified log report from the commits in that repository. The last log report generated is shown in appendix 1.

```python
import os
import subprocess

git_log_command = "git log --decorate=no --no-merges --abbrev-commit --pretty=fuller " \
                  "--date=format:'%Y-%m-%d %H:%M:%S' > raw_data.txt"
# start the code
content = r"""
\documentclass[margin=1in, 12pt]{article}
\usepackage[utf8]{inputenc}
\usepackage{fancyhdr}
\usepackage{amsmath}
\usepackage{tabularx,booktabs}
\usepackage{array}
\usepackage{graphicx}
\usepackage{tcolorbox}
\usepackage{minted}

\newcolumntype{L}{>{$}l<{$}}
\newcolumntype{C}{>{$}c<{$}}
\newcolumntype{R}{>{$}r<{$}}
\newcommand{\nm}[1]{\textnormal{#1}}

\addtolength{\topmargin}{-.875in}
\addtolength{\textheight}{1.75in}

\pagestyle{fancy}
\fancyhf{}
\rhead{MoM Adventure Language}
\lhead{Log Registry}
\rfoot{Page \thepage}

\begin{document}
    \tcbset{colback=black!5!white,colframe=black!5!white}
    \tcbsetforeverylayer{colframe=gray!50!blue}"""
```

```

block = r"""
    \begin{{tcolorbox}}[title=Commit: {0}, subtitle style={{boxrule=0.4pt,
        colback=yellow!50!red!25!white}}, coltext=black!75!black,
        coltitle=white!75!white]

        \textbf{{Author}}: {1}

        \textbf{{Commit}}: {2}

        \textbf{{Date}}: {3}

        \vspace{{0.2em}}
        \begin{{tcolorbox}}[leftrule=3mm, colback=black!5!white,
            colframe=black!60!white]
        {4}
        \end{{tcolorbox}}"""

block_summary = r"""
\begin{{tcolorbox}}[leftrule=3mm, colback=black!5!white,
            colframe=green!50!gray]
        \textbf{{Summary:}} {0}
\end{{tcolorbox}}"""

block_issues = r"""
\begin{{tcolorbox}}[leftrule=3mm, colback=black!5!white,
            colframe=red!60!white]
        \textbf{{Issues:}} {0}
        \end{{tcolorbox}}"""

blocks = []


def create_block(git_hash, author, commit, date, title, summary, issues):
    blocks.append({
        "git_hash": git_hash,
        "author": author,
        "commit": commit,
        "date": date,
        "title": title,
        "summary": summary,
        "issues": issues
    })


def create_log():
    gen_raw_data = subprocess.Popen([git_log_command], shell=True)
    gen_raw_data.wait()

    with open("raw_data.txt", 'r') as f:
        git_hash = ""
        author = ""
        commit = ""
        date = ""
        in_summary = False
        in_issues = False
        summary = ""
        issues = ""
        title = ""
        first = True
        s_blank = False
        i_blank = False

        for line in f:
            line = line.replace("!@#$%^&*()", "SPECIAL_CODE")
            line = line.replace(")(*&^%$#@!", "REVERSE_SPECIAL_CODE")
            line = line.replace("^", "\^")
            line = line.replace("&", "\&")
            line = line.replace("\\", "\\textbackslash ")
            line = line.replace("[", "{[")
            line = line.replace("]", "]}")
            line = line.replace(">", "$>$")
            line = line.replace("_", "\_")

            if "commit" in line:
                if not first:
                    create_block(git_hash, author, commit, date, title, summary, issues)
                first = False
                git_hash = line[7:].strip()
                author = ""
                commit = ""
                date = ""
                in_summary = False
                in_issues = False
                summary = ""
                issues = ""
                title = ""
                s_blank = False
                i_blank = False

                continue

```

```
            if "Author:" in line:
                author = line[8:line.index('<')].strip()
                continue

            if "Commit:" in line:
                commit = line[8:line.index('<')].strip()
                continue

            if "CommitDate: " in line:
                date = line[12:].strip()
                continue

            if "AuthorDate:" in line:
                continue

            if "Summary:" in line:
                in_summary = True
                continue

            if "Issues:" in line:
                in_summary = False
                in_issues = True
                continue

            if in_summary:
                if s_blank:
                    summary += " \n"
                    s_blank = False
                if not len(line.strip()) == 0:
                    summary += line.strip() + " "
                else:
                    s_blank = True

                continue

            if in_issues:
                if i_blank:
                    issues += " \n"
                    i_blank = False
                if not len(line.strip()) == 0:
                    issues += line.strip() + " "
                else:
                    i_blank = True

                continue

            if not in_summary or not in_issues:
                if not len(line.strip()) == 0:
                    title += line.strip() + " "

        create_block(git_hash, author, commit, date, title, summary, issues)

    with open('mom_log.tex', 'w') as f:
        f.write(content)

        for block_dict in blocks:
            text = block.format(block_dict["git_hash"],
                                block_dict["author"],
                                block_dict["commit"],
                                block_dict["date"],
                                block_dict["title"])

            if len(block_dict["summary"]) > 0:
                text += block_summary.format(block_dict["summary"])

            if len(block_dict["issues"]) > 0:
                text += block_issues.format(block_dict["issues"])

            text += "\end{tcolorbox}\n"
            f.write(text)

        f.write("\n\end{document}")

    cmd = ['/Library/TeX/Root/bin/x86_64-darwin/pdflatex', '-shell-escape', 'mom_log.tex']
    proc = subprocess.Popen(cmd)
    proc.communicate()

    retcode = proc.returncode
    if not retcode == 0:
        os.unlink('mom_log.pdf')
        raise ValueError('Error {} executing command: {}'.format(retcode, ' '.join(cmd)))

    os.unlink('mom_log.tex')
    os.unlink('mom_log.log')


create_log()
```

Listing 1: create_log.py

1.7 Personal Reflection

For me, this project was of the best that I have had during my academic life due to its complexity and interest factor. It was very fun and intellectually stimulating. I think that this is probably very related to what I will be doing in my first years at my first job, so it was interesting to get a bit into the field. I liked the concepts that I learned and how it gives me greater insight into the inner workings of the computer. This knowledge might prove useful to me one day, so I am grateful for it.

Marco A. Peyrot

This project was a challenge in my academic life because of its complexity but also it was interesting because it integrates a lot of topics of many of my favorite classes like: Data Structures, Algorithms. Before this class, I had the idea that I had the idea of how a compiler worked, but with this project and class I realized that I was so wrong about it. I learned a lot and, for me, that's the best part of a project. I'm proud of the final result of our project because of the effort we put on it.

Elias A. Mera

2 The Language

2.1 Programming Language Name

The full name of the programming language developed as part of this project is: Mansions of Madness Adventure Language, which is abbreviated as MoM Adventure Language. As a result, the files written with this language must have the `.mom` extension.

2.2 Main Language Properties

The MoM Adventure Language supports basic arithmetic and logic expressions, reading from and writing to the terminal, basic decision statements like `if` and `if-else`, as well as iteration statements like `while`. In addition the the latter, the language supports single inheritance, class composition up to one level, and recursion.

The language comes with four basic primitive types and a special type which represents the `void` keyword in the `C` programming language, these are:

- `Text` : represents a sequence of characters. It is the name for the `string` type in languages like `Java` and `C`.
- `Int` : represents an integer number.
- `Real` : represents a real number. More specifically, the *double* or *float* types common to `C`-like languages.
- `Boolean` : represents the conditional values `TRUE` and `FALSE`.
- `Nothing` : represents no return value, equivalent to the `void` type in `C`-like languages.

2.2.1 Default Classes

In addition to the creative liberty the language provides to the user, it will provide some classes by default. These classes will serve as the base for further development and aid the programmer to start making as soon as possible new stories and scenarios for the game. As the user of the language gains more experience, he/she will be able to develop new game mechanics, rules, or physical components through more complicated programs and objects. As a result, the language should be able to support the creation of any type of text-based game.

- Component: this is the base class for the MoM Adventure Language. It represents a game component (like a card, mat, or token). All other objects in the language inherit from this class. Its fields are `width` and `height`.
- Face: represents the side of a *component*. It is a container for several *sections*.
- Card: represents a physical game card, it has two *faces* and each one is divided into *sections* such as title, image, description, cost, etc..
- Token: represents a physical game token or carton piece.
- Message: it is a simple container that comes with a title, message, and two types of input: via a numeric or text entry, or a series of options which the player can select.
- Player: it is a special instance of component since it can have several other components attached. Its base members and fields are those found on the player sheet in the real board game.
- Creature: it is a kind of special component since it has several additional actions related to it. These actions are *move*, *attack*, and *evade*.

Their attributes, methods, and relationships are shown in the class diagram shown in figure 1.

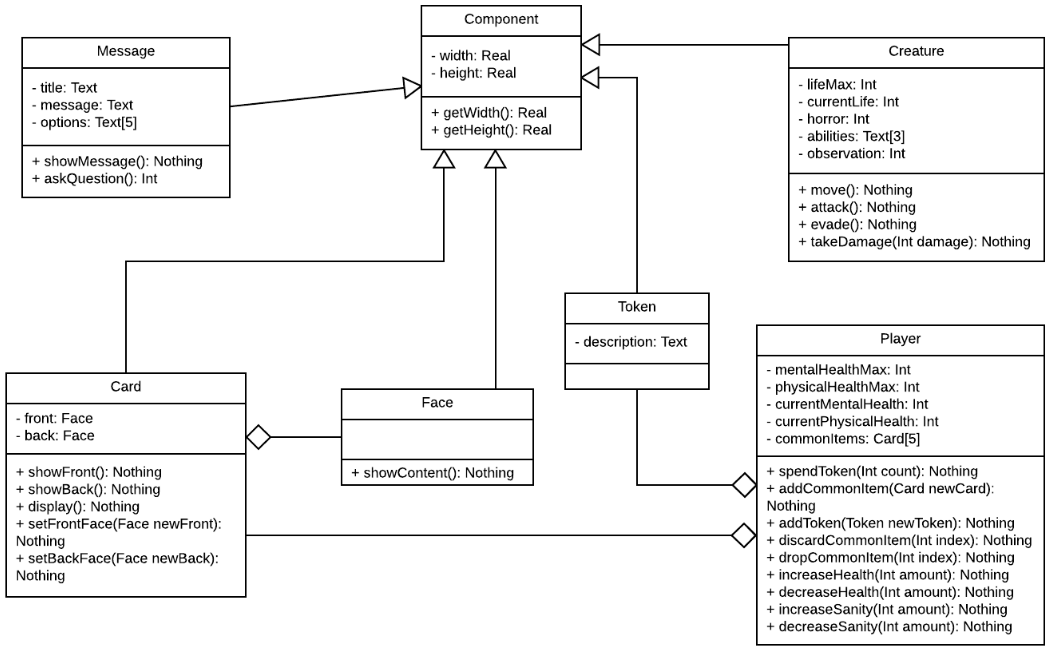

Figure 1: Default classes class diagram.

2.2.2 Special Functions

Just as the language comes with default classes to aid the programmer, it comes with some helper functions. These are listed below:

- `Write`: this special function is used to print into the standard output a stream of characters which are not followed by a default newline character.

```
Nothing Write(Text text);
Nothing Write(Int number);
Nothing Write(Real number);
Nothing Write(Boolean value);
Nothing Write();
```

- `WriteLine`: this special function does the same as the `Write` function, but it adds the newline character at the end of the stream. It has two versions, one with arguments and one without.

```
Nothing WriteLine(Text text);
Nothing WriteLine(Int number);
Nothing WriteLine(Real number);
Nothing WriteLine(Boolean value);
Nothing WriteLine();
```

- `ReadText` : this function reads from the standard input a stream of characters and stores them in a `Text` variable.

```
Nothing ReadText(Text variable);
```

- `ReadInt` : this function reads from the standard input a stream of numeric characters and stores them in an `Int` variable.

```
Nothing ReadInt(Int variable);
```

- `ReadReal` : this function reads from the standard input a stream of numeric characters (and the '.' character) and stores them in a `Real` variable.

```
Nothing ReadReal(Real variable);
```

- `ReadBoolean` : this function reads from the standard input a stream of alphanumeric characters that form the words `True` or `False` and stores them in a `Boolean` variable.

```
Nothing ReadBoolean(Boolean variable);
```

2.3 Errors thrown by the Language

2.3.1 Compilation Errors

The possible errors that can occur during compilation are:

1. TypeError("Local variable assignment not supported for OTHER (1).")
2. TypeError("Local variable assignment not supported for <INVALID> (2)")
3. TypeError("Global variable assignment not supported for OTHER (2).")
4. TypeError("No CLASS types supported for temporal variables (3)")
5. TypeError("Global increment for OTHER or <INVALID> not supported (6)")
6. raise RuntimeError("Main method not found, define program entry point.")
7. raise TypeError("Argument for method `x` call wrong.
8. NameError("Cannot assign type ")
9. NameError("Variable 'x' is undefined.")

10. `NameError("Redefinition of class")`

11. `TypeError("Wrong return type for method")`

These are just representative examples of similar errors.

2.3.2 Runtime Errors

During execution the following errors might occur:

1. `IndexError("No memory location defined.")`
2. `NameError("Cannot apply unary NOT to non-bool element")`
3. `RuntimeError("No class instance found.")`
4. `TypeError("Type cannot be printed.")`
5. `TypeError("Type cannot be verified.")`
6. `TypeError("Error, Out of bounds")`
7. `NameError("operation " + str(op) + " not recognized.")`

Again, these errors are not exhaustive, only representative.

3 The Compiler

3.1 Development Environment

As mentioned in section 1.6, the project was developed in the Windows 10 and Mac OS X *El Capitan* operating systems. The devices used had the following specifications:

- Mac OS X current version is 10.11.6 and is installed in a MacBook Pro with Retina display, 15-inch, Mid 2015) laptop. The device has a 2.5 GHz Intel Core i7 processor and 16 GB 1600 MHz DDR3 of RAM memory. Additionally, it has an Intel Iris Pro 1536 MB graphics card.

- Windows 10 installed on a Microsoft Surface Book laptop with a 6th Gen Intel Core i5 processor, 256GB SSD of internal memory and 8GB of RAM. It has a 13.5-inch PixelSense Display touchscreen and comes with a Surface Pen.

The project, compiler and virtual machine, was developed on the Python language version 3.6.1 as provided by the Anaconda package, version 4.4.0 (x86_64). The lexical and semantic analyzer that will be used to generate the scanner and parser is named Antlr for Python [2]. This tool must be installed if the grammar is to be changed. To run it, execute this command from the project directory:

```
$ antlr4 -Dlanguage=Python3 src/grammar/MoM.g4
```

However, keep in mind that this will erase the file `MoMListener.py`, so it is recommended to keep a copy and put it back after the command is run. The user must supply the missing listener functions that were added by the command above for each new rule added to the grammar.

3.2 Lexical Analysis

3.2.1 Language Tokens and Reserved Words

Since the base language is `Pyhton 3`, `Python 3's` reserved tokens [3] are also reserved for the MoM Adventure Language. Antlr also has its own set of reserved words, which must be accounted for. These are shown in tables 3, 4 and 5. In table 5 each element is divided into two parts (separated by the `:` character) the value on the left is the identifier used within the code, while the one on the right is the literal representation of the token.

Reserved Words				
False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	

Table 3: Python 3 Reserved Words.

Reserved Words				
import	fragment	lexer	parser	grammar
returns	locals	throws	catch	finally
mode	options	tokens	rule	

Table 4: Antlr Reserved Words.

3.2.2 Regular Expressions

Some tokens are more difficult to represent textually, so regular expressions are used to identify and classify them, these are shown in listing 2:

Reserved Words		
THIS: this	IF: if	ELSE: else
INT: Int	TEXT: Text	FLOAT: Real
NOTHING: Nothing	PRINT: Write	PRINT_LINE: WriteLine
READ_INT: ReadInt	READ_REAL: ReadReal	READ_TEXT: ReadText
READ_BOOL: ReadBoolean	CLASS: class	NEW: new
ENUMERATE: enumerate	FIELD: field	SPEC: specification
RETURN: return	BOOLEAN: Boolean	TYPE: type
OF_TYPE: of_type	IS_A: is_a	WHILE: while
TRUE: TRUE	FALSE: FALSE	

Table 5: MoM Adventure Language Reserved Words.

```
fragment DIGIT  : [0-9] ;
fragment UPPERC : [A-Z] ;
fragment LOWERC : [a-z] ;
CAPITALID    : UPPERC+ ;
CLASSID      : UPPERC (UPPERC | LOWERC | DIGIT | UNDERSCORE)* ;
VARID      : LOWERC (UPPERC | LOWERC | DIGIT | UNDERSCORE)* ;
INTEGER      : DIGIT+ ;
REAL       : DIGIT+ ([.,] DIGIT+)? ;
WHITESPACE     : ' ' -> skip ;
STRING       : '"' ( ~('\'' | '\\' | '\n' | '\r') ) + '"' ;
WS : [ \t\n\r]+ -> skip;
```

Listing 2: Language regular expressions (MoM.g4).

3.3 Operators and delimiters

The language also has several operators and delimiters recognized by the scanner, these are defined in table 6. Just as the previous table, each element is divided into two parts (separated by the : character) the value on the left is the identifier used within the code, while the one on the right is the literal representation of the operator.

<table>
<tr><th colspan="4">MoM Language Operators</th></tr>
<tr><td>OPEN_PAREN: (</td><td>CLOSE_BRACKET: }</td><td>CLOSE_PAREN:)</td><td>EQUALS: =</td></tr>
<tr><td>COMMA: ,</td><td>OPEN_SBRACKET: [</td><td>STAR: *</td><td>MINUS: -</td></tr>
<tr><td>SEMI_COLON: ;</td><td>CLOSE_SBRACKET:]</td><td>SLASH: /</td><td>PLUS: +</td></tr>
<tr><td>UNDERSCORE: _</td><td>OPEN_BRACKET: {</td><td>PERIOD: .</td><td>NOT: !</td></tr>
<tr><td>NOT_EQUALS: !=</td><td>LESS_THAN: <</td><td>LESS_EQUAL: <=</td><td>AND: &&</td></tr>
<tr><td>GREATER_THAN: ></td><td>GREATER_EQUAL: >=</td><td>EQUAL_EQUAL: ==</td><td>OR: ||</td></tr>
</table>

Table 6: MoM Adventure Language operators.

3.4 Syntax Analysis

Since the scanner and parser were made with one of the most widely used tools for this kind of projects, Antlr, the grammar was largely left intact thanks to the use of listeners. As a result, three separate files were created.

3.4.1 Grammar

The grammar is located in a `.g4` file alongside the tokens, which are put at the bottom of the file. Listing 3 contains only the syntactic rules.

```
program     : (class_rule
                |   enumeration
                |   specification)+ EOF
                ;
after_argument  :   // nothing
                ;
advance_count   :   // nothing
                ;
arguments   : ss_exp after_argument advance_count (COMMA ss_exp after_argument advance_count)*
                ;
assignation   : (THIS PERIOD)? (VARID | array_var) EQUALS (construct_call | ss_exp)
                ;
block     : statute SEMI_COLON
                ;
class_rule    : CLASS CLASSID IS_A complex_type (OF_TYPE CLASSID (COMMA CLASSID)*)? OPEN_BRACKET
                            field* construct_def function_def* CLOSE_BRACKET SEMI_COLON
                ;
exit_if_check   :   // nothing.
                ;
condition_end   :   // nothing.
                ;
enter_else      :   // nothing.
                ;
condition   : IF OPEN_PAREN ss_exp CLOSE_PAREN exit_if_check OPEN_BRACKET block* CLOSE_BRACKET
                            (ELSE enter_else OPEN_BRACKET block* CLOSE_BRACKET)? condition_end
                ;
constant    : INTEGER
                |   REAL
                |   STRING
                |   VARID
                |   array_var
                |   CAPITALID
                |   TRUE
                |   FALSE
                |   function_call
                ;
construct_call  : NEW CLASSID OPEN_PAREN (arguments)? CLOSE_PAREN
                ;
exit_con_def    :   // nothing
                ;
construct_def : CLASSID OPEN_PAREN CLOSE_PAREN OPEN_BRACKET block* CLOSE_BRACKET
                            exit_con_def SEMI_COLON
                ;
enum      : CAPITALID (COMMA CAPITALID)* SEMI_COLON
                ;
enumeration   : ENUMERATE CLASSID OPEN_BRACKET enum CLOSE_BRACKET SEMI_COLON
                ;
exit_sexp       :   // nothing
                ;
and_op          :   // nothing
                ;
or_op           :   // nothing
                ;
ss_exp      : s_exp exit_sexp ((AND and_op| OR or_op) s_exp exit_sexp)*
                ;
exit_exp        :   // nothing
                ;
s_exp       : expression (operand expression exit_exp)?
                ;
exit_term       :   // nothing
                ;
plus_op         :   // nothing
                ;
minus_op        :   // nothing
                ;
expression    : term exit_term ((PLUS plus_op | MINUS minus_op) term exit_term)*
                ;
open_paren      :   // nothing
                ;
close_paren     :   // nothing
                ;
factor      : OPEN_PAREN open_paren ss_exp close_paren CLOSE_PAREN
                |   (PLUS | MINUS | NOT)? constant
```

```
                ;
function_args : super_type VARID (COMMA super_type VARID)*
                ;
function_call : ((THIS | VARID) PERIOD)? VARID OPEN_PAREN (arguments)? CLOSE_PAREN
                ;
exit_func_def   :   // nothing
                ;
function_def  : simple_type VARID OPEN_PAREN (function_args)? CLOSE_PAREN OPEN_BRACKET block*
                            CLOSE_BRACKET exit_func_def SEMI_COLON
                ;
operand     : LESS_THAN
                |   LESS_EQUAL
                |   GREATER_THAN
                |   GREATER_EQUAL
                |   EQUAL_EQUAL
                ;
field           : FIELD LESS_THAN (super_type | array_def) GREATER_THAN VARID SEMI_COLON
                ;
spec_function   :   simple_type VARID OPEN_PAREN (function_args)? CLOSE_PAREN SEMI_COLON
                ;
specification : SPEC CLASSID OPEN_BRACKET spec_function* CLOSE_BRACKET SEMI_COLON
                ;
assignation_def :   super_type VARID EQUALS (construct_call | ss_exp)
                ;
statute     : function_call
                |   assignation
                |   assignation_def
                |   while_loop
                |   condition
                |   write_func
                |   write_line_func
                |   vdim
                |   read_int_func
                |   read_real_func
                |   read_text_func
                |   read_bool_func
                |   RETURN ss_exp
                |   RETURN
                ;
exit_factor     :   // nothing
                ;
star_op         :   // nothing
                ;
div_op          :   // nothing
                ;
term      : factor exit_factor ((STAR star_op | SLASH div_op) factor exit_factor)*
                ;
super_type    : simple_type
                |   complex_type
                ;
simple_type   : INT
                |   TEXT
                |   FLOAT
                |   NOTHING
                |   BOOLEAN
                ;
complex_type  : SET
                |   MAP
                |   SIZE
                |   CLASSID
                ;
end_while       :   // nothing.
                ;
after_while     :   // nothing.
                ;
while_loop    : WHILE after_while OPEN_PAREN ss_exp CLOSE_PAREN exit_if_check OPEN_BRACKET block* CLOSE_BRACKET
     end_while
                ;
array_def       :   super_type OPEN_SBRACKET INTEGER CLOSE_SBRACKET (OPEN_SBRACKET INTEGER CLOSE_SBRACKET)*
                ;
vdim            :   super_type OPEN_SBRACKET INTEGER CLOSE_SBRACKET (OPEN_SBRACKET INTEGER CLOSE_SBRACKET)* VARID
                ;
open_sbracket   :   //nothing
                ;
close_sbracket  :   // nothing
                ;
array_var       :   (THIS PERIOD)? VARID open_sbracket OPEN_SBRACKET ss_exp  close_sbracket CLOSE_SBRACKET (
     open_sbracket OPEN_SBRACKET ss_exp close_sbracket CLOSE_SBRACKET)*
                ;
write_func      :   PRINT OPEN_PAREN (ss_exp) CLOSE_PAREN
                ;
write_line_func :   PRINT_LINE OPEN_PAREN (ss_exp) CLOSE_PAREN
                ;
read_int_func   :   READ_INT OPEN_PAREN VARID CLOSE_PAREN
                ;
read_real_func  :   READ_REAL OPEN_PAREN VARID CLOSE_PAREN
                ;
read_text_func  :   READ_TEXT OPEN_PAREN VARID CLOSE_PAREN
                ;
read_bool_func  :   READ_BOOL OPEN_PAREN VARID CLOSE_PAREN
```

```
                    ;
```

Listing 3: MoM.g4

3.4.2 Error Listener

Antlr comes with a `Error Listener` which is used to check during parser for possible errors. The listener is shown in listing 4. Its errors will be reported during compilation.

```
from antlr4.error.ErrorListener import ErrorListener


class MoMErrorListener(ErrorListener):
    def __init__(self):
        super(MoMErrorListener, self).__init__()

    def syntaxError(self, recognizer, offendingSymbol, line, column, msg, e):
        raise Exception("Error in code detected, by: " + str(offendingSymbol) + ". More details: " + msg)

    def reportAmbiguity(self, recognizer, dfa, startIndex, stopIndex, exact, ambigAlts, configs):
        raise Exception("Ambiguity detected in grammar.")

    def reportAttemptingFullContext(self, recognizer, dfa, startIndex, stopIndex, conflictingAlts, configs):
        raise Exception("Should not happen.")

    def reportContextSensitivity(self, recognizer, dfa, startIndex, stopIndex, prediction, configs):
        raise Exception("Should not happen.")
```

Listing 4: Control Data Structures

3.4.3 Rule Listener

Antlr works with listeners instead of nerve points. For that it defines two listener methods for each defined rule. The syntax for the listener names is: `enterRuleName` and `exitRuleName` and the code inside them is executed by the parser before it enters the rule and after it exits the rule respectively. There are 26 nerve points in the grammar that are implemented with empty rules. These are discussed individually inside their corresponding parent rule[1].

enterProgram Creates the component base class and inserts the initial quadrupole that points to main.

exitProgram Inserts the `end` quadrupole that signifies the end of the intermediate code and checks that the main method was declared.

enterArguments Empty.

1. exitAfter_argument: creates the quadrupole that gets called before a method is invoked, it checks that the argument is valid and of the correct type as specified by the function.
2. exitAdvance_count: advances the argument counter so that the next expressions is matched to the appropriate parameter.

exitAssignation Creates the quadrupole that is equivalent to the assignation made by equal. It checks that the left and right arguments exist in the current context; if not it raises an error.

[1] If no listener method is defined for a rule, it means that the listener was left empty.

enterClass_rule Creates the corresponding class to store the relevant data. It is responsible for ensuring that the class name is unique, of initializing its internal data structures, of adding the specifications it implements, and to incorporate all the fields and methods obtained from its ancestors to its registry.

exitClass_rule This listener does two things. First, that all the methods in the specifications are defined. Second, it created the `end class` quadrupole to indicate the virtual machine that that is the end of the class definition.

enterCondition Empty.

1. enterExit_if_check: it checks that the evaluated expression is indeed of a boolean type and then it generates the `go to false` quadrupole.
2. exitCondition_end: sets the appropriate `jump` quadrupole so that after the `if` segment is executed the if *hops* over the `else` block.
3. exitEnter_else: sets the appropriate quadrupole, which is a `go to` one, where to jump when the conditional expression evaluates to false.

enterConstant determines the appropriate value and type of a constant and stores it in their corresponding stacks. It is worth noting that to avoid conflicts with the Text and the Int constants, True is represented as “!@#$%^&*()” and False is represented as “)(*&%^$#@!”. If it is a variable instead of a constant, it registers it in the correct context.

enterConstruct_call Checks that the constructor exists in its current context and determines if it belongs to the current class or another.

exitConstruct_call Creates the appropriate `constructor call` quadrupole and specifies if it is for the calling class or from another.

enterConstruct_def Resets the local memory counters, registers the constructor to its corresponding class and checks that there is no name conflicts. It inserts as part of the constructor call, as its first line, an implicit call to the parent’s constructor.

1. exitExit_con_def: always appends a return quadrupole at the end of a constructor’s code to ensure no issues when run by the virtual machine. It returns the address where the variable that represents the constructor is stored.

enterEnum Registers all the values read and stores them in the current enum collection.

enterEnumeration Creates a new enumeration class and checks for name conflicts.

enterSs_exp Empty.

1. enterExit_sexp: checks if there is a pending `and` or `or` operation and evaluates it if true; it then verifies the result type to ensure that the operation was a valid one. Finally, it increments the adequate counters in the calling method, creates the `operator` quadrupole, and stores the value and type in the correct stacks.

2. enterAnd_op: adds an `and` operator to the `pending operators` stack.
3. enterOr_op: adds an `or` operator to the `pending operators` stack.

enterS_exp Empty.

1. enterExit_exp: checks if there is a pending relational operator to be evaluated and does so if true. If validates that it was a valid operation through the semantic cube, it increments the adequate counters in the calling method, creates the `operator` quadrupole, and stores the value and type in the correct stacks.

enterExpression Empty.

1. enterExit_term: checks if there is a pending `+` (plus) or `-` (minus) operator to be evaluated and does so if true. If validates that it was a valid operation through the semantic cube, it increments the adequate counters in the calling method, creates the `operator` quadrupole, and stores the value and type in the correct stacks.
2. enterPlus_op: adds an `plus` operator to the `pending operators` stack.
3. enterMinus_op: adds an `minus` operator to the `pending operators` stack.

exitFactor Checks if there is a unitary operator to be evaluated and creates the appropriate quadrupoles to apply such operator to the factor.

1. enterOpen_paren: adds an `(` operator to the `pending operators` stack. It acts like a false bottom.
2. enterClose_paren: adds an `)` operator to the `pending operators` stack. It acts like a false bottom.

enterFunction_args Appends the names of the arguments read when defining a method in the `argument_names` list.

exitFunction_args Joins the argument names and their types and adds them to the corresponding method definition; this assignation could be to a class or a specification.

enterFunction_call Validates the existence of the call. To do that, it checks if it was defined locally, if not, it checks if it was defined in an ancestor; it also checks if it is defined in another class y it was a call made with composition.

exitFunction_call First it checks that the call was made with the correct number of arguments. Then it creates the `go subroutine` quadrupole and if it is not a `Nothing` method, immediately it creates a `retrieve` quadrupole to save the result in case the same method is called right after since it would cause an overwrite.

enterFunction_def Resets the local variable counters and initializes the appropriate class to store the relevant information, like its name or return type. It checks for name conflicts within the current context. Finally it checks if the new method being defined in the main method, if it is it updates the first quadrupole to point to it. It also is responsible of checking that there are never two main method declarations.

1. exitExit_func_def: always appends a return quadrupole at the end of a method's code to ensure no issues when run by the virtual machine. It returns the address where the variable that represents the method is stored.

enterOperand If there is a relational operator, it adds it to the `pending operators` stack.

enterField Stores all the names for the global variables or fields of the current class.

exitField Adds to the class registry the name of the global variables along with their types and assigned virtual addresses.

enterSpec_function Checks that the method in a specification is not duplicated and adds its instance to the registry.

enterSpecification Creates a new specification instance and registers it in the master tables. Before this it checks to validate no name conflicts with other specifications or classes.

enterAssignation_def Initializes helper variables and data structures and keeps track of the name of the new variable.

exitAssignation_def Creates, registers and stores the new declared local variables inside the current method's context.

exitStatute Checks that a return statement is present if the method has a return type that is different from `Nothing` and if that is the case, it makes sure that the return type matches the type of the expression being returned.

enterSuper_type Creates a variable instance.

enterSimple_type Keeps track of the current type being precessed.

enterComplex_type Keeps track of the current type being precessed.

enterTerm Empty.

1. enterExit_factor: checks if there is a pending `*` (times) or `/` (divides) operator to be evaluated and does so if true. If validates that it was a valid operation through the semantic cube, it increments the adequate counters in the calling method, creates the `operator` quadrupole, and stores the value and type in the correct stacks.
2. enterStar_op: adds an `times` operator to the `pending operators` stack.

3. enterDiv_op: adds an `divides` operator to the `pending operators` stack.

enterWhile_loop Empty.

1. exitAfter_while: sets the quadrupole index where the program counter (PC) must jump when the condition in a while statement evaluates to `false`.
2. exitEnd_while: creates the `go to` quadrupole to return to the start of the cycle.

enterVdim Registers the name of a multidimensional variable.

exitVdim Does all the processing to store and manipulate a multidimensional variable.

enterArray_def Keep track that an array is being processed.

exitArray_def Register in the class or method were it is defined, the multidimensional variable.

enterArray_var Check that a call to a multidimensional variable is correctly written and that it is within the specified limits.

1. exitClose_sbracket Neuralgic point for every dimension when a vector is called. Verifies that the pointed value is in the range. Checks that the index is integer.
2. enterClose_sbracket Updates pending dimensions stack. Increments the dimension of the current dimensioned variable.

exitArray_var Creates the appropriate quadrupole to process multidimensional variables (indirect addressing and literal addresses).

enterWrite_func adds an `Write` operator to the `pending operators` stack.

exitWrite_func Create a `Write` quadrupole.

enterRead_int_func adds an `ReadInt` operator to the `pending operators` stack.

exitRead_int_func Create a `ReadInt` quadrupole.

enterRead_real_func adds an `ReadReal` operator to the `pending operators` stack.

exitRead_real_func Create a `ReadReal` quadrupole.

enterRead_text_func adds an `ReadText` operator to the `pending operators` stack.

exitRead_text_func Create a `ReadText` quadrupole.

enterRead_bool_func adds an `ReadBoolean` operator to the `pending operators` stack.

exitRead_bool_func Create a `ReadBool` quadrupole.

enterWrite_line_func adds an `WriteLine` operator to the `pending operators` stack.

exitWrite_line_func Create a `WriteLine` quadrupole.

enterNot_op adds an unary `not` operator to the `pending operators` stack.

enterUnary_plus adds an unary `plus` operator to the `pending operators` stack.

enterUnary_minus adds an unary `minus` operator to the `pending operators` stack.

3.5 Semantic Analysis and Intermediate Code Generation

3.5.1 Operation Codes

The compiler works with two types of operation codes: `Operation` and `Operator`, which are represented internally as two `IntEnum` classes. These are used to parse and solve expressions correctly, as well as ensure that the language semantics are followed to the letter during compilation. Operators are shown in table 7 and operations on table 8.

MoM Language Operators with Codes			
PLUS (1)	READ_BOOL (31)	TIMES (3)	DIVIDES (4)
LESS_THAN (8)	WRITE_LINE (35)	EQUAL_EQUAL (12)	AND (5)
LESS_EQUAL (9)	GREATER_THAN (10)	GREATER_EQUAL (11)	NOT (7)
OPEN_PAREN (13)	CLOSE_PAREN (14)	EQUAL (15)	WRITE (27)
READ_INT (28)	READ_REAL (29)	READ_TEXT (30)	MINUS (2)
OPEN_SPAREN (32)	CLOSE_SPAREN (33)	VERIFY (34)	OR (6)

Table 7: Compiler operators with corresponding codes.

MoM Language Operations with Codes		
GO_TO_FALSE (16)	GO_TO_TRUE (17)	GO_TO (18)
RETURN (19)	ERA (20)	PARAM (21)
GO_SUB (22)	GO_CONSTRUCTOR (23)	ERA_CONSTRUCTOR (24)
END (25)	END_CLASS (26)	GO_MAIN (36)
RETRIEVE (37)		

Table 8: Compiler operations with corresponding codes.

3.5.2 Virtual Addresses

The language manages memory during compilation through the use of virtual addresses. These addresses serve two purposes. The first one is to classify them and avoid having collisions between different class and method contexts. The second one, is to identify which type a variable or constant is based on their address because the language's intermediate code is processed with *erasure*.

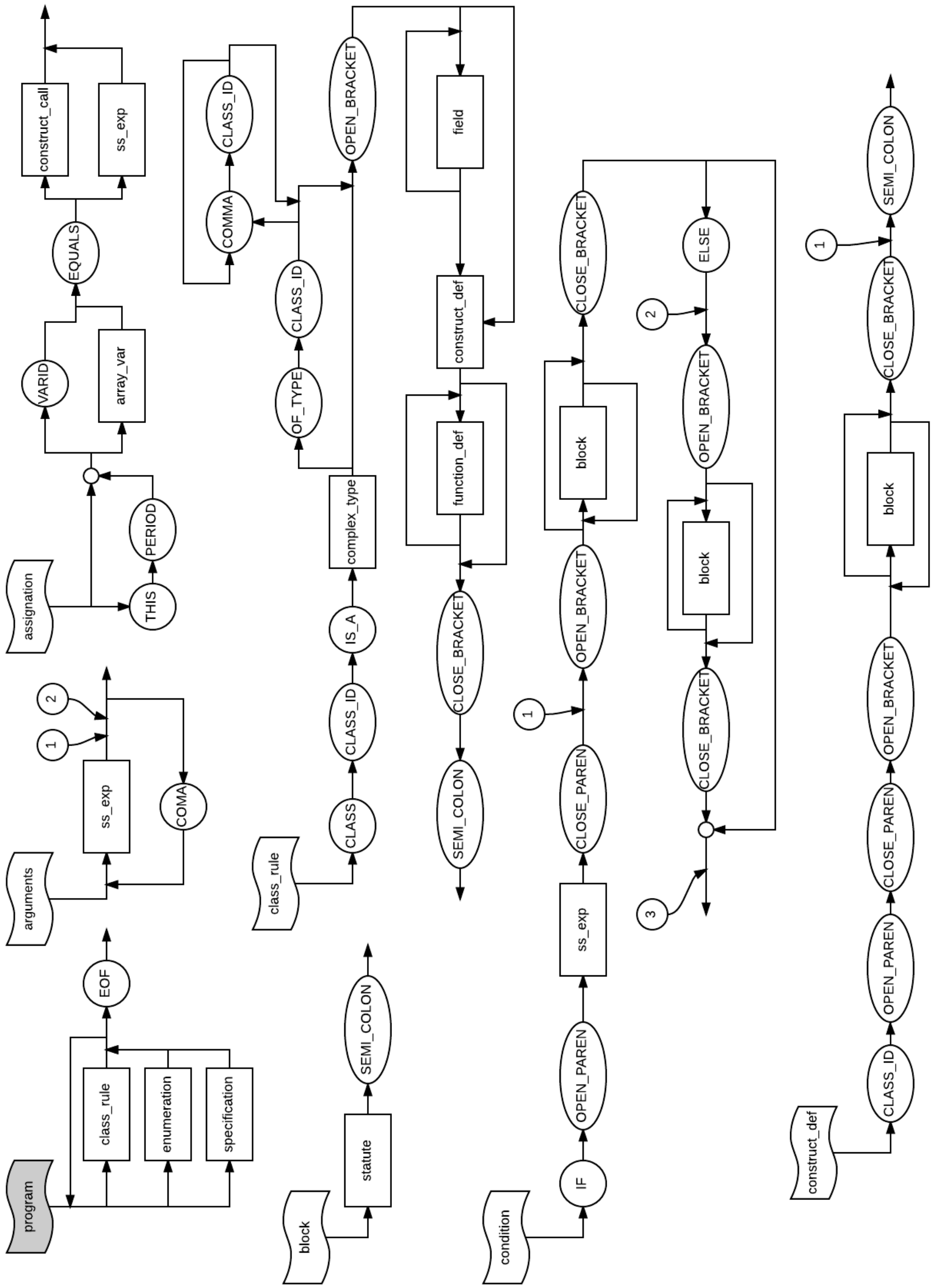

Figure 2: Syntax Diagrams (A).

Virtual memory is divided into four groups depending on the general type of element stored. One is for constants, the other is for temporal values, there is one for local values, and the last one is for global values. These are presented in tables 9, 10, 11, and 12. It is

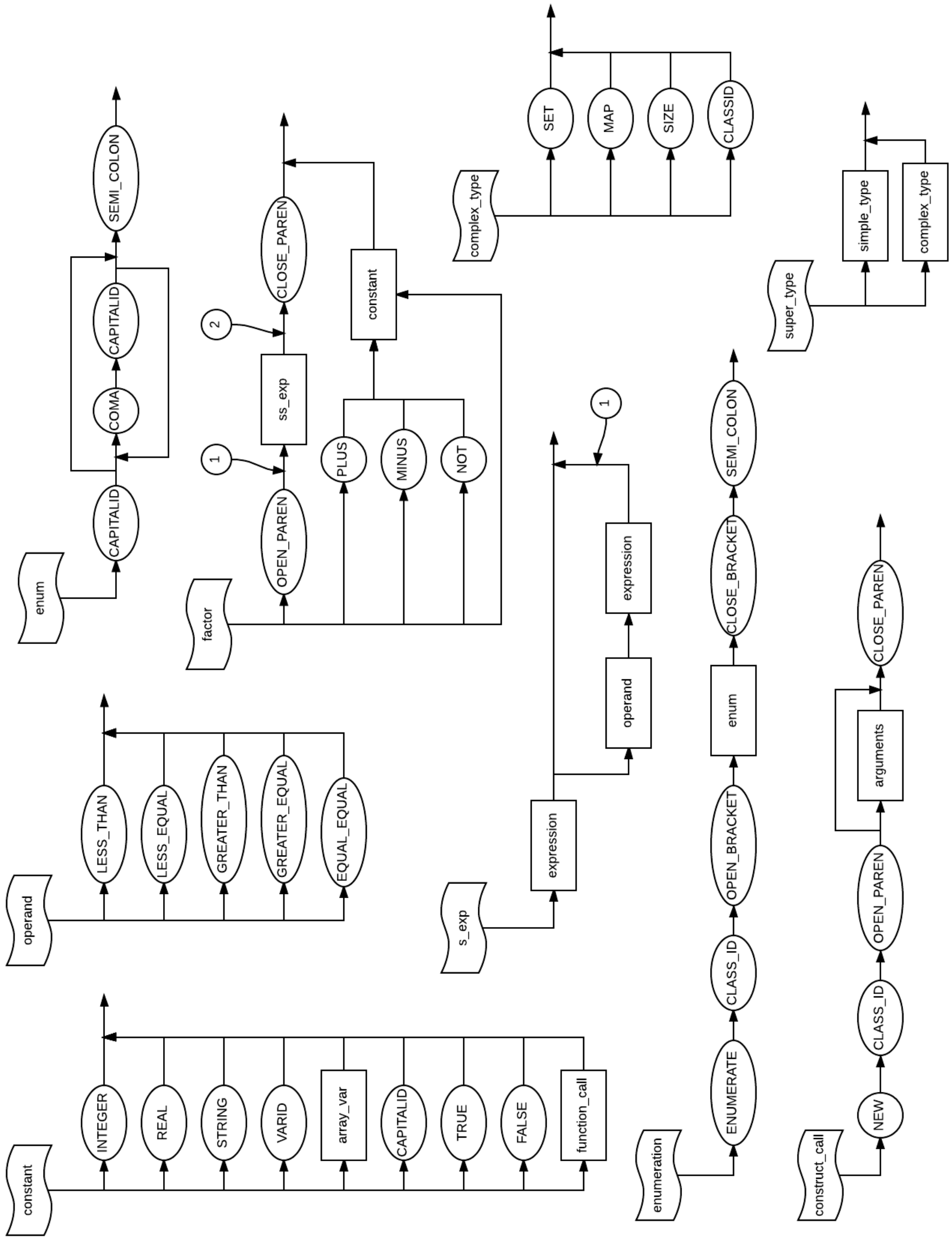

Figure 3: Syntax Diagrams (B).

worth noting that constants and temporal elements do not have memory assignation due to how the compiler is implemented.

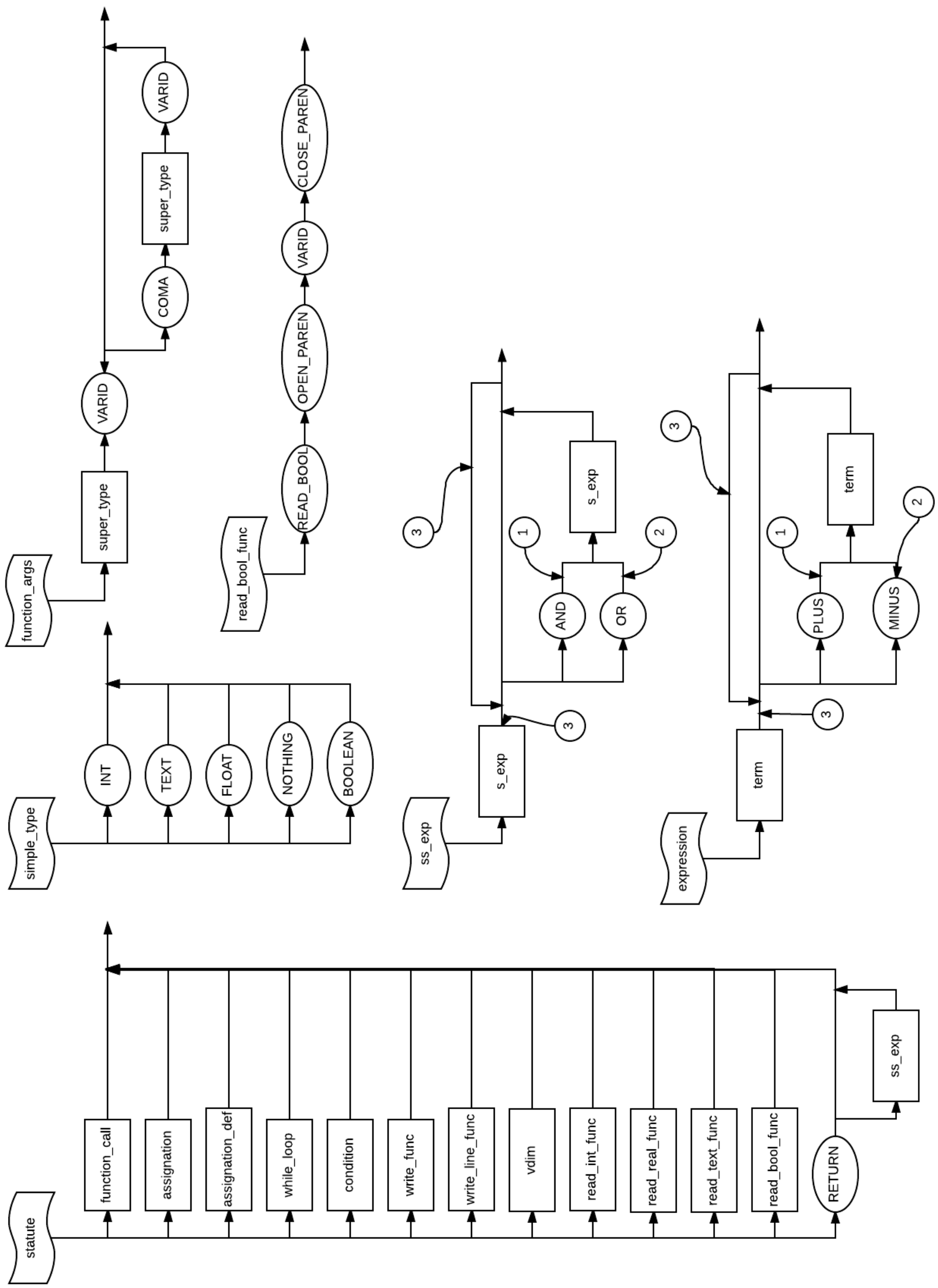

Figure 4: Syntax Diagrams (C).

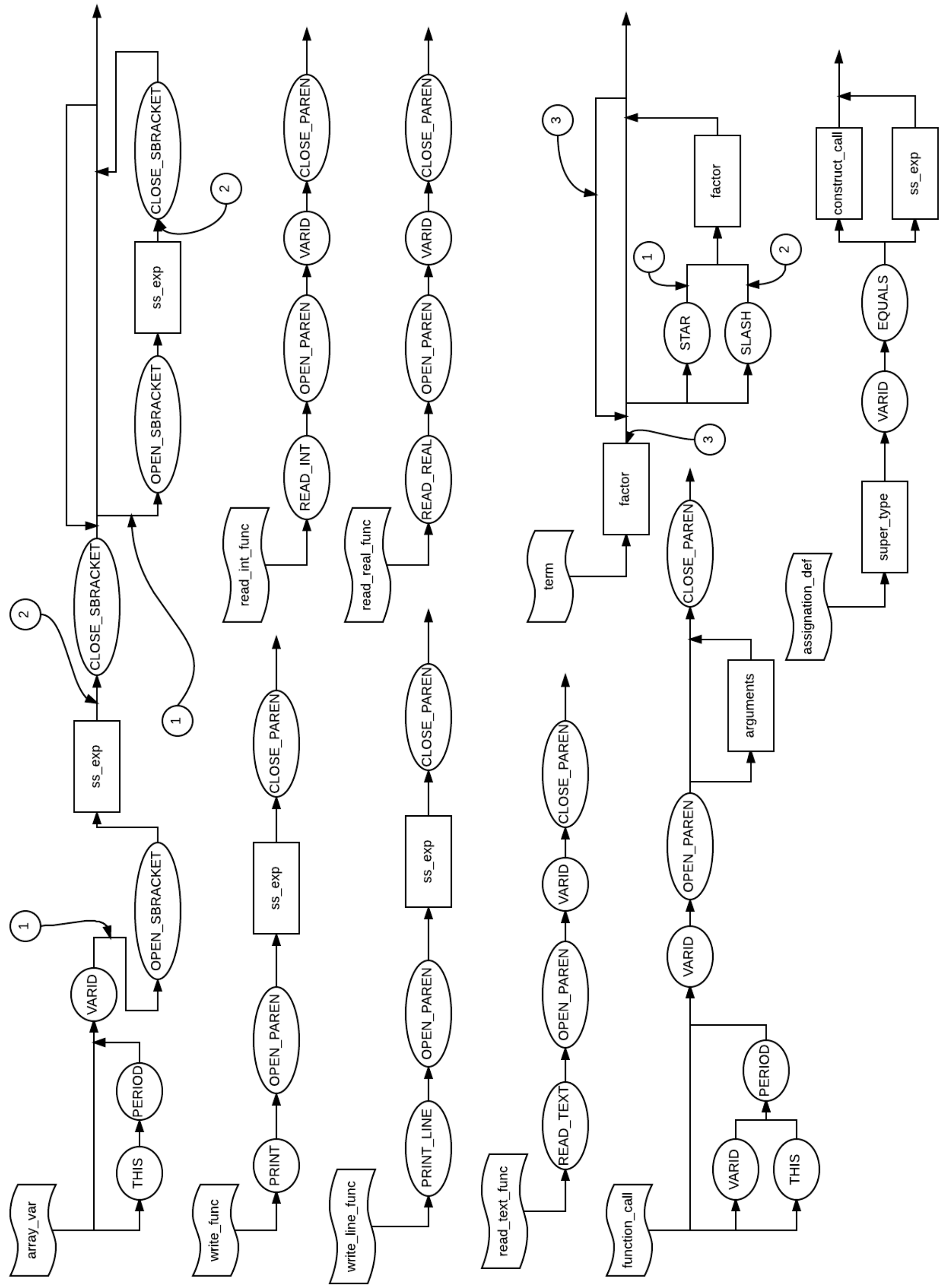

Figure 5: Syntax Diagrams (D).

3.5.3 Semantic Cube

In order to promote the correct evaluation of types when evaluating expressions, as well as to guarantee the correctness of the types used in the language. Operations on variables

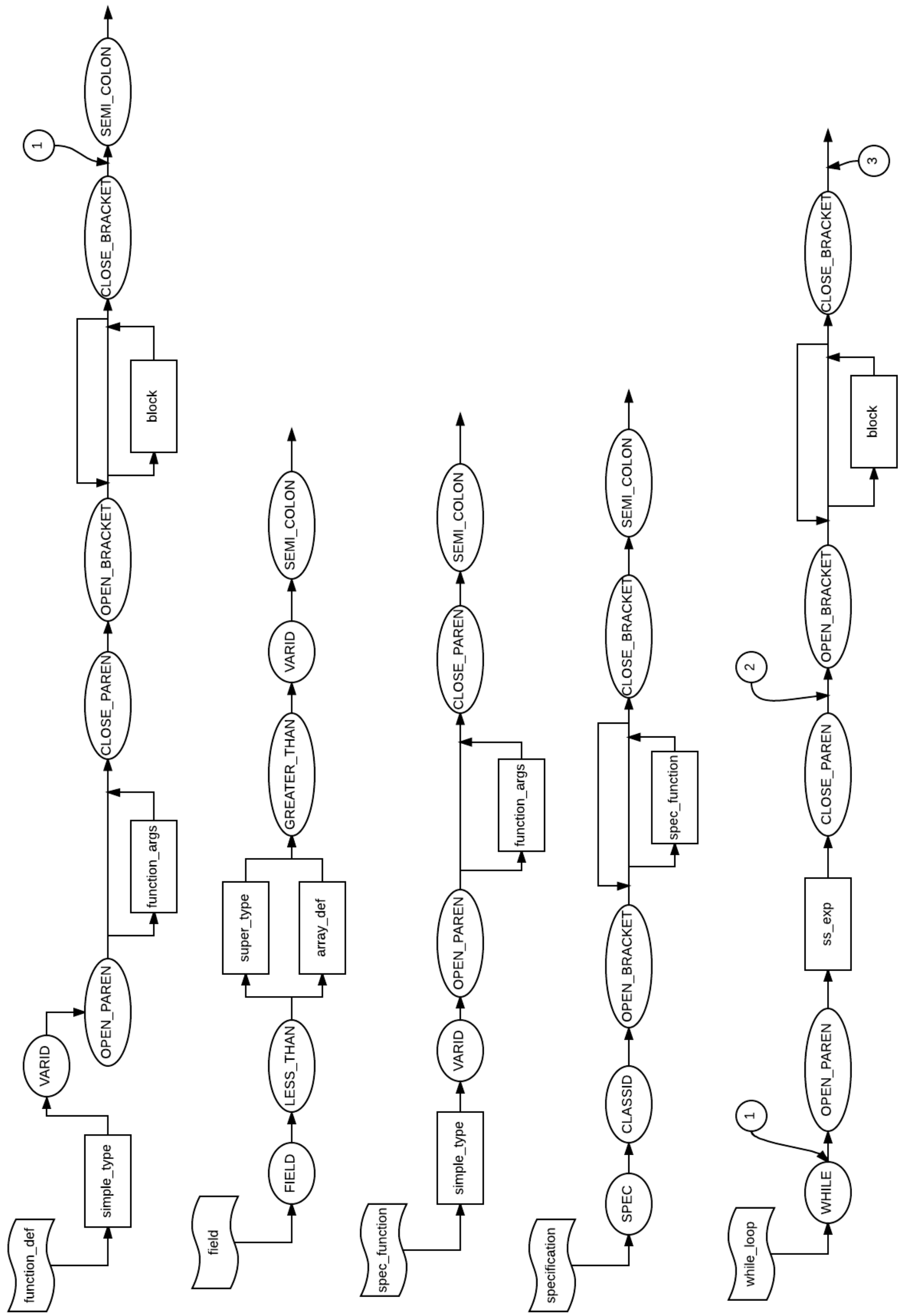

Figure 6: Syntax Diagrams (E).

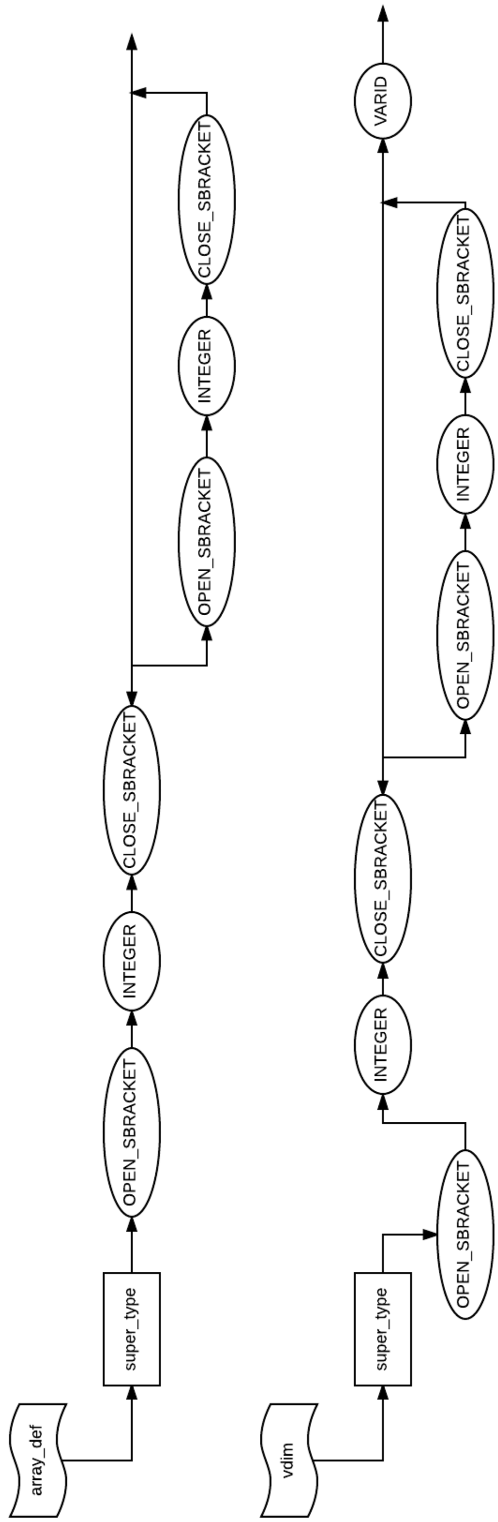

Figure 7: Syntax Diagrams (F).

Range			Type
10000	—	10999	Integer
11000	—	11999	Real
12000	—	12999	Boolean
13000	—	13999	Text
50000	—	59999	Object

Table 9: Global virtual memory sections by type and range.

Range			Type
14000	—	14999	Integer
15000	—	15999	Real
16000	—	16999	Boolean
17000	—	17999	Text
60000	—	69999	Object

Table 10: Local virtual memory sections by type and range.

Range			Type
18000	—	18999	Integer
19000	—	19999	Real
20000	—	20999	Boolean
21000	—	21999	Text

Table 11: Temporal virtual memory sections by type and range.

Range			Type
22000	—	22999	Integer
23000	—	23999	Real
24000	—	24999	Text
25000	—	25100	Boolean

Table 12: Constant virtual memory sections by type and range.

and constants are validated through the semantic cube, a three-dimensional matrix that returns the resulting type for a given operation applied to two operands. If the operation was invalid, `Type.OTHER` is returned as an error. The cube is shown in table 13.

If there is a query that is not listed on table 13, the result is automatically `Type.OTHER`.

3.6 Data Structures and Custom classes

To facilitate the design and implementation of the compiler, custom classes were used to increase the level of abstraction of the compilation process, as well as data structures to keep control over those class objects.

Semantic Cube							
Operand	Operand	Operation	Result	Operand	Operand	Operation	Result
TEXT	TEXT	+	TEXT	TEXT	INT	+	TEXT
TEXT	REAL	+	TEXT	TEXT	BOOL	+	TEXT
INT	INT	+	INT	INT	TEXT	+	TEXT
INT	INT	-	INT	INT	INT	*	INT
INT	INT	/	INT	INT	INT	<	BOOL
INT	INT	<=	BOOL	INT	INT	>	BOOL
INT	INT	>=	BOOL	INT	INT	==	BOOL
INT	REAL	+	REAL	INT	REAL	-	REAL
INT	REAL	*	REAL	INT	REAL	/	REAL
INT	REAL	<	BOOL	INT	REAL	<=	BOOL
INT	REAL	>	BOOL	INT	REAL	>=	BOOL
INT	REAL	==	BOOL	REAL	TEXT	+	TEXT
REAL	REAL	+	REAL	REAL	REAL	-	REAL
REAL	REAL	*	REAL	REAL	REAL	/	REAL
REAL	REAL	<	BOOL	REAL	REAL	<=	BOOL
REAL	REAL	>	BOOL	REAL	REAL	>=	BOOL
REAL	REAL	==	BOOL	REAL	INT	+	REAL
REAL	INT	-	REAL	REAL	INT	*	REAL
REAL	INT	/	REAL	REAL	INT	<	BOOL
REAL	INT	<=	BOOL	REAL	INT	>	BOOL
REAL	INT	>=	BOOL	REAL	INT	==	BOOL
BOOL	BOOL	&&	BOOL	BOOL	BOOL	——	BOOL
BOOL	BOOL	!	BOOL	BOOL	BOOL	==	BOOL

Table 13: Semantic Cube.

3.6.1 Global Tables

As mentioned in section, the user is capable of creating three types of complex structures: classes, enumerations, and specifications. These are stored in Python dictionaries in `MasterTables.py` and are declared as shown in listing 5. Notice that the all constants declared in a given `.mom` file are stores in these tables too.

```
enumerations = {}
classes = {}
specifications = {}
constants = collections.OrderedDict()
```

Listing 5: Master Tables

Similar to the above case, the Listener, which is in charge of executing custom Python code as the compiler reads the file, holds control data structures. These are shown in listing 6.

```
destination_type = []
```

```python
source_type = []
arguments = []
argument_names = []

pending_operands = list()
pending_types = list()
pending_operators = list()
pending_jumps = list()
pending_dims = list()
quads = list()
```

Listing 6: Control Data Structures

The `destination_type` variable is a stack which keeps control of the type of the expression or variable located on the left side of the assignation operator. On the other hand, `source_type` is the stack that controls the types of expressions on the right. The `arguments` and `argument_names` stacks are required because function arguments are read from a function definition in their entirety by the corresponding listener method.

The other stacks are similar to the ones seen in the compilers course. `pending_operands` holds the operands while evaluating an expression, `pending_types` the types of their respective operands, and `pending_operators` is the corresponding stack for the operators to be applied to a given expression. `pending_dims` keeps control of the dimensions of variables if they are structures like arrays. Finally, the `quads` list is used to store the quadrupoles as they are being generated by the compiler.

The rest of the subsections explore the custom classes created for the project.

3.6.2 Class

This class represents every user defined class found in a `.mom` program. Its responsibility is to encapsulate all related data regarding to a specific class definition. The information stored by instances of this class is shown below. For more details see listing 7

- The name of the class.
- The parent of the class.
- The list of specifications that the class implements.
- The method instances defined within the class and obtained through its ancestors.
- The global variables defined within the class and those obtained through its ancestors.

```python
from src.structures import Method


class Class:
    GLOBAL_INT_TOP = 10_000
    GLOBAL_REAL_TOP = 11_000
    GLOBAL_BOOLEAN_TOP = 12_000
    GLOBAL_TEXT_TOP = 13_000
    GLOBAL_OBJECT_TOP = 50_000

    GLOBAL_INT_BOTTOM = 10_999
    GLOBAL_REAL_BOTTOM = 11_999
    GLOBAL_BOOLEAN_BOTTOM = 12_999
```

```python
    GLOBAL_TEXT_BOTTOM = 13_999
    GLOBAL_OBJECT_BOTTOM = 59_999

    def __init__(self, class_name: str, class_parent: str, class_specifications: set):
        self._class_name = class_name
        self._class_parent = class_parent
        self._class_specifications = class_specifications
        self._methods = {}
        self._variables = {}
        self.cur_global_int = self.GLOBAL_INT_TOP
        self.cur_global_real = self.GLOBAL_REAL_TOP
        self.cur_global_boolean = self.GLOBAL_BOOLEAN_TOP
        self.cur_global_text = self.GLOBAL_TEXT_TOP
        self.cur_global_object = self.GLOBAL_OBJECT_TOP
        self.nothing_address = 99_999

    def add_method(self, method: Method):
        self._methods[method.name] = method

    def reset_address_counters(self):
        """The counters that keep track of the current virtual memory assignations are reset to the top
        value. This means that previous assigned memory is erased.

        :return: None.
        """
        self.cur_global_int = self.GLOBAL_INT_TOP
        self.cur_global_real = self.GLOBAL_REAL_TOP
        self.cur_global_boolean = self.GLOBAL_BOOLEAN_TOP
        self.cur_global_text = self.GLOBAL_TEXT_TOP

    def add_argument(self, arg_name: str, arg_type, is_array: bool, address: int, mem_size: int, dim=[], c_type=""):
        """Add argument to variable dictionary along with its type.

        :param dim: Stores dims for array
        :param c_type: if the argument is of type CLASS, this field holds the name of the class or specification.
        :param address: the virtual address for this argument, according to its type, in the VM.
        :param is_array: a boolean argument that specifies if the argument is an array.
        :param arg_type: the type of the argument, may be simple or complex (a.k.a a super type in the grammar).
        :param arg_name: the name of the argument, must be unique among the rest of the arguments and
            the local variables of the method.
        :param mem_size : the size in memory, by default 1
        :return: None.
        """
        if arg_name in self._variables:
            raise NameError("Variable with name `" + arg_name + "` in class already defined within scope.")

        self._variables[arg_name] = {'name': arg_name,
                                     'type': arg_type,
                                     'is_array': is_array,
                                     'address': address,
                                     'mem_size': mem_size,
                                     'dim': dim,
                                     'class_type': c_type}

    @property
    def name(self):
        return self._class_name

    @property
    def parent(self):
        return self._class_parent

    @property
    def specifications(self):
        return self._class_specifications

    @property
    def methods(self):
        return self._methods

    @property
    def variables(self):
        return self._variables

    @property
    def size(self):
        return len(self.variables)
```

Listing 7: Class.py

3.6.3 Enumeration

Enumerations, like C, are just a cleaner way of using constants within the code. As such, this class in in charge of keeping track of the elements that belong to a certain collection, and assigning them a unique value **within the collection**. The class is shown in listing 8.

The data that it keeps track of is shown below:

- The name of that collection, that is, the name by which the enumeration is identified.
- The name of the values stored within the enumeration. These values are assigned consecutive numbers as they are being added to the collection starting from one.

```python
class Enumeration:
    """Holds the metadata for a enumeration structure in the MoM Adventure language.

    Attributes:
        name: the name by which this enumeration can be identified. It must be unique in the program.
        values: a dictionary that maps the names of the different values to their respective values
            in this same enumeration.
    """

    def __init__(self, name: str):
        self._name = name
        self._values = {}
        self._counter = 1

    def add_value(self, value: str) -> None:
        """Adds a new value to this enumeration.

        The name of the value must be unique. There is no guarantee of which value it will hold.
        """
        self._values[value] = self._counter
        self._counter += 1

    @property
    def name(self):
        return self._name

    @property
    def values(self):
        return self._values

    __author__ = "Marco A. Peyrot (mpeyrotc)"
    __license__ = "MIT"
    __version__ = "1.0.0"
    __email__ = "macpeyrot@hotmail.com"
    __status__ = "Development"
```

Listing 8: Enumeration.py

3.6.4 Method

This class stores the relevant information for each of the defined methods declared by a given class or specification. Its code is shown in listing 9 and the information they store in the list below.

- The name of the method, to avoid overwriting methods (which is prohibited by the language).
- The return type of the method, which must be of primitive type by language specification.
- The variables expected as arguments to the method and their corresponding types.
- The quadrupole address where the code inside the method begins execution.
- The number of temporal values used by th method's call to enable the correct amount of storage assigned to it when invoked by the virtual machine.

```python
from src.structures.SemanticTable import Type


class Method:
    """Represents a method contained within a class.

    It holds a name, a return type, and if required the arguments it receives.
    """

    LOCAL_INT_TOP = 14_000
    LOCAL_REAL_TOP = 15_000
    LOCAL_BOOLEAN_TOP = 16_000
    LOCAL_TEXT_TOP = 17_000
    TEMP_INT_TOP = 18_000
    TEMP_REAL_TOP = 19_000
    TEMP_BOOLEAN_TOP = 20_000
    TEMP_TEXT_TOP = 21_000

    LOCAL_OBJECT_TOP = 60_000
    LOCAL_OBJECT_BOTTOM = 69_999

    LOCAL_INT_BOTTOM = 14_999
    LOCAL_REAL_BOTTOM = 15_999
    LOCAL_BOOLEAN_BOTTOM = 16_999
    LOCAL_TEXT_BOTTOM = 17_999
    TEMP_INT_BOTTOM = 18_999
    TEMP_REAL_BOTTOM = 19_999
    TEMP_BOOLEAN_BOTTOM = 20_999
    TEMP_TEXT_BOTTOM = 21_999

    def __init__(self, name: str, return_type: Type) -> None:
        """The de facto constructor for a method definition.

        :param return_type: the return type for this specific method. It can be any value from the Type enum
            except for Other.
        :param name: a str value that represents the name of this method.
        """
        self._name = name
        self._return_type = return_type
        self._variables = {}
        self._argument_types = []
        self._start_quad = -1
        self.temp_count = 0

        self.cur_local_int = self.LOCAL_INT_TOP
        self.cur_local_real = self.LOCAL_REAL_TOP
        self.cur_local_boolean = self.LOCAL_BOOLEAN_TOP
        self.cur_local_text = self.LOCAL_TEXT_TOP
        self.cur_local_object = self.LOCAL_OBJECT_TOP
        self.cur_temp_int = self.TEMP_INT_TOP
        self.cur_temp_real = self.TEMP_REAL_TOP
        self.cur_temp_boolean = self.TEMP_BOOLEAN_TOP
        self.cur_temp_text = self.TEMP_TEXT_TOP

    def add_argument(self, arg_name: str, arg_type, is_array: bool, address: int, mem_size: int, dim=[], p_type="") ->
     None:
        """Add argument to variable dictionary along with its type.

        :param dim: Structure representing dimension
        :param p_type: if the argument is of type CLASS, this field holds the name of the class or specification.
        :param address: the virtual address for this argument, according to its type, in the VM.
        :param is_array: a boolean argument that specifies if the argument is an array.
        :param arg_type: the type of the argument, may be simple or complex (a.k.a a super type in the grammar).
        :param arg_name: the name of the argument, must be unique among the rest of the arguments and
            the local variables of the method.
        :param mem_size: size in memory, default 1
        :return: None.
        """
        if arg_name in self._variables:
            raise NameError("Method " + arg_name + " already defined within scope.")

        self._variables[arg_name] = {'type': arg_type,
                                     'is_array': is_array,
                                     'address': address,
                                     'mem_size': mem_size,
                                     'dim': dim,
                                     'p_type': p_type}

    def add_argument_type(self, arg_type, is_array: bool):
        self._argument_types.append({"arg_type": arg_type, "is_array": is_array})

    def reset_address_counters(self):
        self.cur_local_int = self.LOCAL_INT_TOP
        self.cur_local_real = self.LOCAL_REAL_TOP
        self.cur_local_boolean = self.LOCAL_BOOLEAN_TOP
        self.cur_local_text = self.LOCAL_TEXT_TOP
        self.cur_temp_int = self.TEMP_INT_TOP
        self.cur_temp_real = self.TEMP_REAL_TOP
        self.cur_temp_boolean = self.TEMP_BOOLEAN_TOP
        self.cur_temp_text = self.TEMP_TEXT_TOP

    @property
```

```python
    def name(self):
        return self._name

    @property
    def return_type(self):
        return self._return_type

    @property
    def variables(self):
        return self._variables

    @property
    def argument_types(self):
        return self._argument_types

    @property
    def num_of_params(self):
        return len(self._argument_types)

    @property
    def num_local_vars(self):
        return len(self._variables) - len(self._argument_types)

    @property
    def start(self):
        return self._start_quad

    @start.setter
    def start(self, start):
        self._start_quad = start

    @property
    def size(self):
        return self.temp_count + self.num_local_vars + self.num_of_params


__author__ = "Marco A. Peyrot (mpeyrotc)"
__license__ = "MIT"
__version__ = "1.0.0"
__email__ = "macpeyrot@hotmail.com"
__status__ = "Development"
```

Listing 9: Method.py

3.6.5 Quadrupole

The quadrupole is a kind of tuple that holds four values. On the leftmost space an operator or operation code is specified, then the left argument and right arguments are specified, finally on the rightmost part, the destination is declared. The code is fairly simple and is shown in listing 10.

```python
class Quadrupole:
    def __init__(self, operator, left_operand, right_operand, result):
        self._operator = operator
        self._left_operand = left_operand
        self._right_operand = right_operand
        self._result = result

    @property
    def operator(self):
        return self._operator

    @property
    def left_operand(self):
        return self._left_operand

    @property
    def right_operand(self):
        return self._right_operand

    @property
    def result(self):
        return self._result

    @result.setter
    def result(self, value: int):
        self._result = value
```

Listing 10: Quadrupole.py

3.6.6 Specification

Finally, the specification is a class that is in charge of two things. First, keeping track of its name to avoid name conflicts; and second, the methods which it defines. Its code is in listing 11.

```python
from src.structures import Method


class Specification:
    """This class contains the metadata for the specification structure of the
    MoM language.

    It holds the method signatures that will be acquired by the classes
    that implement a specific implementation.
    """

    def __init__(self, name: str):
        self._name = name
        self._methods = {}

    def add_method(self, method: Method) -> None:
        """Adds a method to this specification's group of registered
        methods.

        :param method: the method object to be added
        :return: None.
        """
        self._methods[method.name] = method

    @property
    def name(self):
        return self._name

    @property
    def methods(self):
        return self._methods
```

Listing 11: Specification.py

4 The Virtual Machine

The Virtual machine was developed on the same machines and programming language. Therefore, no additional specifications regarding this subject are required.

4.1 Execution Memory Administration

Since the MoM Adventure Language is Object Oriented, its memory mappings are a bit complex. This section covers all the relevant information so that the user can understand its inner workings. All the relevant data structures and their relationships are shown in figure 8.

The virtual machine environment has three independent global structures that can be accessed by any of the other objects. One is an array of arrays of constants, from index 0 to 4, that store integers, real numbers, strings, and booleans in that order. The other is a parameter `list` called params. This list is filled with the arguments that a method expects when it is called by th virtual machine, and emptied afterwards the method has grabbed its arguments from it. The last one is another `list` but this one is called `new_class` and holds the constructors as they are being call, in case that they are interrupted by another internal constructor call.

There are two other structures worth mentioning, actually, they are the ones that run the whole execution of a given program. The first one is a dictionary of classes and is called `Classes`. It holds, as shown in figure 8, all the details for all classes present in the program. Classes are identified by their name and hold a `list` for the global variables declared for that class. The second one is the *class execution stack* which holds the current class being executed (an internal method is running) and stores the ones that are asleep.

As a result of this architecture, there are two types of classes present in the system at any given time: the specification and the instance. The specification is a static element of the environment, that is, it does not change and only serves to instantiate class instances. These instances are the ones that hold the real values being used by the program (more on this later). These elements hold another type of dictionary, this one is for the method specifications. Its sole purpose is to identify the methods present in a given class and to keep track of how many local and temporal elements it has. It is indexed from 0 to 8, with the local variables coming first in this order: integers, real numbers, strings, booleans, and objects. Later come the temporal values, which go in the order of integers to real numbers, strings, and ending with booleans.

Finally, as mentioned previously, the other major structure is the class instance stack. It starts with an instance of the class where the main method was located. This structure is the heart of the execution process. Each element holds its own method stack, the dormant classes that were declared as fields of the class, and its global variables. The method stack similarly, holds its own variables and a reference to other class instances within its scope.

It is worth noting that there is no structure for enumerations because these are translated to integer values during compilation. Interfaces are transformed to regular classes too because their main purpose is to guarantee that the specification signature is followed, nothing more.

4.2 Address Translation

Virtual address translation to real addresses is accomplished with the help of four methods. These are described below:

1. `is_x(address: str)`, where x is the type of variable, that is local, temporal, constant or global. This function returns `True` if the argument is of the memory section specified by x. See listing 12 for an example.

```
def is_global(address: int) -> bool:
    return GLOBAL_INT_TOP <= address <= GLOBAL_TEXT_BOTTOM or
   GLOBAL_OBJECT_TOP <= address <= GLOBAL_OBJECT_BOTTOM
```

Listing 12: is_global function

2. `get_x(ci, address: str)`, where x is the type of variable, that is local, temporal, constant, or global. This function returns the real variable stored in a memory space of the control structures mentioned in the last section. However, this returned value is just in `string` format. See listing 13 for an example.

```
def get_global(ci, address: int):
    if GLOBAL_INT_TOP <= address <= GLOBAL_INT_BOTTOM:
        return ci.memory[0][address - GLOBAL_INT_TOP]
    if GLOBAL_REAL_TOP <= address <= GLOBAL_REAL_BOTTOM:
```

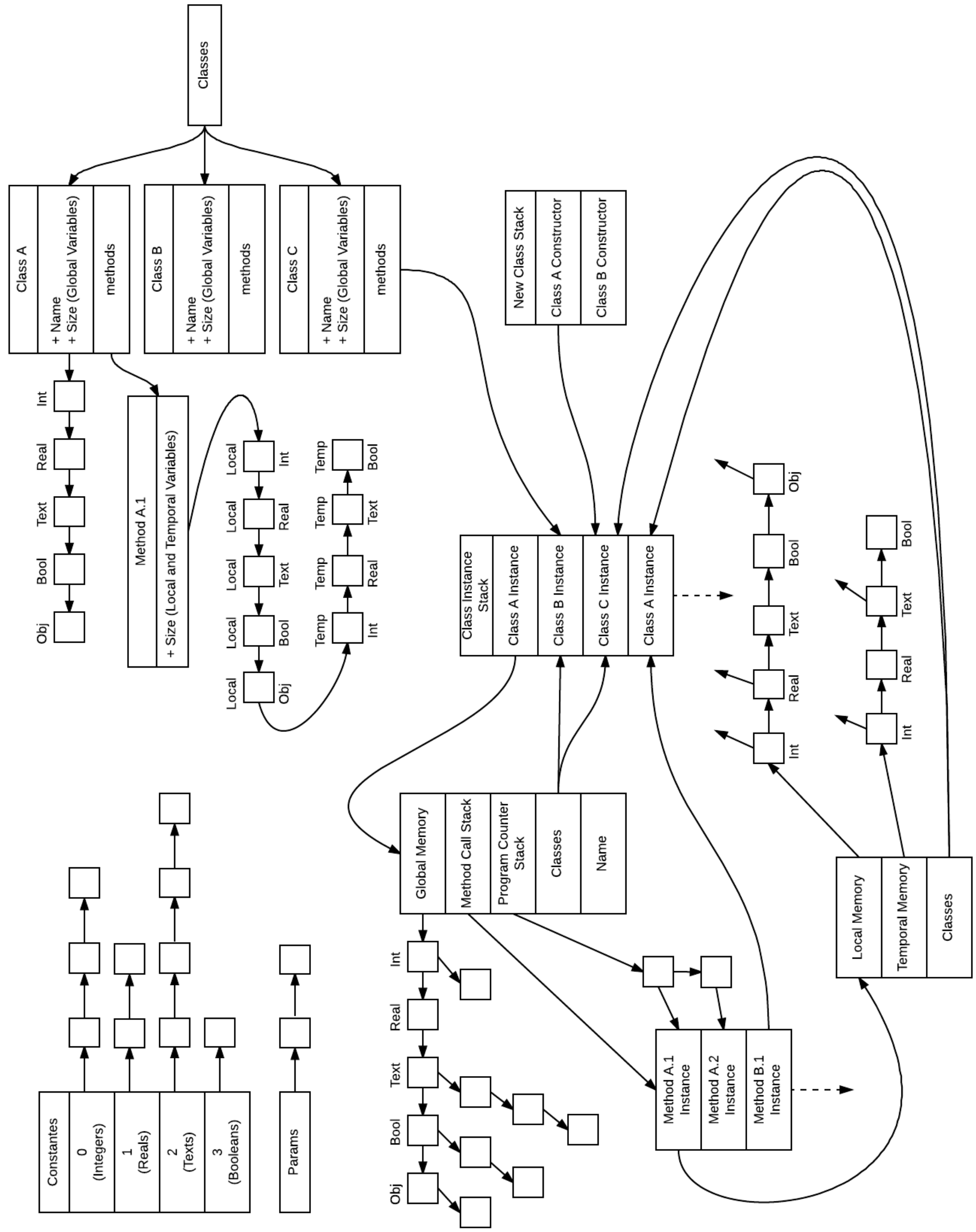

Figure 8: Virtual Memory Data Structures.

```
        return ci.memory[1][address - GLOBAL_REAL_TOP]
    if GLOBAL_TEXT_TOP <= address <= GLOBAL_TEXT_BOTTOM:
        return ci.memory[2][address - GLOBAL_TEXT_TOP]
    if GLOBAL_BOOLEAN_TOP <= address <= GLOBAL_BOOLEAN_BOTTOM:
        return ci.memory[3][address - GLOBAL_BOOLEAN_TOP]
    else:
```

```
        return ci.memory[4][address - GLOBAL_OBJECT_TOP]
```

Listing 13: get_global function

3. `set_x(ci, address: str, value)`, where x is the type of variable, that is local, temporal, constant, or global. This function stores in the correct real memory slot the value passed to it. See listing 14 for an example.

```
def get_global(ci, address: int):
    if GLOBAL_INT_TOP <= address <= GLOBAL_INT_BOTTOM:
        return ci.memory[0][address - GLOBAL_INT_TOP]
    if GLOBAL_REAL_TOP <= address <= GLOBAL_REAL_BOTTOM:
        return ci.memory[1][address - GLOBAL_REAL_TOP]
    if GLOBAL_TEXT_TOP <= address <= GLOBAL_TEXT_BOTTOM:
        return ci.memory[2][address - GLOBAL_TEXT_TOP]
    if GLOBAL_BOOLEAN_TOP <= address <= GLOBAL_BOOLEAN_BOTTOM:
        return ci.memory[3][address - GLOBAL_BOOLEAN_TOP]
    else:
        return ci.memory[4][address - GLOBAL_OBJECT_TOP]
```

Listing 14: set_global function

4. `get_raw_value(left_arg: str, right_arg: str)`, is a function that casts the value represented as a `string` to its appropriate type. See listing 15 for an example.

```
def get_raw_value(left_arg: str, right_arg: str):
    """Returns the appropriate type of variable according to its address.

    :param left_arg: left element of the quadrupole (the one to the right of the operator/operand)
    :param right_arg: righ element of the quadrupole (the one on the left of the destination value)
    :return: a tuple with the left and right values with their correct type (for Python) or None if not present.
    """
    left_arg = str(left_arg)
    right_arg = str(right_arg)
    left_value = None
    right_value = None
    left_is_absolute = left_arg.find("$")
    right_is_absolute = right_arg.find("$")

    if not left_arg == "":
        if not left_is_absolute == -1:
            left_value = left_arg[left_is_absolute + 1:].strip()
        elif is_constant(int(left_arg)):
            left_value = get_constant(int(left_arg))
        elif is_global(int(left_arg)):
            left_value = get_global(class_stack[-1], int(left_arg))
        elif is_local(int(left_arg)):
            left_value = get_local(class_stack[-1].method_stack[-1], int(left_arg))
```

```python
        elif is_temporal(int(left_arg)):
            left_value = get_temporal(class_stack[-1].method_stack[-1], int(left_arg))
        else:
            raise IndexError("No memory location defined.")

    if not right_arg == "":
        if not right_is_absolute == -1:
            right_value = right_arg[right_is_absolute + 1:].strip()
        elif is_constant(int(right_arg)):
            right_value = get_constant(int(right_arg))
        elif is_global(int(right_arg)):
            right_value = get_global(class_stack[-1], int(right_arg))
        elif is_local(int(right_arg)):
            right_value = get_local(class_stack[-1].method_stack[-1], int(right_arg))
        elif is_temporal(int(right_arg)):
            right_value = get_temporal(class_stack[-1].method_stack[-1], int(right_arg))
        else:
            raise IndexError("No memory location defined.")

    return left_value, right_value
```

Listing 15: get_raw_value function

As seen in the previous listings, address translation is done by applying the following function to the virtual addresses:

$$RealAddress = Memory[Section][VirtualAddress - TopAddressOfSection] \quad (1)$$

5 Language Functionality Tests

This section presents the execution results of the tests described in section 1.5.

1. Array_find:

```
class Math is_a Component {
    field<Int[30]> arrA;

    Math() {

    };

    Boolean findArrA(Int x){
        Int i = 0;
        while(i < 30){
            if(arrA[i] == x){
                return TRUE;
            };
            i = i + 1;
        };
        return FALSE;
    };

    Nothing main(){
        Int i = 0;
        Int j = 0;
        Int counter = 1;
```

```
        WriteLine("Array: ");
        while(i < 30){
            arrA[i] = counter;
            Write(arrA[i]);
            Write(" ");
            i = i + 1;
            counter = counter + 1;
        };
        WriteLine(" ");

        Int x = 6;
        if(findArrA(x)){
            Write(x);
            WriteLine(" found on array");
        }
        else{
            Write(x);
            WriteLine(" not found on array");
        };

        x = 200;
        if(findArrA(x)){
            Write(x);
            WriteLine(" found on array");
        }
        else{
            Write(x);
            WriteLine(" not found on array");
        };
    };
};
```

Listing 16: array_find.mom

```
$ run test.mom
Array:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
        26 27 28 29 30
6 found on array
200 not found in array
```

2. Array_sort:

```
class Math is_a Component {
    field<Int[10]> arrS;

    Math() {

    };

    Int partition(Int lo, Int hi){
        Int pivot = arrS[hi];
        Int i = (lo - 1);
        Int j = lo;
        Int temp = 0;
        while(j < hi){
            if(arrS[j] <= pivot){
                i = i + 1;
                temp = arrS[i];
                arrS[i] = arrS[j];
                arrS[j] = temp;
            };
            j = j + 1;
        };
        temp = arrS[i + 1];
        arrS[i + 1] = arrS[hi];
        arrS[hi] = temp;
        return i + 1;
    };

    Nothing sortArrS(Int lo, Int hi){
        if(lo < hi){
            Int pi = partition(lo, hi);
            sortArrS(lo, pi - 1);
            sortArrS(pi + 1, hi);
        };
    };
```

```

    Nothing main(){
        arrS[0] = 34;
        arrS[1] = 0;
        arrS[2] = 10;
        arrS[3] = 4;
        arrS[4] = 12;
        arrS[5] = 0;
        arrS[6] = 99;
        arrS[7] = 48;
        arrS[8] = 1;
        arrS[9] = 28;

        Int i = 0;
        WriteLine("Unsorted array:");
        while(i < 10){
            Write(arrS[i]);
            Write(" ");
            i = i + 1;
        };
        WriteLine(" ");
        sortArrS(0, 9);
        WriteLine("Sorted array: ");
        i = 0;
        while(i < 10){
            Write(arrS[i]);
            Write(" ");
            i = i + 1;
        };
        WriteLine(" ");
    };
};
```

Listing 17: array_qsort.mom

```
$ run test.mom
Unsorted Array:
34 0 10 4 12 0 99 48 1 28
Sorted Array:
0 0 1 4 10 12 28 34 48 99
```

3. Complex_class:

```
specification Player2 {
    Int a();
};

class Face2 is_a Component {
    field<Card> card;
    field<Int> aaaa;

    Face2() {
        card = new Card();
    };

    Nothing print() {
        WriteLine("Hey");
    };
};

class Character is_a Component of_type Player2 {
    field<Card> card;
    field<Int> ints;
    field<Int> physicalLife;
    field<Int[30]> arrNums;
    field<Int> numberOfItems;


    Character() {
    };

    Int a() {
    };

    Nothing test(){

    };

```

```
    Int setCommonCard(Card card) {

        if(TRUE || FALSE) {
            physicalLife = 100;
        };

        return 3;
    };

    Boolean addCommonCard(Real cards, Int ints) {
        if(numberOfItems <  -1) {
            physicalLife = physicalLife;
        } else {
            if(numberOfItems <  -1) {
                physicalLife = physicalLife;
            } else {

            };
        };
    };

    Nothing doAction(Face2 x) {
        x.print();
        WriteLine("que onda");
    };

    Text name() {
        while(TRUE && FALSE) {
        };

        return "Father Mateo" + "!!!";
    };

    Nothing main() {
        Face2 f = new Face2();
        Face2 object = new Face2();

        f.print();
        doAction(object);
    };
};
```

Listing 18: complex_class.mom

```
$ run test.mom
Hey
Hey
que onda
```

4. Conditions:

```
class Condition is_a Component {
    field<Int> a;
    field<Int> b;
    field<Int> c;
    field<Int> d;

    Condition() {
        if(a + b > d) {
            if(a < b) {
                a = 0;
                b = b + d;
            } else {
                c = a + b;
            };
        } else {
            a = b + c;
        };

        d = b + a * c;
    };

    Nothing test1() {
        if(1 < 5) {
            WriteLine("1 is less than 5");
        };
    };

    Nothing test2() {
```

```
        if(1 + 5 > 2) {
            WriteLine("6 is greater than 2");
        } else {
            WriteLine("6 is not greater than 2");
        };
    };

    Nothing main() {
        test1();
        test2();
    };

    Nothing test3() {
        while(a < b) {
            c = d + c;
        };
    };
};
```

Listing 19: conditions.mom

```
$ run test.mom
1 is less than 5
6 is greater than 2
```

5. Constructors:

```
class ComplexNumber is_a Component {
    ComplexNumber() {
    };

    Int real() {
        WriteLine("Hey Im a complex number");
        return 1;
    };

    Int imaginary() {
        return 2;
    };
};

class Expressions is_a Component {
    field<Int> integer;

    Expressions() {
        this.integer = 2;
    };

    Nothing exp1() {
        Int b = 2;
        Int c = 3;
        Int a = b + c;
        ComplexNumber num = new ComplexNumber();
        Expressions e = new Expressions();
        num.real();
    };

    Nothing main() {
        exp1();
    };
};
```

Listing 20: constructors.mom

```
$ run test.mom
Hey Im a complex number
```

6. Enumeration:

```
enumerate Days {
    MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY;
};

class Class is_a Component {
    Class() {

    };

    Nothing main() {
        WriteLine("MONDAY + FRIDAY = ");
        WriteLine(MONDAY + FRIDAY);
    };
};
```

Listing 21: enumeration.mom

```
$ run test.mom
MONDAY + FRIDAY =
6
```

7. Expressions:

```
class Expressions is_a Component {
    field<Int[2]> arr;

    Expressions() {
    };

    Nothing exp1() {
        Int b = 2;
        Int c = 3;
        Int a = b + c;
        WriteLine("b + c = ");
        WriteLine(a);
    };

    Nothing exp2() {
        Real a = 3.14;
        Int b = 3;
        Real result = a - b;
        WriteLine("a - b = ");
        WriteLine(result);
    };

    Nothing exp3() {
        Real a = 3.14;
        Int b = 2;
        Real result = a * b;
        WriteLine("a * b = ");
        WriteLine(result);
    };

    Nothing main() {
        exp1();
        exp2();
        exp3();
    };
};
```

Listing 22: expressions.mom

```
$ run test.mom
b + c =
5
a - b =
0.14000000000000012
a * b =
6.28
```

8. Factorial:

```
class Math is_a Component {
    Math() {

    };

    Int fact1(Int limit){
        if(limit < 0){
            return -1;
        };

        if(limit <= 1){
            return 1;
        };

        Int counter = 2;
        Int ans = 1;
        while(counter <= limit){
            ans = ans * counter;
            counter = counter + 1;
        };
        return ans;
    };

    Int fact2(Int limit) {
        if(limit <0){
            return -1;
        };

        if(limit <= 1){
            return 1;
        };

        return limit * fact2(limit - 1);
    };

    Nothing main() {
        WriteLine("Iterative factorial of 7");
        WriteLine(fact1(7));
        WriteLine("Recursive factorial of 7");
        WriteLine(fact2(7));
    };
};
```

Listing 23: factorial.mom

```
$ run test.mom
Iterative factorial of 7
5040
Recursive factorial of 7
5040
```

9. Fibonacci:

```
class Math is_a Component {
    Math() {
```

```
    };

    Int fib1(Int limit){
        if(limit < 0){
            return -1;
        };

        if(limit == 0){
            return 0;
        }
        else{
            if(limit <= 2){
                return 1;
            };
        };

        Int counter = 2;
        Int left = 1;
        Int right = 1;
        Int temp = 0;

        while(counter < limit){
            temp = right;
            right = left + right;
            left = temp;
            counter = counter + 1;
        };
        return right;
    };

    Int fib2(Int limit) {
        if(limit < 0){
            return -1;
        };
        Int temp = 0;
        Int temp2 = 0;

        if(limit == 0) {
            return 0;
        } else {
          if(limit == 1) {
            return 1;
          } else {
            temp = fib2(limit - 1);
            temp2 = fib2(limit - 2);
            return temp + temp2;
          };
        };
    };

    Nothing main() {
        WriteLine("Iterative 7th fibonacci number");
        WriteLine(fib1(7));
        WriteLine("Recursive 7th fibonacci number");
        WriteLine(fib2(7));
    };
};
```

Listing 24: fibonacci.mom

```
$ run test.mom
Iterative 7th fibonacci number
13
Recursive 7th fibonacci number
13
```

10. Functions:

```
class Expressions is_a Component {
    field<Int[2]> arr;
    field<Int> num;

    Expressions() {
    };

    Int func1() {
        Int b = 2;
```

```
        Real c = 3.0;
        Real a = b + c;
        return 0;
    };

    Nothing func2(Int a, Int b, Int c) {
        WriteLine(a + b + c);
    };

    Nothing func4() {
        WriteLine("This is function 4.");
    };

    Int func3(Real a) {
        return 0;
    };

    Text func5() {
        this.func4();
    };

    Nothing main() {
        func4();
        func2(100, 2, 1218);
        func2(1, 1, 1);
    };
};
```

Listing 25: functions.mom

```
$ run test.mom
This is function 4.
1320
3
```

11. Implicit_constructors:

```
class Character is_a Component {
    Character() {
        WriteLine("Implicit constructor of Character");
        Int a = 2 * 3;
    };
};

class Bob is_a Player {
    Bob() {
        WriteLine("Implicit constructor of Bob");
        Int a = 2 - 3;
    };
};

class Persona is_a Character {
    Persona() {
        Int a = 2 / 3;
    };

    Nothing main() {
        Bob b = new Bob();
        Character c = new Character();
    };
};
```

Listing 26: implicit_constructors.mom

```
$ run test.mom
Implicit constructor of Character
```

12. Matrix_multiplication:

```
class Math is_a Component {

    Math() {

    };

    Nothing main(){
        Int i = 0;
        Int j = 0;
        Int n = 3;
        Int counter = 1;
        Int[3][3] arrB;
        Int[3][3] arrC;
        Int[3][3] ans;

        while(i < n){
            j = 0;
            while(j < n){
                arrB[i][j] = counter;
                Write(arrB[i][j]);
                Write(" ");
                j = j + 1;
                counter = counter + 1;
            };
            WriteLine(" ");
            i = i + 1;
        };

        WriteLine(" * ");
        counter = 1;
        i = 0;
        while(i < n){
            j = 0;
            while(j < n){
                arrC[i][j] = counter;
                ans[i][j] = 0;
                Write(arrC[i][j]);
                Write(" ");
                j = j + 1;
                counter = counter + 1;
            };
            WriteLine(" ");
            i = i + 1;
        };

        WriteLine(" = ");
        i = 0; j = 0;
        Int k = 0;
        while(i < n){
            j = 0;
            while(j < n){
                k = 0;
                while(k < n){
                    ans[i][j] = ans[i][j] + arrB[i][k] * arrC[k][j];
                    k = k + 1;
                };
                j = j + 1;
            };
            i = i + 1;
        };

        i = 0;
        while(i < n){
            j = 0;
            while(j < n){
                Write(ans[i][j]);
                Write(" ");
                j = j + 1;
            };
            WriteLine(" ");
            i = i + 1;
        };
    };
};
```

Listing 27: matrix_multiplication.mom

```
$ run test.mom
1 2 3
4 5 6
7 8 9
 *
1 2 3
4 5 6
7 8 9
 =
30 36 42
66 81 96
102 126 150
```

13. Returns:

```
class Character is_a Component {
    Character() {
    };

    Nothing a() {
        Int b = 2 + 2;
        return;
    };

    Int b() {
        Int b = 2 * 2;
        return b;
    };

    Text c(Int length) {
        if(length <= 3) {
            return "THREE";
        } else {
            return "MORE THAN THREE";
        };
    };

    Nothing main() {
        Int x = b();
        Text y = c(5);
        WriteLine("b() returned:");
        WriteLine(x);
        WriteLine("c() returned:");
        WriteLine(y);
    };
};
```

Listing 28: returns.mom

```
$ run test.mom
b() returned:
4
c() returned:
MORE THAN THREE
```

14. Single_class:

```
class Number is_a Component {
    field<Int> aaa;

```

```
    Number() {
        aaa = 111;
    };

    Nothing f3() {
        WriteLine("NUMBER");
    };

    Real funName3() {
        return 0.5;
    };
};

class Complex is_a Number {
    field<Int> bbb;

    Complex() {
        bbb = 222;
    };

    Nothing f() {
        Number n = new Number();
        WriteLine("IN COMPLEX");
        n.f3();
    };

    Text funName() {
        return "Alice";
    };

    Real funName2() {
        Int b = 30;
        Number n = new Number();
        WriteLine(b);
        Real d = n.funName3();

        return b * d;
    };

    Int sum(Int a, Int b) {
        return a + b;
    };
};

class Character is_a Complex {
    field<Int> a;
    field<Int> ccc;

    Character() {
        a = 2;
        ccc = 333;
    };

    Nothing f2() {
        Write(a);
    };

    Int aNumber() {
        Int result = 1200 * 10;
        result = result + 345;
        return result;
    };

    Int bNumber() {
        return 1200 * 10;
    };

    Nothing main() {
        Creature x = new Creature();
        x.attack();
        Complex temp = new Complex();
        Text b = "HAHA";
        Text c = "!!!";
        Boolean d = TRUE && FALSE;
        Int num = 0;
        Write(a);
        WriteLine(3 + 5);
        WriteLine(3 + 2.3);
        WriteLine(b + c);
        WriteLine("HOHOHO");
        WriteLine(d);
        WriteLine(num + 3.3);
        temp.f();
        WriteLine(aaa);
        WriteLine(bbb);
        WriteLine(ccc);
        Int z = aNumber();
        WriteLine(z);
        WriteLine(aNumber());
        WriteLine(bNumber());
        WriteLine(temp.funName());
        WriteLine(temp.sum(2, 3) * temp.sum(6, 0) / (2 * temp.sum(1, 0)));
```

```
    };
};
```

Listing 29: single_class.mom

```
$ run test.mom
Attacking
28
5.3
AHA!!
HOHOHO
False
3.3
IN COMPLEX
NUMBER
111
222
333
12345
12345
12000
Alice
15
```

15. Specifications:

```
specification ComplexNumber {
    Int real();
    Int imaginary();
    Nothing convert(Int angle);
    Real summary(Int c, Real r);
};

class Class is_a Component {
    Class() {

    };

    Nothing main() {

    };
};
```

Listing 30: specification.mom

```
$ run test.mom
Im a Character of type Card and ComplexNumber
```

16. Test_interfaces:

```
specification ComplexNumber {
    Int real();
    Int imaginary();
```

```
    Nothing convert(Int angle);
    Real summary(Int c, Real r);
};

class BClass is_a Component {
    BClass() {

    };

    Int num() {
        return 333;
    };
};

class AClass is_a Component of_type ComplexNumber {
    field<ComplexNumber> number;

    AClass() {

    };

    Int myMethod() {
        return 345;
    };

    Int imaginary() {
        return 678;
    };

    Int real() {
        ComplexNumber numberTwo = new AClass();
        WriteLine("Created a complex number with interface AClass");
        BClass b = new BClass();
        return numberTwo.imaginary();
    };

    Nothing convert(Int angle) {

    };

    Real summary(Int c, Real r) {
        return 0.0;
    };

    Nothing main() {
        WriteLine(real());
    };
};
```

Listing 31: test_interfaces.mom

```
$ run test.mom
Created a complex number with interface AClass
678
```

17. Two_classes:

```
class Character is_a Component {
    Character() {
        WriteLine("We are on class 2");
    };

    Nothing func1() {
    };
};

class Card2 is_a Component {
    Card2() {
    };

    Int cardCost() {
        Int a = 2 * 3;
        return a;
    };

    Nothing main() {
        WriteLine("We are on class 1");
        Character x = new Character();
        x.func1();
    };
```

```
};
```

Listing 32: two_classes.mom

```
$ run test.mom
We are on class 1
We are on class 2
```

6 Code Listings

Since we used a different tool from everybody else, Antlr, our code works differently. In this section we present representative sections of our code. These are self explanatory and are commented in-line.

```python
    def enterArray_var(self, ctx: MoMParser.Array_varContext) -> None:
        """ Neuralgic point before an array is access.
            Generates a dimension and pushes it to the pending dims stack.
            Verifies that the acceded variable has the same dimensions.
        :param ctx: Context generated by antlr4.
        :return: None.
        """
        var = ctx.VARID().getText()
        text = ctx.getText()
        c = master_tables.classes[self.current_class]
        m = c.methods[self.current_method]

        # Look in local variables, if not, look in global variables
        if var in m.variables:
            dim = m.variables[var]["dim"]
        elif var in c.variables:
            dim = c.variables[var]["dim"]
        else:
            # if not present report error.
            raise NameError("Variable ' " + var + " is undefined.")

        # if there is nesting
        self.pending_dims.append((var, 1))
        self.pending_operators.append(Operator.OPEN_SPAREN)
        original_dim = len(dim)

        # count dimensions of accessed array
        actual_dim = 0
        stack_brackets = list()
        for c in text:
            if c == '[':
                stack_brackets.append('[')
            if c == ']':
                stack_brackets.pop()
                if len(stack_brackets) == 0:
                    actual_dim = actual_dim + 1

        if original_dim != actual_dim:
            raise NameError("Cannot convert var " + var + " of " + str(original_dim) + " dimensions to " + str(
                actual_dim) + " dimensions")
```

Listing 33: enterArray_var method

```python
    def enterClass_rule(self, ctx: MoMParser.Class_ruleContext) -> None:
        """This listener manages registering the classes that the user defines within a program.

        This listener is in charge of creating the class objects, specifying their names, parents
        and possible specifications.

        :param ctx: the context for this specific class instance.
        :return: None.
        """
        class_specifications = set()
        class_name = ctx.CLASSID()[0].getText()

        # It is a class with at least one specification
        for i in range(1, len(ctx.CLASSID())):
            specification_name = ctx.CLASSID()[i].getText()
```

```python

            if specification_name not in master_tables.specifications:
                raise NameError("Specification with name `" + specification_name + "` for class `" + class_name +
                                "` is undefined.")

            class_specifications.add(specification_name)

        self.current_class = class_name
        self.current_structure = StructureType.CLASS

        if class_name in master_tables.classes:
            raise NameError("Redefinition of class '" + class_name + "' found. This is not supported by the language.")

        if class_name in master_tables.classes:
            raise NameError("Name collision with interface '" + class_name +
                            "'. Classes cannot have the same name as interfaces.")

        # get the parent of this class
        self.enterComplex_type(ctx.complex_type())
        class_parent = self.current_type

        master_tables.classes[class_name] = Class(class_name, class_parent, class_specifications)

        ancestor = master_tables.classes[class_name].parent
        methods = master_tables.classes[ancestor].methods
        variables = master_tables.classes[ancestor].variables

        for var_n in variables:
            v = variables[var_n]
            if var_n not in methods:
                t = self.get_global_address_by_type(master_tables.classes[class_name], v["type"])
                self.increment_global_address_by_type(master_tables.classes[class_name], v["type"], v["mem_size"])
                master_tables.classes[class_name].add_argument(v["name"], v["type"], v["is_array"],
                                                               t, v["mem_size"], v["dim"], v["class_type"])

        for method_n in methods:
            method = methods[method_n]
            master_tables.classes[class_name].add_method(method)
            self.create_method_field(method.name, method.return_type)
```

Listing 34: enterClass_rule method

```python
    def enterConstruct_def(self, ctx: MoMParser.Construct_defContext) -> None:
        """ Neuralgic point when a Constructor is defined.
            Checks that there is only 1 Constructor of a Class.
            Generates quadruple GO_SUB that helps us find where the class begins.

        :param ctx: Context generated by antlr4.
        :return: None.
        """
        # reset virtual memory counters
        Method.cur_local_boolean = Method.LOCAL_BOOLEAN_TOP
        Method.cur_local_real = Method.LOCAL_REAL_TOP
        Method.cur_local_int = Method.LOCAL_INT_TOP
        Method.cur_local_text = Method.LOCAL_TEXT_TOP

        method_name = ctx.CLASSID().getText()
        self.current_method = method_name
        new_method = Method(method_name, Type.CLASS)
        new_method.start = len(self.quads)
        # Checks that there is just 1 constructor
        c = master_tables.classes[self.current_class]
        if method_name in c.methods:
            raise NameError("Method '" + method_name + "' redefined in class '"
                            + self.current_class + "', this is not supported at language level.")

        c.add_method(new_method)
        self.create_method_field(new_method.name, new_method.return_type)

        # Implicit calls to ancestor constructors
        if not c.name == "Component":
            p = master_tables.classes[c.parent]
            quad = Quadrupole(Operation.GO_SUB, self.current_class, c.parent, p.methods[p.name].start)

            self.quads.append(quad)
```

Listing 35: enterConstruct_def method

```python
    def enterFunction_call(self, ctx: MoMParser.Function_callContext) -> None:
        """ Listener when a function is called.
            Handles both scenarios: when the method is local/inherited and when it's from another class.

        :param ctx: Context generated by antlr4.
        :return: None.
        """
        class_instance = master_tables.classes[self.current_class]
        self.pending_operators.append(Operator.OPEN_PAREN)

        if ctx.THIS() is not None or len(ctx.VARID()) == 1:
```

```python
            # it is a local method, or inherited
            method_name = ctx.VARID()[0].getText()

            if method_name not in class_instance.methods:
                raise NameError("Method name `" + method_name + "` not defined for class: " + self.current_class)
            else:
                self.current_method_instance = class_instance.methods[method_name]

            self.current_counter = 0
        else:
            # it is a method from another class instance
            self.current_counter = 0
            var_name = ctx.VARID()[0].getText()
            func_name = ctx.VARID()[1].getText()
            # Looks for the variable first in local and then in global context
            if var_name in class_instance.methods[self.current_method].variables:
                c_ref = master_tables.classes[class_instance.methods[self.current_method].variables[var_name]["p_type"]]
                if func_name not in c_ref.methods:
                    raise NameError("Method name `" + func_name + "` not defined for class: " + c_ref.name)

                self.current_method_instance = c_ref.methods[func_name]
            elif var_name in class_instance.variables:
                if not class_instance.variables[var_name]["type"] == Type.CLASS:
                    raise TypeError("Function call not made from class reference.")

                c_ref = master_tables.classes[class_instance.variables[var_name]["class_type"]]
                if func_name not in c_ref.methods:
                    raise NameError("Method name `" + func_name + "` not defined for class: " + c_ref.name)

                self.current_method_instance = c_ref.methods[func_name]
            else:
                raise NameError("Variable `" + var_name + "` not defined in class: " + class_instance.name)

            self.class_reference = c_ref.name
```

Listing 36: enterFunction_call method

```python
    def enterProgram(self, ctx: MoMParser.ProgramContext) -> None:
        """ This is the first listener of a program where the class declarations,
            specifications and enums are declared and managed.
        :param ctx: Context generated by antlr4.
        :return: None.
        """
        # Create quadrupole that points to main method.
        quad = Quadrupole(Operation.GO_CONSTRUCTOR, None, None, None)
        self.quads.append(quad)
        quad = Quadrupole(Operation.GO_MAIN, None, None, None)
        self.quads.append(quad)

        # Register in tables the Component class, which is the base class of the language
        class_specifications = set()
        class_name = "Component"
        class_parent = ""
        master_tables.classes[class_name] = Class(class_name, class_parent, class_specifications)

        # reset virtual memory counters
        Method.cur_local_boolean = Method.LOCAL_BOOLEAN_TOP
        Method.cur_local_real = Method.LOCAL_REAL_TOP
        Method.cur_local_int = Method.LOCAL_INT_TOP
        Method.cur_local_text = Method.LOCAL_TEXT_TOP

        method_name = class_name
        self.current_method = method_name
        new_method = Method(method_name, Type.CLASS)
        new_method.start = len(self.quads)
        master_tables.classes[class_name].add_method(new_method)
        self.create_method_field_aux(class_name, new_method.name, new_method.return_type)

        quad = Quadrupole(Operation.RETURN, None, None, None)

        self.quads.append(quad)

        # The basic methods for the base class are width and height
        # reset virtual memory counters
        Method.cur_local_boolean = Method.LOCAL_BOOLEAN_TOP
        Method.cur_local_real = Method.LOCAL_REAL_TOP
        Method.cur_local_int = Method.LOCAL_INT_TOP
        Method.cur_local_text = Method.LOCAL_TEXT_TOP

        return_type, method_name = "Real", "getWidth"
        new_method = Method(method_name, get_type(return_type))
        new_method.start = len(self.quads)
        master_tables.classes[class_name].add_method(new_method)
        self.create_method_field_aux(class_name, new_method.name, new_method.return_type)

        # create width method quadrupoles
        quad = Quadrupole(Operation.RETURN, None, None, master_tables.classes[class_name].cur_global_real)

        self.quads.append(quad)

        Method.cur_local_boolean = Method.LOCAL_BOOLEAN_TOP
```

```
        Method.cur_local_real = Method.LOCAL_REAL_TOP
        Method.cur_local_int = Method.LOCAL_INT_TOP
        Method.cur_local_text = Method.LOCAL_TEXT_TOP

        return_type, method_name = "Real", "getHeight"
        new_method = Method(method_name, get_type(return_type))
        new_method.start = len(self.quads)
        master_tables.classes[class_name].add_method(new_method)
        self.create_method_field_aux(class_name, new_method.name, new_method.return_type)

        # create height method quadrupoles
        quad = Quadrupole(Operation.RETURN, None, None, master_tables.classes[class_name].cur_global_real + 1)

        self.quads.append(quad)
```

Listing 37: enterProgram method

```
    def exitArray_def(self, ctx: MoMParser.Array_defContext) -> None:
        """ Listener after an array is defined.
            Calculates r and Mi as seen in class with 2 sweeps.

        :param ctx: Context generated by antlr4.
        :return: None.
        """
        if self.in_signature:
            self.arguments[-1].is_array = True
        self.in_signature = False
        text = ctx.getText()
        dim = []
        r = 1
        pos_open = text.find("[")
        # Calculates r
        while pos_open != -1:
            pos_close = text.find("]", pos_open + 1)
            size = int(text[pos_open + 1:pos_close].strip())
            dim.append(size)
            r = r * size
            pos_open = text.find("[", pos_open + 1)

        dim_real = []
        self.arguments[-1].mem_size = r
        # Calculates M sub i
        for d in dim:
            dim_real.append((d, int(r / d)))
            r = r / d
        self.arguments[-1].dim = dim_real
```

Listing 38: exitArray_def method

```
    def exitAssignation(self, ctx: MoMParser.AssignationContext) -> None:
        """ Listener after an assignation occurs.
            Manages both: assignation of atomic variable and array variable assignation.

        :param ctx: Context generated by antlr4.
        :return: None
        """
        if ctx.VARID() is None:
            # array assignation handler
            holder = self.pending_operands.pop()
            type_holder = self.pending_types.pop()
            destination = self.pending_operands.pop()
            type_dest = self.pending_types.pop()
            if type_holder != type_dest:
                raise NameError("Cannot assign type " + get_name(type_holder) + " to type " + get_name(type_dest))
            quad = Quadrupole(Operator.EQUAL, holder, None, destination)

            self.quads.append(quad)
            return
        else:
            var = ctx.VARID().getText()

        c = master_tables.classes[self.current_class]
        m = c.methods[self.current_method]

        # Look in local variables, if not, look in global variables
        if var in m.variables:
            destination = m.variables[var]["address"]
            holder = self.pending_operands.pop()
            self.pending_types.pop()
            quad = Quadrupole(Operator.EQUAL, holder, None, destination)

            self.quads.append(quad)
        elif var in c.variables:
            destination = c.variables[var]["address"]
            holder = self.pending_operands.pop()
            self.pending_types.pop()
            quad = Quadrupole(Operator.EQUAL, holder, None, destination)

            self.quads.append(quad)
```

```
        else:
            # if not present report error.
            raise NameError("Variable ' " + var + " is undefined.")
```

Listing 39: exitAssignation method

```
    def exitClass_rule(self, ctx: MoMParser.Class_ruleContext) -> None:
        """ For all interfaces declared for current class,
            check that their methods are defined in the body of the class

        :param ctx: Context generated by antlr4.
        :return: None.
        """
        class_instance = master_tables.classes[self.current_class]
        specifications = class_instance.specifications

        for specification_name in specifications:
            specification = master_tables.specifications[specification_name]
            for method_name in specification.methods:
                if method_name not in class_instance.methods:
                    raise NameError("The method `" + method_name +
                                    "` is not implemented by the class: " + class_instance.name)

                class_method = class_instance.methods[method_name]

                for s_var, c_var in zip(specification.methods[method_name].variables, class_method.argument_types):
                    t = specification.methods[method_name].variables[s_var]["type"]
                    is_arr = specification.methods[method_name].variables[s_var]["is_array"]
                    if not t == c_var['arg_type']:
                        raise TypeError("Argument type mismatch in class `" + class_instance.name +
                                        "`, method `" + method_name + "`. Expected: " + t +
                                        ", got " + c_var['arg_type'] + " instead.")

                    if not is_arr == c_var["is_array"]:
                        raise TypeError("Argument type mismatch in class `" + class_instance.name +
                                        "`, method `" + method_name + "`. Expected: " + t +
                                        "[], got " + c_var['arg_type'] + " instead.")

                # TODO: check for return type, num arguments, name, type arguments, if they are arrays or not in tests
                r = specification.methods[method_name].return_type
                if not r == class_method.return_type:
                    raise TypeError("Return type mismatch in class `" + class_instance.name +
                                    "`, method `" + method_name + "`. Expected: " + str(r) +
                                    ", got " + str(class_method.return_type) + " instead.")

        quad = Quadrupole(Operation.END_CLASS, None, None, None)

        self.quads.append(quad)
```

Listing 40: exitClass_rule method

```
    def exitFunction_call(self, ctx: MoMParser.Function_callContext) -> None:
        """ Neuralgic point after a function is called.
            Checks if it was a same-class call or was another's class function.

        :param ctx: Context generated by antlr4.
        :return: None.
        """
        self.pending_operators.pop()
        # Checks that the exact number of arguments were given.
        if not self.current_counter == self.current_method_instance.num_of_params:
            raise IllegalStateException("Method `" + self.current_method_instance.name +
                                        "` has wrong number of arguments. Should be " +
                                        str(self.current_method_instance.num_of_params) +
                                        ", got " + str(self.current_counter) + " instead.")

        if self.class_reference == "":
            # Same class call
            quad = Quadrupole(Operation.GO_SUB, self.current_class, self.current_method_instance.name,
                              self.current_method_instance.start)
            self.quads.append(quad)
            # Generates quadruple for the return variable of the function.
            if not self.current_method_instance.return_type == Type.NOTHING:
                m = master_tables.classes[self.current_class].methods[self.current_method]
                result = self.get_temp_address_by_type(m, self.current_method_instance.return_type)
                self.increment_temp_address_by_type(m, self.current_method_instance.return_type)
                # noinspection SpellCheckingInspection
                addr = master_tables.classes[self.current_class].variables[self.current_method_instance.name]["address"]

                quad = Quadrupole(Operator.EQUAL, addr, None, result)
                self.quads.append(quad)
                self.pending_operands.append(result)
                self.pending_types.append(self.current_method_instance.return_type)
        else:
            var = ctx.VARID()[0].getText()
            c = master_tables.classes[self.current_class]
            m = c.methods[self.current_method]

            # Look in local variables, if not, look in global variables
```

```
            if var in m.variables:
                address = m.variables[var]["address"]
            elif var in c.variables:
                address = c.variables[var]["address"]
            else:
                # if not present report error.
                raise NameError("Variable ' " + var + " is undefined.")
            quad = Quadrupole(Operation.GO_SUB, str(address)+":"+self.class_reference, self.current_method_instance.name,
                              self.current_method_instance.start)
            c = master_tables.classes[self.class_reference]
            self.class_reference = ""
            self.quads.append(quad)

            if not self.current_method_instance.return_type == Type.NOTHING:
                m = master_tables.classes[self.current_class].methods[self.current_method]
                result = self.get_temp_address_by_type(m, self.current_method_instance.return_type)
                self.increment_temp_address_by_type(m, self.current_method_instance.return_type)

                quad = Quadrupole(Operation.RETRIEVE, str(address) + ":" + str(c.variables[self.current_method_instance.name]["address"]),
                                  self.current_method_instance.name,
                                  result)

                self.quads.append(quad)
                self.pending_operands.append(result)
                self.pending_types.append(self.current_method_instance.return_type)
```

Listing 41: exitFunction_call method

```
    def exitProgram(self, ctx: MoMParser.ProgramContext) -> None:
        """ Exit of the first listener.
        :param ctx: Context generated by antlr4.
        :return: None
        """

        # Creates last quadruple - End of program
        quad = Quadrupole(Operation.END, None, None, None)
        self.quads.append(quad)

        # Verifies there is a main function
        if not self.main_found:
            raise RuntimeError("Main method not found, please define program entry point.")

        # Prints all the generated quadruples to the console
        for index, quad in enumerate(MoMListener.quads):
            print(str(index) + ") " + str(quad.operator) + ", " + str(quad.left_operand) + ", "
            + str(quad.right_operand) + ", " + str(quad.result))
```

Listing 42: exitProgram method

```
    def exitVdim(self, ctx: MoMParser.VdimContext):
        """ Neuralgic point for a declared n-dimension array.
            Calculates r and then Mi as seen in class with 2 sweeps.

        :param ctx: Context generated by antlr4.
        :return: None.
        """
        if self.in_signature:
            self.arguments[-1].is_array = True
        self.in_signature = False

        text = ctx.getText()
        dim = []
        r = 1
        pos_open = text.find("[")
        # Looks for every bracket that represents a dimension
        # Calculates r
        while pos_open != -1:
            pos_close = text.find("]", pos_open + 1)
            size = int(text[pos_open + 1:pos_close].strip())
            dim.append(size)
            r = r * size
            pos_open = text.find("[", pos_open + 1)

        dim_real = []
        self.arguments[-1].mem_size = r
        # Calculates M sub i.
        for d in dim:
            dim_real.append((d, int(r / d)))
            r = r / d
        self.arguments[-1].dim = dim_real
        # Generates the corresponding memory for the whole array.
        for name, var in zip(self.argument_names, self.arguments):
            if self.current_structure == StructureType.CLASS:
                m = master_tables.classes[self.current_class].methods[self.current_method]
                address = self.get_address_by_type(m, get_type(var.var_type))
                t = get_type(var.var_type)

```

```
                if t == Type.CLASS:
                    m.add_argument(name, t, var.is_array, address, var.mem_size, var.dim, var.var_type)
                else:
                    m.add_argument(name, t, var.is_array, address, var.mem_size, var.dim)
                m.add_argument_type(get_type(var.var_type), var.is_array)
                self.increment_address_by_type(m, get_type(var.var_type), var.mem_size)
            elif self.current_structure == StructureType.SPECIFICATION:
                master_tables.specifications[self.current_specification].methods[
                    self.current_method].add_argument(name, get_type(var.var_type), var.is_array, -2, var.mem_size)
```

Listing 43: exitVdim method

7 User manual

User manual is located at https://github.com/mpeyrotc/MoM-Adventure-Language.

References

[1] F. Flight. Mansions of madness second edition. https://www.fantasyflightgames.com/en/products/mansions-of-madness-second-edition/. [Online; accessed September 5th, 2017].

[2] T. Parr. About the antlr parser generator. http://www.antlr.org/about.html. [Online; accessed September 5th, 2017].

[3] T. P. L. Reference. Lexical analysis. https://docs.python.org/3/reference/lexical_analysis.html#identifiers, 2017. [Online; accessed September 5th, 2017].

Appendices