ALGONQUIN COLLEGE

SCHOOL OF ADVANCED TECHNOLOGY

ICT - Applications & Programming
Computer Engineering Technology – Computing Science

A31

Model Definitions (RE/Automaton)

Lab Professor / Lab Session:
Anurag Das - 011

Team:
Henry Forget - Id: 041023812 / Colin Tapp – Id: 040774946

Language Name [Mouse]

ALGONQUIN COLLEGE

Mouse - Language Specification

General View

1. The Mouse Language Lexical Specification

1.1. White Space

White space is defined as the ASCII characters: space, tab space, and form feed. As well as the line terminators: new line, and carriage return. White space is discarded by the scanner except tab spaces.

<white space> → *one of* { SPACE, TAB, FF, NL, CR }

1.2. Tab Space

Mouse Language supports code blocks using tabulation. When the scanner detects a lone tab space character, it produces a single TAB_T token.

<tab space> → TAB_T

1.3. Comments

Mouse Language supports only single-line comments: all the text from the ASCII character '**#**' to the end of line produces a single token: CMT_T.

<comments> → CMT_T → # { sequence of ASCII chars } NL

1.4. Variable Identifiers

Variable identifiers (VID) are any string of text starting with a letter character, containing only letters, digits, and underscore characters, that does not match a keyword.A VID found by the scanner produces a VAR_T token.

<variable identifier> → VAR_T → L(L|D|U)*

1.5. Keywords

Any VID matching a keyword from the keywordTable[] will be assigned a KW_T. The list of keywords in **Mouse** language is given by:

Int, float, str, bool, True, False, None, if, elif, else, while, for, def, return, break, continue, pass, and, or

1.6. Integer Literals

The scanner produces a single token: **INL_T** with an integer value as an attribute.

<integer_literal> → INL_T

1.7. Floating-point Literals

FLT_T token with a real decimal value as an attribute is produced by the scanner.

<float_literal> → FLT_T

1.8. String Literals

STR_T token is produced by the scanner.

<string_literal> → STR_T

1.9. Separators

<separator> → *one of* { (,), ,, : }

Tokens produced by the scanner - **LPR_T**, **RPR_T**, **COM_T, COL_T**.

1.10. Operators

OP_T token is produced by the scanner. Operators are divided into 3 subcategories: arithmetic, relational and logical. Each distinct operator has a token that is assigned as an attribute to the overlying operator token.

<operator> → *one of* { ArithmeticOperators, RelationalOperators, LogicalOperators }

1.10.1. Arithmetic Operators

<operator> → *one of* { +, -, *, /, %, ^ }

1.10.2. Relational Operators

<operator> → *one of* { =, != , >, < }

1.10.3. Logical Operators

<operator> → *one of* { &, |, ! , == }

2. The Mouse Language Syntactic Specification

2.1. Mouse Language Program

2.1.1. Program

In Mouse language, the program requires no main method. All code will be compiled and executed from top to bottom.

<program> → **code**

2.1.2. Code

The code of the Mouse language can be split into two categories. Either Data (variable assignment) or Statements (everything else). Neither is required for a program to compile.

<code> → **{** **opt_data | ϵ** **opt_statements | ϵ** **}**

2.1.3. Optional Data

Variables can be declared anywhere in the program. They can be implicitly or explicitly typed depending on how they are declared.

<opt_data> → {
<implicitly_typed_var>
<explicitly_typed_var>
}

Implicitly Typed Vars

Variables can be implicitly typed, and are defined as: variable = value

<implicitly_typed_var> → <VAR_T> <OP_EQ> <value>

Explicitly Typed Vars

Variables can also be explicitly type, and defined as: variable:dataType = value

<explicitly_typed_var> → <VAR_T> <COL_T> <datatype_keyword> <OP_EQ> <value>

2.1.4. Variable Data Types:

The explicit type of a variable can be defined using one of the keywords in this list:

Data Type Keywords:

<datatype_keyword> → {
<integer> → "int"
<floating_point> → "float"
<string> → "string"
<boolean> → "bool"
}

2.1.5. Variable Types:

The 'value' a variable can be set equal to can be any among the following list:

Integers:

<integer_variable> → INL_T

Floating-points:

<float_variable> → FLT_T

Strings:

<string_variable> → STR_T

Booleans:

<boolean_variable> → BLN_T

2.1.6. Optional Statements

<opt_statements> → <statements> | ϵ

2.1.7. Statements

<statements> → <statement> | <statements> <statement>

2.2. Statement

<statement> → <assignment statement> | <selection statement> | <iteration statement>
| <input statement> | <output statement>

2.2.1. Assignment Statement

<assignment statement> → <assignment expression>

2.2.2. Assignment Expression

<assignment expression> → <integer_variable> = <arithmetic expression>
| <float_variable> = <arithmetic expression>
| <string_variable> = <string expression>
| <boolean_variable> = <boolean expression>

2.2.3. Selection Statement (if statement)

<selection statement> → **if** (<conditional expression>):
('\t') <opt_statements>
<optional elif statement>
<optional else statement>

<optional elif statement> → **elif** (<conditional expression>):
('\t')<opt_statements> | ϵ
<optional elif statement>
<optional else statement>

<optional else statement> → **else** ('\t')<opt_statements> | ϵ

2.2.4. Iteration Statement (the loop statement)

<iteration statement> → **while** (<conditional expression>):
('\t')<statements>

2.2.5. Input Statement

<input statement> → **input** (<variable list>);

Variable List:

<variable list> → <variable identifier> | <variable list>,<variable identifier>

Variable Identifier:

<variable identifier> → <integer_variable>
| <integer_variable>
| <float_variable>
| <string_variable>

2.2.6. Output Statement

<output statement> → **print** (<opt_variable list>) | **print** (STR_T);

- **PROBLEM DETECTED: Left factoring – SOLVING FOR YOU:**

<output statement> → **print** (<output statement Prime>)
<output statement Prime> → <opt_variable list> | STR_T

Optional Variable List:

<opt_variable list> → <variable list> | ϵ

2.3. Expressions

2.3.1. Arithmetic Expression

<arithmetic expression> → <unary arithmetic expression> | <additive arithmetic expression>

Unary Arithmetic Expression:

<unary arithmetic expression> → - <primary arithmetic expression>
| + <primary arithmetic expression>

Additive Arithmetic Expression:

<additive arithmetic expression> →
<additive arithmetic expression> + <multiplicative arithmetic expression>
| <additive arithmetic expression> - <multiplicative arithmetic expression>
| <multiplicative arithmetic expression>

Multiplicative Arithmetic Expression:

<multiplicative arithmetic expression> →
<multiplicative arithmetic expression> * <primary arithmetic expression>
| <multiplicative arithmetic expression> / <primary arithmetic expression>
| <primary arithmetic expression>

Primary Arithmetic Expression:

<primary arithmetic expression> → <integer_variable>
| <float_variable>
| FPL_T | INL_T
| (<arithmetic expression>)

2.3.2. String Expression

<string expression> →
<primary string expression> | <string expression> + <primary string expression>

Primary String Expression:

<primary string expression> → <string_variable> | STR_T

2.3.3. Conditional Expression

<conditional expression> → <logical OR expression>

Logical OR Expression:

<logical OR expression> → <logical AND expression>
| <logical OR expression> **||** <logical AND expression>

Logical AND Expression:

<logical AND expression> → <logical NOT expression>
| <logical AND expression> **&&** <logical NOT expression>

Logical NOT Expression:

<logical NOT expression> → **!** <relational expression>
| <relational expression>

2.3.4. Relational Expression

<relational expression> →
<relational a_expression> | <relational s_expression>

Relational Arithmetic Expression:

<relational a_expression> →
<primary a_relational expression> == <primary a_relational expression>
| <primary a_relational expression> <> <primary a_relational expression>
| <primary a_relational expression> > <primary a_relational expression>
| <primary a_relational expression> < <primary a_relational expression>

Relational String Expression:

<relational s_expression> →
<primary s_relational expression> == <primary s_relational expression>
| <primary s_relational expression> <> <primary s_relational expression>
| <primary s_relational expression> > <primary s_relational expression>
| <primary s_relational expression> < <primary s_relational expression>

Primary Arithmetic Relational Expression:

<primary a_relational expression> → <integer_variable> | <float_variable> | FPL_T | INL_T

<primary s_relational expression> → <primary string expression>