Malice Specification

Martin Donlin, Le Thanh Hoang, Eleanor Vincent

October 31, 2012

1 Grammars

1.1 Context-Free Grammar and Notation

In this specification we will describe Malice's syntax by using the Context-Free Grammar notation, Backus-Naur Form. We will be detailing the Syntactic Grammar(Ch 2) and Lexical Structure(Ch 3).

1.1.1 Notation

Each instruction consists of a left Terminal Node (representated by *italic*) followed by "::=" and a combination of one or more Non-terminal Nodes (representated by straight text and surround by " if the node is a literal string) or Terminal Nodes. We will be using '|' to represent a choice between instructions and '..' to represent the choice range. e.g "A..C" means A | B | C.

2 Syntactic Grammar

2.1 Main Function

The main function in Malice comprises of the key phrase "The looking-glass hatta' followed by any number of arguments enclosed in an open and close parenthesis. To start typing code into the main function, an 'opened' keyword must be used and a 'closed' keyword must be used to finish the main function.
BNF representation: Ch 6.0.2

2.2 Programs And Statements

Programs consist of any number of Statements including none at all. The first statement in a program (if any) must not have the 'too.' seperator at the end. Any statements following the first statement can have any seperator. Statements consist of a Declaration(Ch 4.1), an Assignment(Ch 4.2), or a stepping instruction(Ch 3.5) followed by a Seperator(Ch 3.6).
BNF representation: Ch 6.0.3 & 6.0.4

3 Lexical Structure

3.1 Keywords/Keyphrases

Malice has a number of reserved keywords/keyphrases. These keywords cannot be used for naming Identifiers(Ch 3.3). The keywords are as follows (seperated by '|'):

ate | drank | and | but | then | too | number | letter | was a | became | said Alice | the looking glass hatta | opened | closed

3.2 Literals

3.2.1 Numbers

In Malice there is a type 'number' which is defined as a 4 byte, base 10, integer. Where byte is defined as the size of a char in the architecture. The internal representation of a number uses 2's Complement. A number can consist of any number of digits so long as it doesn't start with a 0.

Number := Non-Zero-Digit Digits | Digit
Digits:= Digit | Digit Digits
Digit := 0 | Non-zero-Digit
Non-Zero-Digit := 1..9

3.2.2 Letters

In Malice there is a type 'letter' which is defined as a byte. Where byte us defined as the size of a char in the architecture. A letter can consist of any letter from A to Z or a to z.

Letter ::= A..Z | a..z (see notation)

3.3 Identifiers

An identifier name can be made up of any number of Malice letters and any number of Malice digits so long as the identifier name does not start with a number. An identifier name can be defined as follows:
IdentName ::= letter letterOrNum
letterOrNum ::= letter letterOrNum | number letterOrNum | ”

3.4 Operators

Malice has a number of binary and unary operators that only work on numbers. The precedence of operators are defined as follows:

Operation	Symbol	Level Of Precedence
Bitwise Not	~	0
Multiply	*	1
Divide	/	1
Mod	%	1
Add	+	2
Subtract	-	2
Bitwise And	&	3
Bitwise Xor	^	4
Bitwise Or	|	5

3.5 Stepping

Malice has two stepping functions. The keyword 'drank' decrements a variable and the keyword 'ate' increments a variable. Both functions store the stepped value back into the variable.
BNF representation: Ch 6.0.7

3.6 Seperators

Malice has a number of seperators that can be used to seperate statements(Ch 2.2). A statement can be finished by any seperator so long as the statement is not the first statement nor the last. There are pre-defined rules for the first and last statement such that the first statement cannot finish with the 'too.' seperator and the last statement must finish with a full stop.
BNF representation: Ch 6.0.8

3.7 Printing

To print values to the console, Malice uses the keyphrase 'said Alice' which must be preceeded by an expression of terms, an identifier or a combination of both.
BNF representation: Ch 6.0.9

4 Variables

4.1 Declaration

Variables are declared in Malice using the keyphrase 'was a' which must be preceeded by a new identifier name and followed by either the keyword 'number' or 'letter'
BNF representation: Ch 6.0.5

4.2 Assignment

Variables are assigned values in Malice by using the keyword 'became' preceeded by an existing identifier and followed by a expression of the same type.
BNF representation: Ch 6.0.5

5 Error Messages

5.1 Semantic Rules

1. Assignments of values into variables must have the same type
2. Assignments can only be done to existing variables
3. Casting from a letter to a number is not allowed
4. Stepping instructions can only be applied to existing variables, that has already been assigned a value and is of type number.
5. The same variable name cannot be declared twice
6. Variables but not have the same name as any of the keywords
7. All variable names must start with a letter followed by any number of digits or letters
8. Operators only work on variables of type number or literals of type number

5.1.1 Semantic error messages

Type Clash in assignment on line #. One type is -type1- and the other is -type2-.
This error message indicates the user has tried to assign a value into a variable that is of different type. i.e type1 and type2 are different.

6 BNF Representation

6.0.2 Main Function

Main ::= 'The looking-glass hatta' '(' ')' 'opened' *StatementList* 'closed'

6.0.3 Programs

StatementList ::= *FirstStatement FollowStatements* | *Print*

6.0.4 Statements

FirstStatement ::= *DeclareAssignStatement FirstSeperator* | *Step FirstSeperator*
FollowStatement ::= *DeclareAssignStatement FollowSeperator* | *Step FollowSeperator* | ''
FollowStatements ::= *FollowStatement FollowStatements* | *FollowStatement*

6.0.5 Declaration and Assignment

DeclareAssignStatement ::= Ident *DeclareAssign*
DeclareAssign ::= 'was a' *Type* | 'became' *Var*

6.0.6 Type

Type ::= 'number' | 'letter'

6.0.7 Stepping

Step ::= Ident *StepChoice*
StepChoice ::= 'drank' | 'ate'

6.0.8 Seperators

ConnTerms ::= 'and' | 'but' | 'then' | ','
EndTerm ::= '.'
FirstSeperator ::= *EndTerm* | *ConnTerms*
FollowSeperator ::= *FirstSeperator* | 'too' *EndTerm*

6.0.9 Printing

Print ::= *Val* 'said Alice' *EndTerm*

6.0.10 Values

Val ::= *Exp* | *Char*

6.0.11 Operators

Exp ::= *Exp* '|' *Xors* | *Xors*
Xors ::= *Xors* '^' *Ands* | *Ands*
Ands ::= *Ands* '&' *Sums* | *Sums*
Sums ::= *Sums* '+' *Multi* | *Sums* '-' *Multi* | *Multi*
Multi ::= *Multi* '*' *Factor* | *Multi* '/' *Factor* | *Multi* '%' *Factor* | *Factor*
Factor ::= Number | Ident | '~' *Factor*