MAlice Language Specification

Ethel Bardsley Joe Slade Thomas Wood

November 2, 2010

1 BNF Grammar

Program	→	Statements Output
Statements	→	Statement Terminator Statements
Terminator	→	‘`,`’ | ‘`.`’ | ‘`and`’ | ‘`but`’ | ‘`then`’
Output	→	‘`Alice found`’ Exp ‘`.`’
Statement	→	*Id* ‘`was a`’ Type Too
	|	*Id* ‘`became`’ Exp
	|	*Id* ‘`ate`’
	|	*Id* ‘`drank`’
Type	→	‘`number`’ | ‘`letter`’
Too	→	‘`too`’ | ϵ
Exp	→	Exp ‘`|`’ Exp1 | Exp1
Exp1	→	Exp1 ‘`^`’ Exp2 | Exp2
Exp2	→	Exp2 ‘`&`’ Exp3 | Exp3
Exp3	→	Exp3 ‘`+`’ Exp4
	|	Exp3 ‘`-`’ Exp4
	|	Exp4
Exp4	→	Exp4 ‘`*`’ Exp5
	|	Exp4 ‘`/`’ Exp5
	|	Exp4 ‘`%`’ Exp5
	|	Exp5
Exp5	→	‘`~`’ Exp5 | Val
Val	→	*Int* | *Id*
	|	‘`’` *Character* ‘`’`

- *Int* is an integer, matching the regular expression pattern `[0-9]+`
- *Id* is a variable identifier, matching `[a-zA-Z_]+`

- *Character* is any ASCII character

2 Semantics

2.1 Types

2.1.1 Number

Numbers are signed integers of length 32 bits. Underflow and overflow are undefined behaviours. All operators listed in the operators section can be used.

However, when returned via the program's exit code, all numbers will be returned as unsigned 8 bit integers, from the lowest 8 bits of the number.

2.1.2 Letter

Although `letter` appears as a type in the given examples, there are no working exapmles in which its functionality is exhibited. Consequently, nothing can be inferred about this possible type, including whether it is a valid type or not!

However, it was stated at a later date that the letter type is to be stored as an 8-bit ASCII value. This means that it is a valid type. The BNF above has been modified to include it.

The given examples infer that it is not permitted to mix number and letter types over a single operation. Nothing can be inferred about how the letter type should be returned, nor what operators it is compatible with.

2.2 Statements

An Alice program is defined as a list of statements followed by the output statement.

2.2.1 Output

The `Alice found` statement is analogous to the return statement of other languages. It evaluates its parameter (an expression) and returns the value.

For example:

```
Alice found 3.
```

returns the value 3.

2.2.2 Declaration

The `was a` statement declares the preceding identifier as a variable of the given type.

Declaring the same variable name multiple times is not permitted and will result in a compile-time error.

The keyword `too` may be placed at the end of this statement. No meaning could be inferred from the examples given, so none has been assumed at this point.

For example:
```
x was a number
```
declares a variable called x as a number

2.2.3 Assignment

The `became` statement assigns the value of an expression to the given variable.

The type of the expression must match the type of the variable, otherwise a compile-time error will result.

For example:
```
x became 5
```
assigns 5 to x.

2.2.4 Increment and Decrement

The `drank` statement decrements the given variable by 1.

The `ate` statement increments the given variable by 1.

For example:
```
x drank
```
if x is 5, x will become 4

For example:
```
x ate
```
if x is 5, x will become 6

2.3 Expressions

Operator	Operation	Precedence
`|`	Bitwise OR	1
`&`	Bitwise XOR	2
`^`	Bitwise AND	3
`+`	Addition	4
`-`	Subtraction	4
`*`	Multiplication	5
`/`	Division	5
`%`	Modulo	5
`~`	Bitwise NOT	6

- Numerically higher precedences bind more tightly.
- All operators are mathematically associative, and implemented as left-associative.
- Division by 0 is undefined and will be handled by the operating system.
- All operators are binary, except for Bitwise NOT which is unary.
- All operator precedences (except for Bitwise NOT) taken from example programs given, including the additional ones provided after the Milestone 1 deadline. Bitwise NOT presumed to have the highest precedence, since it is the only Unary Operator.
- Underflow and overflow conditions are undefined.