GEKO

Gökberk Altıparmak 21901798 SEC-02

Ege Ayan 22002478 SEC-01

27.10.2022

1. BNF of Language

<program> → <stmt_list>

<stmt_list> → <stmt> | <stmt> <stmt_list>

<stmt> → <decl_stmt> | <non_decl_stmt>

<decl_stmt> → <var_decl> | <func_decl>

<var_decl> → <var> ; | <identifier> , <var_decl>

<args> → <identifier> | <var> , <args>

<func_decl> → function <identifier> (<args>) { <stmt_list> }

<non_decl_stmt> → <arith_expr_stmt> | <funct_call> | <bool_expr_stmt> | <loop_stmt> | <if_stmt> | <return_stmt>

| <print_stmt> | <assign_stmt> | <comment_stmt> | <primitive_funct> | <mechanism_funct> |

<bool_expr_stmt> → <bool_expr> | <bool_expr_stmt> <logic_op> <bool_expr>

<bool_expr> → <var> <bool_op> <var>

<arith_expr> → <term> | <arith_expr> <add_op> <term>

<term> → <factor> | <term> <mul_op> <factor>

<factor> → <var> | (<arith_expr>)

<funct_call> → <identifier> (<args>) ;

<assign_stmt> → <identifier> = <var> ; | <identifier> = <funct_call> ; | <identifier> = <arith_expr> ;

<return_stmt> → return <expr_stmt> ;

<expr_stmt> → <string> | <var>

<if_stmt> → <matched> | <unmatched>

<matched> → if (<bool_expr>) { <matched> } else { <matched> } | -> <non_decl_stmt>

<unmatched> → if (<bool_expr>) { <stmt> } | if (<bool_expr>) { <matched> } else { <unmatched> }

<print_stmt> → print (<expr_stmt>) ;

<primitive_funct> → <read_sensor> | <read_time>

<read_sensor> → readSensor (<sensor>) ;

<read_time> → readTime ( ) ;

<mechanism_funct> → <connect_URL> | <send_to_URL> | <receive_from_URL>

<connect_URL> → connectToURL (<string>) ;

<send_to_URL> → sentToURL (<string> , <number>) ;

<receive_from_URL> → receiveFromURL (<string>) ;

<loop_stmt> → <for_stmt> | <while_stmt>

<while_stmt> → while (<bool_expr>) { <stmt_list> }

<for_stmt> → for (<assign_stmt> <bool_expr> ; <assign_stmt>) { <stmt_list> }

<comment_stmt> → // <comment>

<comment> → <char> | <comment> <char>

<sensor> → sensor1 | sensor2 | sensor3 | sensor4 | sensor5 | sensor6 | sensor7 | sensor8 | sensor9 | sensor10

<identifier> → <char> | <identifier> <alphanumeric>

<alphanumeric> → <char> | <alphanumeric> <char> | <alphanumeric> <digit>

<string> → <char> | <digit> | <char> <string> | <digit> <string> |

<digit> → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<char> → a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z

| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z

<number> → <digit> | <number> <digit>

<var> → <identifier> | <number>

<bool_op> → <= | >= | == | < | > | !=

<add_op> → + | -

<mul_op> → * | / | %

<logic_op> → && | ||

2. Description of Language Constructs

<program>

Starting literal of the language. Consist of statement list.

<stmt_list>

List of statements that contains at least one statement.

<stmt>

Statement that consists of either declaration statement or non-declaration statement

<decl_stmt>

Declaration statement for declaring either variable or function.

<non_decl_stmt>

Any other statements rather than declaration statements such as arithmetic expression statement, bool expression statement, loop statement etc.

<funct_decl>

Defines C type function that takes one or multiple arguments as a parameter and return desired value.

<args>

Arguments that are passed into the functions in order to be used.

<var_decl>

Any type of variable declaration. The type of the variable is not specified while declaring as in Python.

<non_decl_stmt>

Consist of 12 different type of statements that will be explained detailly below.

<arith_expr_stmt>

Five different arithmetic expression which are addition, subtraction, multiplication, division and modulo operations that has either number or variable as operands.

Precedence of the operations are arranged as multiplication = division = modulo > addition = subtraction.

<term>

It accepts either one factor or multiple factors that are attended to multiplication/division/modulo operations.

<factor>

It accepts identifier or arithmetic expression inside brackets.

<bool_expr_stmt>

One or multiple connected boolean expressions with or without logical operations which are and(&&), or(||).

<bool_expr>

Single boolean statement with boolean operands which are <, >, ==, !=, >=, <=

<loop_stmt>

Usual C type loop consist of for or while loops.

<while_stmt>

Generic C type while statement.

<for_stmt>

Generic C type for statement.

<if_stmt>

Generic C type if statement. In order to prevent the ambiguity of if-else statements, matched and unmatched conditions are used.

<matched>

If statement with else statement.

<unmatched>

If statement without else statement.

<return_stmt>

Returns the desired value ie. output of a function.

<print_stmt>

Prints the desired value.

<assign_stmt>

Assigns the desired value into the desired variable.

<comment_stmt>

It either accepts comment line or comment block.

Comment statements that will not be executed. In order to provide for the programmer to take notes.

<primitive_funct>

It either reads from sensors or timestamp.

<read_sensor>

Function that reads the sensor from given sensor.

<read_time>

Function that reads the time.

<mechanism_funct>

Consist of three functions which are ConnectToURL, ReceiveFromURL, SendToURL.

<connect_to_URL>

It makes connection to given URL.

<send_to_URL>

It sends an integer value to given URL.

<receive_from_URL>

It receives an integer from URL.

<function_call>

Calls the function.

<expr_stmt>

It either accepts string or variable.

<string>

Array of characters.

<identifier>

Single character or alphanumeric.

<alphanumeric>

Combinations of characters and digits.

<var>

It accepts identifier or number

<number>

Combinations of digits.

3. Terminals

<: Smaller than operation.

>: Greater than operation.

=: Assign operator.

==: Equality operator.

<=: Less than or equal operator.

>=: Greater than or equal operator.

+: Addition operator.

-: Subtraction operator.

*: Multiplication operator.

/: Division operator.

%: Modulo operator.

,: Used to separate arguments.

;: Points the end of the line.

(): Parentheses are used to group expressions.

{}: Curly brackets are used to define function statements.

&&: And logical operator.

||: Or logical operator.

//: Line comment indicator.

->: Indicates statement in matched if-else block.

Digits: Numbers from 0-9 inclusive.

Char: All characters from English alphabet.

4. Nontrivial Tokens

4.1 Identifiers

Identifier is a group of one or more characters and digits. Identifier cannot start with digit. We used C types of identifiers to increase the readability and writability of the language.

Identifier Example: example_Identifier123sad

4.2 Comments

Comments in our language can either be a single line or a block. For the single line situation, “//” can be used. Language will recognize the line as a comment when it sees the notation. For the block comment, it comments the everything between “/*” and “*/”. In order to increase the readability and writability different types of comments has used.

Comment Example 1) // Line Comment

Comment Example 2) /* Block

Comment */

4.3 Literals

Digit: Single number in the range 0-9 inclusive. In our language digits can be combined to form integers and real numbers.

Char: All characters from English alphabet. Chars can be combined to form strings. In addition, chars and digits can be combined in a way to form alphanumeric and identifiers.

4.4 Reserved Words

if: Used in if-else statements.

else: Used in if-else statements.

while: Used in while loops.

for: Used in for loops.

return: Used in functions.

print: Used to print.

function: Used to indicate the function.

readSensor: Used to indicate the specified function.

readTime: Used to indicate the specified function.

connectToURL: Used to indicate the specified function.

sendToURL: Used to indicate the specified function.

receiveFromUrl: Used to indicate the specified function.

5. Language Design Criteria

5.1 Readability

The way that variables and functions are defined such that their type is not identified. This type of approach might reduce readability in terms of understanding the type of variable that is read by the user. On the other hand, loop structure is similar to C group languages and thus easier to follow by someone who has worked with those kinds of languages before. Identifier names are very similar with as those are in Java that they cannot start with a digit and consists of alphanumeric characters. Arithmetic operations and operators are also similar to many popular programming languages which increases the readability. However, while declaring a function it's return type will not be specified so the reader might have hard time while understanding the return value of a function, especially when that return value is used to assign a value to a variable.

5.2 Writability

The language has very easy and simple way of declaring variables and functions. The type of a variable or the return type of a function is not specified while declaring which makes it very easy to write code in this language. Passing arguments, loops, conditional statements are designed alike the way many popular programming languages use. While writing, the programmer does not have to worry about the things such as type mismatch since everything can be assigned to the variables. In terms of writability, the language has no major weakness and language itself can be considered as the synthesis of C group languages (loops, if statements, arithmetic operations, identifier name convention) and Python (variable and function declarations) in order to implement a highly writable programming language.

5.3 Reliability

The recursive definitions of the variables are defined such that they are either right recursive or left recursive in order to prevent ambiguity in the language. However, since the language is easy to implement and has high writability due to unspecific variable types, the language can be ambiguous and there can be different parse trees for the same program. Also, the unspecific variable types and unspecific return types of the functions might cause logic errors inside the program (e.g., String is used in a division operation) which can be considered as the cost of increasing writability. Moreover, the language has no method for handling exceptions which decreases the reliability of the language.

6. Conflicts

Our program totally has four shift/reduce conflict. Other than that, there is no any reduce/reduce conflict. These minor shift/reduce conflicts stem from identifying every reserved keyword as identifier. For example, while function is considered as custom function since the word while can be both keyword and identifier. The same goes for if and other primitive functions. In addition, <var> contains both number and identifier which causes them to conflict within each other since var has very broad range of use.