IOTA

CS315 - Project 2 - Group 14

7.11.2019

Team Members:

1. Hanzallah Azim Burney - 21701829 - Section 01
2. Kaan Atakan Öztürk - 21302164 - Section 01
3. Gledis Zeneli - 21701065 - Section 01

Backus-Naur Form (BNF)

<program> ::= <stmt_list>

<stmt_list> ::= <empty> | <stmt_list> <stmt>

<stmt> ::= <assign_stmt> | <declare_stmt> | <while_stmt> | <func_call>
| <io_stmt> | <comment> | <if_stmt> | <for_stmt>

<if_stmt> ::= <matched> | <unmatched>

<matched> ::= if (<logic_expr>) {<stmt_list>} else {<stmt_list>}

<unmatched> ::= if (<logic_expr>) {<stmt_list>}

<declare_stmt> ::= <connection_stmt> | <sensor_stmt> | <func_dec> | <actuator_stmt> |
<type> <var_name> | <type> <assign_stmt>

<connection_stmt> ::= <connection> <var_name> (<url>)

<sensor_stmt> ::= <sensor> <var_name> (<conn_param>)

<conn_param> ::= <var_name> <at> (<port_name>)

<actuator_stmt> ::= <actuator> <var_name> (<conn_param>)

<port_name> ::= <unsigned_integer> | <var_name>

<url> ::= <var_name> | URL

<io_stmt> ::= <input_stmt> | <output_stmt>

<input_stmt> ::= read(<var_name>)

<output_stmt> ::= print(<var_name>) | print(<literal>)

<while_stmt> ::= while(<logic_expr>) {<stmt_list>}

<for_stmt> ::= for (<type> <var_name> <assign_op> <literal>, <logic_expr>,
<assign_stmt>) {<stmt_list>}

<func_dec ::= <def> <var_name> (<var_list_i>) {(<stmt_list> return <var_name>)}
| <def> <var_name> (<var_list_i>) {(<stmt_list> return <literal>)}
| <def> <var_name> (<var_list_i>) {(<stmt_list> return)}

<var_list_i> ::= empty | <nonemp_var_list_i>

<nonemp_var_list_i> ::= <type> <var_name> | <connection> <var_name>

| <actuator> <var_name> | <sensor> <var_name>

| <nonemp_var_list_i>, <type> <var_name>

| <nonemp_var_list_i>, <sensor> <var_name>

| <nonemp_var_list_i>, <actuator> <var_name>

| <nonemp_var_list_i>, <connection> <var_name>

<var_list_ii> ::= <empty> | <nonemp_var_list_ii>

<nonemp_var_list_ii> ::= <var_name> | <literal> | <nonemp_var_list_ii>, <var_name>

| <nonemp_var_list_ii>, <literal>

<func_call> ::= <var_name> .<send_data>(<unsigned_integer>)

| <var_name> .<send_data>(SIGNED_INTEGER)

| <var_name> .<get_data>()

| <var_name>(<var_list_ii>)

<assign_stmt> ::= <var_name> <assign_op> <expression>

<expression> ::= <expression> <add_op> <item> | <expression> <sub_op> <item>

| <item>

<item> ::= <item> <mult_op> <basic_item> | <item> <div_op> <basic_item>

| <timestamp> | <basic_item>

<basic_item> ::= <literal> | <var_name> | <func_call> | (<expression>)

<timestamp> ::= TIMESTAMP()

<literal> ::= STRING_LIT | <unsigned_integer> | SIGNED_INTEGER

| BOOL_LIT_TRUE | BOOL_LIT_FALSE | FLOAT_LIT

<at> ::= AT

<unsigned_integer> ::= UNSIGNED_INTEGER

<send_data> ::= SEND_DATA

<get_data> ::= GET_DATA

<sensor> ::= SENSOR LESS_THAN <type> GREATER_THAN

<connection> ::= CONNECTION

<actuator> ::= ACTUATOR

<var_name> ::= VAR

<assign_op> ::= ASSIGN_OP

<add_op> ::= ADD

<def> ::= DEF

<sub_op> ::= SUBTRACT

<div_op> ::= DIVIDE

<mult_op> ::= MULTIPLY

<comment> ::= COMMENT

<type> ::= INT | BOOL | STRING | FLOAT

<empty> ::=

<logic_expr> ::= <expression> <logic_op> <expression>

<logic_op> ::= AND | OR | GREATER_THAN | LESS_THAN | GREATER_EQUAL |

LESS_EQUAL | EQUAL | NOT_EQUAL

LEX

%option yylineno

digit [0-9]

integer {digit}+

letter [a-zA-Z]

alphanum [0-9a-zA-z]*

sign [+|-]

nl \n

ipsub [0-9][0-9]?[0-9]?

ip {ipsub}\.{ipsub}\.{ipsub}\.{ipsub}

```
%%
while                                   return(WHILE);
for                                     return(FOR);
if                                      return(IF);
else                                    return(ELSE);
\.at                                    return(AT);
\.send_data                             return(SEND_DATA);
\.get_data                              return(GET_DATA);
read                                    return(READ_CONSOLE);
print                                   return(PRINT);
return                                  return(RETURN);
bool                                    return(BOOL);
int                                     return(INT);
String                                  return(STRING);
float                                   return(FLOAT);
and                                     return(AND);
or                                      return(OR);
def                                     return(DEF);
True                                    return(BOOL_LIT_TRUE);
False                                   return(BOOL_LIT_FALSE);
{sign}?{integer}*\.{integer}            return(FLOAT_LIT);
\".*\"                                  return(STRING_LIT);
http[s]?\:\/\/www\.{alphanum}\.{letter}+ return(URL);
{ip}                                    return(URL);
{integer}                               return(UNSIGNED_INTEGER);
{sign}{integer}                         return(SIGNED_INTEGER);
Sensor                                  return(SENSOR);
Actuator                                return(ACTUATOR);
Connection                              return(CONNECTION);
Timestamp                               return(TIMESTAMP);
{letter}{alphanum}                      return(VAR);
```

```
\(                                  return(LP);
\)                                  return(RP);
\{                                  return(LB);
\}                                  return(RB);
\,                                  return(COMMA);
\/\/[^\n]*                          return(COMMENT);
\=                                  return(ASSIGN_OP);
\=\=                                return(EQUAL);
\!\=                                return(NOT_EQUAL);
\>\=                                return(GREATER_EQUAL);
\<\=                                return(LESS_EQUAL);
\>                                  return(GREATER_THAN);
\<                                  return(LESS_THAN);
\+                                  return(ADD);
\-                                  return(SUBTRACT);
\*                                  return(MULTIPLY);
\/                                  return(DIVIDE);
[ \t\n]                             ;
.                                   return(yytext[0]);
%%
int yywrap() { return 1; }
```

Definition of BNF Elements and Relation to Language Conventions

I. Non-Terminals

- **<program>:** This is the start symbol and represents the entry point of any program constructed in the language.
- **<stmt_list>:** A sequence of either one or multiple statements that make up the program.
- **<stmt>:** A statement in the language.
- **<matched>:** a matched if statement, the if is paired with an else

- **<unmatched>:** an unmatched if statement, the if is not paired with an else
- **<if_stmt>:** An if statement
- **<assign_stmt>:** the value of a variable is set to the value of expression
- **<declare_stmt>:** a statement for declaring a variable; primitive type variables (int, bool, float) and strings can be initialized at declaration as well
- **<while_stmt>:** The looping structure for the language starting with the reserved word while followed by a logical expression and curly braces. It executes statements enclosed within the curly braces as long as the logical expression is True.
- **<for_stmt>:** Another looping structure starting with the reserved word for that consists of an assignment, a logic expression to evaluate the current status of the assignment variable and an assignment statement to change the value of the variable after each iteration all separated by a comma. It executes the loop while the logic expression is True. This is similar to other languages such as Java.
- **<func_call>:** It defines a call to a function starting with the function name followed by the function parameters (a list of variables) enclosed within ().
- **<connection_stmt>:** a statement which declares a new Connection.
- **<sensor_stmt>:** a statement which declares a new Sensor object
- **<func_dec>:** It defines the structure of a function definition and allows users to define subprograms that may or may not have parameters and return a value consisting of any of the types defined in the language. This enables modularity and abstraction.
- **<actuator_stmt>:** a statement which declares a new Actuator object
- **<type>:** the primitive types the program supports (int, bool, float) and strings
- **<var_name>:** name which refers to a variable
- **<conn_param>:** a Connection instance meant to be given to the Sensor and Actuator classes when created so the object connects to the actual device specified by the <conn_param>
- **<time_stamp>:** a function to return the system time
- **<at>:** a function for Connection objects which takes a port returns the url of the connection with the port appended to its end.
- **<send_data>:** It allows the user to send integers to the connection.
- **<get_data>:** It allows the user to receive integers from the connection.
- **<port_name>:** This is the port name of the local or internet connection.
- **<sensor>:** Class name for the entity which is going to receive data from the sensors
- **<connection>:** Class name for the entity which is going to make the connection to the internet possible
- **<url>:** It represents a url for a connection.
- **<actuator>:** Class name for the entity which is going to flip the switches which control the actuators
- **<io_stmt>:** It represents a statement that can either be read from the console or printed to the console.
- **<input_stmt>:** A statement read from the console.
- **<output_stmt>:** A statement printed to the console.

- **<var_list_i>:** It represents a list of variables acting as the defining parameters of a function.
- **<nonemp_var_list_i>:** It represents a version of <var_list_i> which cannot be empty.
- **<var_list_ii>:** It represents a list of variables passed as parameters when calling a function.
- **<nonemp_var_list_ii>:** It represents a version of <var_list_ii> which cannot be empty.
- **<def>:** This is reserved to define the beginning of functions.
- **<empty>:** Defined by empty spaces.
- **<expression>:** an arithmetic combination of the four basic operators (+, -, *, /), variables, constants, and ().
- **<item>:** a structure meant to increase the precedence of * and /, and make / left associative
- **<basic_item>:** a structure used to increase the precedence of (), or it can just be a variable or a constant
- **<comment>:** This is a comment in the code.

II. Terminals

- **COMMENT:** This terminal specifies a comment which in the language is only a single line comment defined by "//" followed by any text.
- **SENSOR:** This terminal specifies a sensor in the IoT device.
- **CONNECTION:** This terminal specifies the connection to an internet port.
- **ACTUATOR:** This terminal specifies the actuator switches in the IoT network.
- **If:** This terminal represents the start of an if-statement.
- **Else:** This terminal represents the start of an else-statement.
- **While:** This terminal represents the start of a while-statement.
- **RETURN:** This terminal returns the following value as the output of a function.
- **BOOL:** This terminal represents a boolean type variable (can only be True or False).
- **INT:** This terminal represents an integer type (either signed or unsigned) variable.
- **STRING:** This terminal represents a string type variable.
- **FLOAT:** This terminal represents the multiplication (*) operator.
- **UNSIGNED_INTEGER:** This terminal represents an integer without any sign.
- **SIGNED_INTEGER:** This terminal represents a signed (+ or -) integer.
- **STRING_LIT:** This terminal represents a literal string typed within "".
- **FLOAT_LIT:** This terminal represents a literal float typed within the program.
- **BOOL_LIT:** This terminal represents a literal True or False typed within the program.
- **AT:** This terminal specifies the connection method to connect to a port.
- **SEND_DATA:** This terminal represents the method to send integers to the connection.
- **GET_DATA:** This terminal represents the method to read integers from the connection.
- **VAR:** This terminal represents a variable in the language.
- **Read:** This terminal specifies reading input from the console.
- **Print:** This specifies printing output to the console.
- **DEF:** This terminal represents def keyword used to start functions.

- **URL:** This terminal represents a url string.
- **ASSIGN_OP:** This terminal represents the assignment (=) operator.
- **ADD:** This terminal represents the addition (+) operator.
- **SUB:** This terminal represents the subtraction (-) operator.
- **DIV:** This terminal represents the division (\) operator.
- **MULT:** This terminal represents the multiplication (*) operator.
- **AND:** This terminal represents the logical and operator which in the language is “and”.
- **OR:** This terminal represents the logical or operator which in the language is “or”.
- **GREATER_THAN:** This terminal represents the logical > operator.
- **LESS_THAN:** This terminal represents the logical < operator.
- **GREATER_EQUAL:** This terminal represents the logical >= operator.
- **LESS_EQUAL:** This terminal represents the logical <= operator.
- **EQUAL:** This terminal represents the equals (==) operator.
- **NOT_EQUAL:** This terminal represents the not equals (!=) operator.
- **TIMESTAMP:** A function which returns the system time.

The Language

III. About

In IOTA programs have inbuilt functionality to control sensors and actuators by connecting them to either internet or local ports and then relaying actions based on the data received to and from those connections providing swift and efficient IoT functionality. The language is generally modelled on python in that users can write scripts without writing any complicated functions although if the need arises to write enterprise IoT software rather than simple scripts then the language provides the functionality to write functions. It allows users new to IoT programming to easily code and understand programs written in the language since it draws on useful and concise features from other general purpose programming languages.

IOTA is designed with the purpose of providing general purpose IoT support. It supports receiving data from sensors, to which one can connect by supplying an internet or a local address and a port where the specific sensor is. The type of the sensor is not specified in its declaration, but the type of data format to be expected from the sensor must be specified, since there might be a huge array of different sensors to work with, and our language is built to be general. The user of the language can also define switches whose state they can read and change. These switches are managed through by our actuator functionality which you can create by proving an address and a port of the switch, similarly to the sensor declaration. The number of actuators is not set to 10, since the user of the language can declare any number of actuators. This design choice was made to achieve generality.

IV. Readability

The programming language is written in a way that user can understand the syntax easily since it minimizes the overhead associated with writing long function declarations by allowing users to start the function with the keyword “def”, clearly defining user-friendly reserve words and easy to understand conventions for things such as variable names. The language forces the programmer to use curly brackets to enclose multiple statements inside if-statements, functions or loops to allow clear associativity of statements with their respective if or while statements and allows users to easily read nested if or while statements. The users are encouraged to use indentation since the language does not enforce semicolons to mark the end of statements. All operators are directly imported from their mathematical notations which allows easy understanding of their usage and meaning.

V. Writability

IOTA is written in a manner that allows extensive functionality for IoT devices while maintaining the robust and flexible nature of a general purpose programming language. Users do not need to end statements with semicolons. The syntax of the programming language is designed by considering flexibility and ease of use to developers. IOTA requires to be written in small statements which constructs a guideline to provide maintainable code experience from beginners to masters. When the code segments gets smaller, the easier problems will be encountered.

Example Program

```
int port
String url
print("Enter url and sensor port")
read(url)
read(port)

Connection conn(url)
Sensor<float> humiditySens(conn.at(port))

float humidityThreshold
print("Enter humidity threshold")
read(humidityThreshold)
```

```
def triggerActuatorForThreshold( Sensor<float> s, float th, Actuator ac) {(
        int t1 = Timestamp()
        while( s.get_data() < th) {
        // prints humidity every 1 minute
        int t2 = Timestamp()
        if( (t2 - t1) - ((t2 - t1) / 3600000) * 3600000 == 0) {
        print("Humidity in air is: ")
        float f = s.get_data() * 100
        print(f)
        print("%")
        }
}
// actuator triggered
ac.send_data(1)
return
)}

int actuatorPort
print("Enter the port for the humidity sensor actuator")
read(actuatorPort)

Actuator actHum(conn.at(actuatorPort))
triggerActuatorForThreshold(humiditySensor, humidityThreshold, actHum)

if(actHum.get_data() == 0) {
        // the humidity actuator has NOT been activated
        print("Hardware malfunction, the actuator was not activated")
        def tryNewActuator(Connection conn) {(
        print("Provide a new actuator port for backup actuator")
        int backUpPort
```

```
        Actuator backUpActuator(conn.at(backUpPort))
        return backUpActuator
)}

Actuator tryAgain(conn.at(port))
tryAgain.send_data(1)
if(tryAgain.get_data() != 1) {
        print("Hardware Malfunction")
}
else {
        // actuator was successfully turned on
        //wait until humidity goes back to normal
        while(humiditySens.get_data() >= humidityThreshold) {}
// turn off actuator
humAct.send_data(0)
}

print(“Enter the 10 temperature instances as measured through the day: \n”)
float sum = 0.0
for(int i = 0, i < 10, i = i + 1) {
        float temp
        read(temp)
        sum = sum + temp
}

print(“The average of the temperatures you entered: ”)
float avg = sum / 10
print(avg)
```