Bilkent University

Department of Computer Engineering

CS315 Project 1

Spring 2020

Programming Language: SetLab

Group 17

Team Members

Tolga Çatalpınar: 21703611 Section 1

Zeynep Cankara: 21703381, Section 1

Naci Dalkıran: 21601736, Section 1

Instructor: Halil Altay Güvenir

Teaching Assistant(s): Duygu Durmuș, Alper Șahıstan, Furkan Hüseyin

Name of the Language: SetLab

The Complete BNF Description of the SetLab Language

1) Program Definition

<program> ::= < stmt_list>

<stmt_list> ::= <stmt> | <stmt_list> \n <stmt>|

<stmt> ::= <declaration> | <comment_line> | <expr> | <loops> | <funct_dec>

<ident> ::= <non_digit_char> | <digit> |
    <ident><non_digit_char> |
        <ident><digit>

<identifier> ::= <non_digit_char>| <non_digit_char><ident>

<ident_list> ::= <identifier> | <ident_list> , <identifier>

<declaration> ::= <type> <ident_list>

2) Comments

<comment_symbol> ::= #

<comment_line> ::= <comment_symbol><sentence>?<comment_symbol>

3) Types and Constants

<type> ::= **int** | **char** | **boolean | string**

<lowercase_char> ::= a|b| ...| z

<uppercase_char> ::= A|B| ...| Z

<non_digit_char> ::= <lowercase_char> | <uppercase_char>

<digit> ::= 0|1|2|...|9

<non_zero_digit> ::= 1|2| ....|9

<types> ::= <int_type> | <str_type> | <bool_type> | <char_type>

<int_type> ::= <sign>? <number>

<sign> ::= **+** | **-**

<number> ::= <non_zero_digit> | <number> <digit>

<str_type> ::= “<sentence>?”

<sentence> ::= <alpha_num><sentence> | <alpha_num> |<space_character>

<alpha_num> ::= (<non_digit_char> | <digit>) <alpha_num> | (<non_digit_char> | <digit>)

<space_character> :: =

<bool_type> ::= **true | false**

```
<char_type> ::= <lowercase_char> | <uppercase_char> | <digit>
```

3) Type Declerations

```
<expr> ::= <assign_set_expr> | <element_expr> | <int_expr> | <str_expr> | <bool_expr> |
<delete_element_expr> ::= ‘<alpha_num>’
<assign_op> ::= <==
<element_expr> ::= <identifier> <= <alpha_num> | <identifier> <= <identifier>
<int_expr> ::= (int)? <identifier><assign_op>(<int_type> | <identifier> | <funct_call>)
<str_expr> ::= (string)? <identifier><assign_op>(<str_type> | <identifier> | <funct_call>)
<bool_expr> ::= (boolean)? <identifier><assign_op>(<bool_type | <identifier> | <funct_call>)
<char_expr> ::= (char)? <identifier><assign_op>(<char_type | <identifier> | <funct_call>)
```

4) Sets

```
<set_expr_list> :: = <set_expr>| <delete_expr>| <add_expr>|
                     <union_expr>| <intersection_exp>| <subset_expr>|
                      <superset_expr>| <input_expr>| <output_expr>
<set_expr> ::= <identifier> <= <set_init>
<set_init> ::= new Set
<delete_set_expr> ::= delete <identifier>
<add_expr> ::= <identifier>.add(<identifier>) |  <identifier>.add(‘<alpha_num>’)
<union_expr> ::=<identifier>.union(<identifier>)
<intersection_exp> ::= <identifier>.intersection(<identifier>)
<assign_set_expr> ::= (Set)? <identifier><assign_op><identifier> |
                        Set <identifier><assign_op><set_init>
<subset_expr> ::= <identifier>.isSubset (<identifier>)
<superset_expr> :: <identifier>.isSuperset(<identifier>)
```

5) Input/Output

```
<input_expr> ::= <identifier>.inputElements()
<output_expr> ::= <identifier>.print()
```

6) Loops

```
<loops> ::= <while_loop> | <for_loop>
<while_loop> ::= while(<condition>){<stmt_list>}
<condition> ::=<identifier><relation_op><identifier>
<for_loop> ::= for(i = <int_type>:<int_type>){<statement_list>}
```

7) If Statement

<if> ::= if(<condition>){stmt_list}

<condition> ::=<identifier><relation_op><identifier> <condition>?

  | && <identifier><relation_op><identifier>

  | || <identifier><relation_op><identifier>

8) Relational Operators

<relation_op> ::= < | > | <= | >= | == | !=

9) Function Definition and Call

<funct_dec> ::= func <return_type><identifier>(<args>){<stmt_list>}

<return_type> ::= <type>

<funct_call> ::= <identifier>(<args>) | <identifier>.<identifier>(<args>)

<args> := <identifier>? | <identifier>,<args>

Explanation of the SetLab Language Constructions

This non-terminal states that our program consists of statements where statements of our language consists of statement lists.

<program> ::= < stmt_list>

<stmt_list> ::= <stmt> | <stmt_list> \n <stmt>|

This non-terminal describes the statement structure of SetLab which can consist of type declaration, comment lines, assignments, set expressions, loops and function declarations.

<stmt> ::= <declaration> | <comment_line> | <expr> | <loops> | <funct_dec>

<declaration> ::= <type> <ident_list>

This non-terminal describes the supported primitive types by our language.

<type> ::= **int** | **char** | **boolean | string**

Keyword to define comment.

<comment_symbol> ::= #

Definition of a comment statement .

<comment_line> ::= <comment_symbol><sentence>?<comment_symbol>

Assignment Operator.

<assign_op>::= <==

Definition of symbols.
<lowercase_char> ::= a|b| ...| z
<uppercase_char> ::= A|B| ...| Z
<digit> ::= 0|1|2|...|9
<non_zero_digit> ::= 1|2| ....|9
<sign> ::= **+** | **-**

The non-terimnal defined identifiers in SetLab where an identifier must start with a non-digit character and can followed by any digit or character.
<identifier> ::= <non_digit_char>| <non_digit_char><ident>
<ident> ::= <non_digit_char> | <digit> |
<ident><non_digit_char> |
<ident><digit>
<non_digit_char> ::= <lowercase_char> | <uppercase_char>
<ident_list> ::= <identifier> | <ident_list> , <identifier>

Definition of types in SetLab.
<types> ::= <int_type> | <str_type> | <bool_type> | <char_type>
<int_type> ::= <sign>? <number>
<number> ::= <non_zero_digit> | <number> <digit>
<str_type> ::= "<sentence>?"
<sentence> ::= <alpha_num><sentence> | <alpha_num> |<space_character>
<alpha_num> ::= (<non_digit_char> | <digit>) <alpha_num> | (<non_digit_char> | <digit>)
<space_character> :: =
<bool_type> ::= **true | false**
<char_type> ::= <lowercase_char> | <uppercase_char> | <digit>

<element_expr> ::= <identifier> <= <alpha_num> | <identifier> <= <identifier>
<int_expr> ::= (int)? <identifier><assign_op>(<int_type> | <identifier> | <funct_call>)
<str_expr> ::= (string)? <identifier><assign_op>(<str_type> | <identifier> | <funct_call>)
<bool_expr> ::= (boolean)? <identifier><assign_op>(<bool_type | <identifier> | <funct_call>)
<char_expr> ::= (char)? <identifier><assign_op>(<char_type | <identifier> | <funct_call>)

This non-terminal describes the group of set operations defined in the language. The set operations are: initializing a set, deleting a set, set addition operation, union operation, intersection operation, subset checking, superset checking, input and ouput expressions.

```
<expr> ::= <assign_set_expr> | <element_expr> | <int_expr> | <str_expr> | <bool_expr> |
<delete_element_expr> ::= '<alpha_num>'
<set_expr_list> :: = <set_expr>| <delete_expr>| <add_expr>|
                     <union_expr>| <intersection_exp>| <subset_expr>|
                      <superset_expr>| <input_expr>| <output_expr>
<set_expr> ::= <identifier> <= <set_init>
<set_init> ::= new Set
<delete_set_expr> ::= delete <identifier>
<add_expr> ::= <identifier>.add(<identifier>) |  <identifier>.add('<alpha_num>')
<union_expr> ::=<identifier>.union(<identifier>)
<intersection_exp> ::= <identifier>.intersection(<identifier>)
<assign_set_expr> ::= (Set)? <identifier><assign_op><identifier> |
                         Set <identifier><assign_op><set_init>
<subset_expr> ::= <identifier>.isSubset (<identifier>)
<superset_expr> :: <identifier>.isSuperset(<identifier>)
<input_expr> ::= <identifier>.inputElements()
<output_expr> ::= <identifier>.print()
```

This non-terminal describes the loop constructs defined in SetLab language.

```
<loops> ::= <while_loop> | <for_loop>
```

While loops in SetLab consists of a condition statement with a statement list body .

```
<while_loop> ::= while(<condition>){<stmt_list>}
```

This non-terminal defines the condition structure in SetLab which consists of relational operator comparisons.

```
<condition> ::=<identifier><relation_op><identifier> <condition>?
        |  && <identifier><relation_op><identifier>
        | || <identifier><relation_op><identifier>
```

Definition of relational operators in SetLab.

```
<relation_op> ::= < | > | <= | >= | == | !=
```

For loops in SetLab automatically initializes a variable to the start boundary and iterates through the end boundary, executing the instructions defined as statement list within each iteration.

```
<for_loop> ::= for(i = <int_type>:<int_type>){<statement_list>}
```

This non-terminal defines functions in SetLab where function definitions must start with reserved keyword ‘func’ followed by the function name and arguments. The function executes the statement list.
<funct_dec> ::= func <return_type><identifier>(<args>){<stmt_list>}
<return_type> ::= <type>

This non-terminal states function calls in SetLab where function call done by calling the function with arguments or accsessing an identifier’s method via dot notation and then passing the arguments to the function.
<funct_call> ::= <identifier>(<args>) | <identifier>.<identifier>(<args>)

An argument is an optional identifier or identifier list where identifiers are seperated by comma.
<args> := <identifier>? | <identifier>,<args>

This non-terminal describes the if conditional in SetLab.
<if> ::= if(<condition>){stmt_list}

Descriptions of SetLab Non-Trivial Tokens

- **SET_INIT:** Special keyword for set initialization.
- **SET:** Token reserved for the set type.
- **INTEGER:** Token reserved for integer type.
- **BOOLEAN:** Token reserved for boolean type.
- **ELEMENT**: Token reserved for data types that can be added to a set.
- **IDENTIFIER:** Token reserved for variables.
- **COMMENT_KEY:** Token reserved for comment keyword detection.
- **COMMENT:** Reserved for comment statements.
- **SET_CREATE:** Reserved for set initialization statements.
- **SET_DELETE:** Reserved for set deleting statements.

- **ADD:** Reserved for adding items to a set.
- **UNION:** Reserved for finding union between two sets statement.
- **INTERSECTION:** Reserved for finding intersection between two sets statement.
- **SUBSET_CHECK:** Reserved for checking a set is subset of an another set statement.
- **SUPERSET_CHECK:** Reserved for checking a set is superset of an another set statement.
- **READ_INPUT:** Token reserved for reading console input statement.
- **OUTPUT:** Token reserved for outputting to the console.
- **ASSIGN_OP:** Token reserved for assignment operation.
- **RELATION_OP:** Token reserved for relational operations.
- **WHILE_LOOP:** Token reserved for detection of a while loop.
- **FOR_LOOP:** Token reserved for detection of a for loop.
- **FUNC_DEF:** Token reserved for function definiton.
- **FUNC_CALL:** Token reserved for function calls.
- **NL:** Token reserved for new line.

SetLab Evaluation

A. Readability:

SetLab language is designed to reduce the syntax necessities. For instance, there are no type declarations. Therefore, programmers do not have to memorize the types. Also, there is no need for semicolon at the end of the lines. In those aspects, SetLab satisfies python’s readability. However, in contrast to python, SetLab does not require indentation in its syntax. That is a disadvantage in the sense of readability. Nonetheless, thanks to the bracket requirements in the loops and functions, SetLab compensate that disadvantage.

B. Writability:

SetLab gains writability at the expense of its loss in readability, that is, programmers do not have to worry about indentation and can write their code line by line without using any character at the end of the statement. Then, it is required to use brackets in the loops and functions, and parentheses in the conditions. This is a loss of writability in the language.

C. Reliability:

Since SetLab has no type declaration and type checking mechanism, somewhere in the code, programmers might encounter a problem which can be very hard to notice. Then, sets include both alphanumeric and integer characters, operation on the elements may give different, or sometimes unexpected, results.

SetLab Example Programs

Program 1:

```
#Create Set #
set1 <== new Set
#Adding element to Set#
set1.add(Zeynep)
set1.add(Tolga)
set1.add(Naci)
# Create other set #
set2 <== new Set
# Adding element to Set #
set1.add(JF)
set1.add(Kennedy)
set1.add(government)
#Union #
set3 <== set1.union(set2)
#Intersection#
set4 <== set1.intersection(set2)
#Set Relations#
```

```
bool1 <== set3.isSubset(set1)   #False#
bool2 <== set2.isSubset(set3)   #True#
bool3 <== set3.isSuperset(set2) #True#
bool4 <== set4.isSuperset(set2) #False#
```

Program 2:

```
#Create Set #
set1 <== new Set
firstCount <== 5;
secondCount <== 7;
thirdCount <== 9;
#Adding element to Set#
set1.add(firstCount)
set1.add(secondCount)
set1.add(thirdCount)
# Print set #
set1.print()
# Create other set #
set2 <== new Set
# take Input from Console#
set2 <== inputElements()
# Print set #
set2.print()
#Intersection#
set3 <== set1.intersection(set2)
# Print set #
set3.print()
check <== set3.isSubset(set1)
while(check){pass}
```

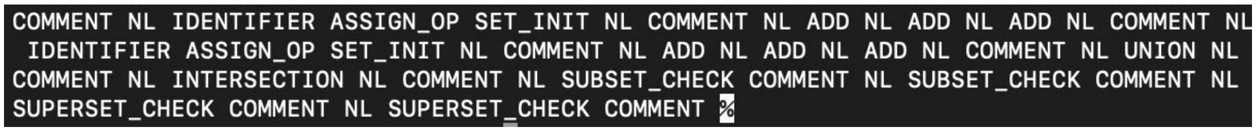
```
delete set1
delete set2
delete set3
```

Program 3:

```
#Create Set #
firstGrade <== new Set
secondGrade <== new Set
thirdGrade <== new Set
student1 <== 'kerim';
student2 <== 'gamze';
student3 <== 'hakan';
#Adding element to Set#
firstGrade.add(student1)
firstGrade.add('emin')
firstGrade.add('Vehbi')
secondGrade.add(student2)
secondGrade.add(student3)
secondGrade.add('ali')
# Print set #
firstGrade.print()
secondGrade.print()
for(i=1:3){pass}
func graduate(grade){pass}
func_name(firstGrade)
delete firstGrade
```

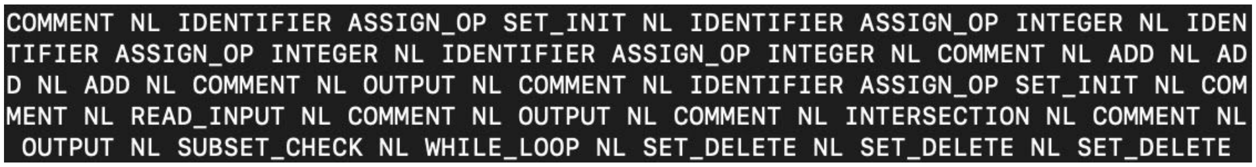
SetLab Example Program Outputs

Program 1:



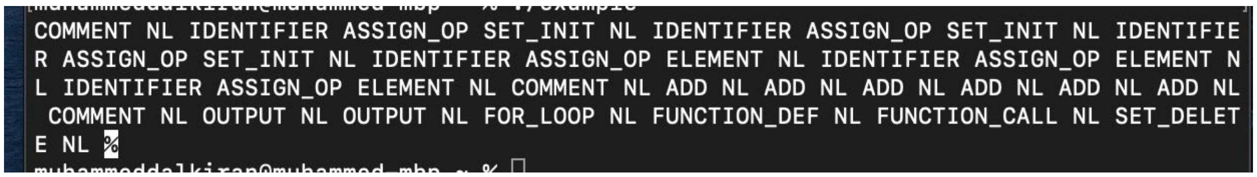
```
COMMENT NL IDENTIFIER ASSIGN_OP SET_INIT NL COMMENT NL ADD NL ADD NL ADD NL COMMENT NL
 IDENTIFIER ASSIGN_OP SET_INIT NL COMMENT NL ADD NL ADD NL ADD NL COMMENT NL UNION NL
COMMENT NL INTERSECTION NL COMMENT NL SUBSET_CHECK COMMENT NL SUBSET_CHECK COMMENT NL
SUPERSET_CHECK COMMENT NL SUPERSET_CHECK COMMENT %
```

Program 2:



```
COMMENT NL IDENTIFIER ASSIGN_OP SET_INIT NL IDENTIFIER ASSIGN_OP INTEGER NL IDEN
TIFIER ASSIGN_OP INTEGER NL IDENTIFIER ASSIGN_OP INTEGER NL COMMENT NL ADD NL AD
D NL ADD NL COMMENT NL OUTPUT NL COMMENT NL IDENTIFIER ASSIGN_OP SET_INIT NL COM
MENT NL READ_INPUT NL COMMENT NL OUTPUT NL COMMENT NL INTERSECTION NL COMMENT NL
 OUTPUT NL SUBSET_CHECK NL WHILE_LOOP NL SET_DELETE NL SET_DELETE NL SET_DELETE
```

Program 3:



```
COMMENT NL IDENTIFIER ASSIGN_OP SET_INIT NL IDENTIFIER ASSIGN_OP SET_INIT NL IDENTIFIE
R ASSIGN_OP SET_INIT NL IDENTIFIER ASSIGN_OP ELEMENT NL IDENTIFIER ASSIGN_OP ELEMENT N
L IDENTIFIER ASSIGN_OP ELEMENT NL COMMENT NL ADD NL ADD NL ADD NL ADD NL ADD NL ADD NL
 COMMENT NL OUTPUT NL OUTPUT NL FOR_LOOP NL FUNCTION_DEF NL FUNCTION_CALL NL SET_DELET
E NL %
```