Bilkent University
Department of Computer Engineering

CS315 Project 2

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 Durmuş, Alper Şahıstan, Furkan Hüseyin

Name of the Language: SetLab

BNF Description of the SetLab Language (final)

1) Types and Constants

<underscore> ::= “_”
<assign_op>::= “<==”
<LB> ::= “{“
<RP> ::= “}”
<LP> ::= “(”
<RP> ::= “)”
<lowercase_char> ::= a|b| …| z
<uppercase_char> ::= A|B| …| Z
<non_digit_char> ::= <underscore> | <lowercase_char> | <uppercase_char>
<digit> ::= 0|1|2|…|9
<non_zero_digit> ::= 1|2| ….|9
<comment_sign> ::= “#”
<set_token> ::= “$”
<new_keyword> ::= “new”
<set_keyword> ::= “Set”
<end_stmt> ::= “;”
<integer> ::= <sign>? <number>
<sign> ::= + | -
<number> ::= <non_zero_digit> | <number> <digit>
<alphanumeric> ::= (<non_digit_char> | <digit>) <alpha_num> | (<non_digit_char> | <digit>)
<identifier> ::= (<lowercase_char> | <uppercase_char>)(<non_digit_char> | <digit>)
<space_character> :: =
<bool_type> ::= **true | false**
<set_type> ::= <set_token><identifier>
<element> ::= ‘<identifier>’

2) Program Definition

<program> ::= < main>
<main>::= <LP><RP><LB><statements><RB>

<statements> ::=<statement>|<statements><statement>
<statement> ::=<comment_line>|<expr><end_stmt>|<loops>|<funct_dec>|<conditional_stmt>
<sentence> ::= <identifier><sentence>|<identifier>
<comment_line> ::= <comment_sign><sentence><comment_sign>
<expr>::=<element_expr>|<int_expr>|<bool_expr>|<set_expr_list>|<set_initialize>|<func_call_dec>|<identifier_dec>

3) Blocks and Args
<args_type> ::= <identifier>|<bool_type>|<integer>|<element>|<set_type>
<block_stmts> ::= pass <end_stmt>|<statements> return <args_type>|<statements>

4) Loops
<loops> ::= <while_stmt>|<for_stmt>
<while_stmt> ::=
while<LP>(<logical_expr>|<set_logical_expr>|<funct_call>)<RP><LB><block_stmts><RB>
<for_stmt> ::= for<LP><for_expr><RP><LB><block_stmts><RB>

5) Conditionals
<conditional_stmt> ::= <if_stmt>
<if_stmt>::=
if<LP>(<logical_expr>|<set_logical_expr>|<funct_call)<RP><LB><block_stmts><RB><else_stmt>
<else_stmt>::=else<LB><block_stmts><RB>

6) Functions
<composite_args> ::= <args_type>, <composite_args>|<args_type>
<args> ::= <identifier>|<composite_args>|
<funct_dec> ::= func <identifier> <LP><args><RP><LB><block_stmts><RB>

<funct_call_dec> ::= (<identifier>|<set_type>)<assign_op><funct_call>|<funct_call>
<funct_call> ::= <identifier><LP><args><RP>

7) **Expressions**
<and> ::= &&
<or> ::= ||
<equal> ::= ==
<not_equal> ::= !=
<logical_expr> ::= <integer> < <integer>|<integer> > <integer>| <integer> >= <integer> |
<integer> >= <integer> | <identifier> < <identifier> | <identifier> > <identifier> | <identifirt> <=

<identifier> >= <identifier> | <identifier> <and> <identifier> | <identifier><or>
<identifier> | <bool_type> <and> <bool_type> | <bool_type><or><bool_type>|
<bool_type><equal><bool_type> | <bool_type><not_equal><bool_type>|<identifier><equal><i
dentifier>|<identifier><not_equal><identifier>
<set_expr_list> ::= <set_delete_op>|<set_add_op>|<input_element_expr>|<output_expr>
<set_expr> ::=
<set_init>|<set_union_op>|<set_intersection_op>|<cartesian_expr>|<input_set_expr>
<int_expr> ::=<identifier><assign_op><integer>
<bool_expr> ::=<identifier><assign_op>(<bool_type>|<set_logical_expr>)
<set_logical_expr> ::= <set_contain_expr>|<superset_expr>|<subset_expr>
<set_contain_expr> ::=
<set_type>.contain<LP>(<identifier>|<element>|<integer>|<set_type>)<RP>
<superset_expr> ::= <set_type>.isSuperset<LP><set_type><RP>
<subset_expr> ::= <set_type>.isSubset<LP><set_type><RP>
<element_expr> ::= <identifier><assign_op><element>

8) Sets
<set_delete_op> ::= <set_type>.delete()
<set_add_op>::= <set_type>.add(<identifier>)|
<set_type>.add(<element>)|<set_type>.add(<integer>)|<set_type>.add(<set_type>)
<input_element_expr> ::=<set_type>.input()
<output_expr> ::=<set_type>.print()
<set_initialize> ::= <set_type><assign_op><set_expr>
<set_init>::= new Set
<set_union_op>::= <set_type>.union(<set_type>)
<set_intersection_op> ::=<set_type>.intersection(<set_type>)
<cartesian_expr> ::=<set_type>.cart(<set_type>)
<input_set_expr> ::= inputElements()

Explanation of the SetLab Language Constructions

Types

SetLab has four inbuilt types: **Integer**, **boolean**, **element** and **set**. Sets are an abstraction of mathematical sets. They have integers, other sets or elements. Elements are any symbol that a mathematical set may possess.

Special signs

- **<assign_op> ::= "<=="**

  In this language, this is the operator where initializations are done.
- **<set_token> ::= "$"**

  This sign is put before the identifiers of sets. The reason why we made this token required is to distinguish sets from other variables.
- **<comment_sign> ::= "#"**

  In Setlab, this sign indicates a comment line.
- **<end_stmt> ::= ";"**

  As Java, Setlab requires semicolon at the end of the statements.
- **<new_keyword> ::= "new"**

  In our grammar, "new" is used when sets are initialized.
- **<set_keyword> ::= "Set"**

  While initializing a set, this keyword is used right after the <new_keyword>

Program Structure

- **<program> ::= < main>**

In SetLab language, to have a valid SetLab code, the program is constructed on "main" components. This provides to start the program.

- **<main>::= <LP><RP><LB><statements><RB>**

Main is starting the program and also it includes statements.

- **<statements> ::=<statement>|<statements><statement>**

Statements are to indicate the statement in main and block statements.

- **<statement> ::=<comment_line>|<expr><end_stmt>|<loops>|<funct_dec>|<conditional_stmt>**

Statement includes comment line, expressions, loops, function declaration, conditional statement since these rules are used and basis for constructing the code between brackets.

Comment Line

- **<comment_line> ::= <comment_sign><sentence><comment_sign>**

Comment line one of the components of the statement. Comment line provides user opportunities to make comments to explain their code blocks. Program defines comment line with "#" character.

- **<sentence> ::= <identifier><sentence>|<identifier>**

Comment lines are composed of sentence/s between two "#".characterç

Expressions

- **<expr>::=<element_expr>|<int_expr>|<bool_expr>|<set_expr_list>|<set_initialize>|<func_call_dec>|<identifier_dec>**

Expressions are one of the components of the statement. It includes some expressions and declarations.

- **<set_expr_list> ::= <set_delete_op>|<set_add_op>|<input_element_expr>|<output_expr>**
- **<set_expr> ::= <set_init>|<set_union_op>|<set_intersection_op>|<cartesian_expr>|<input_set_expr>**
- **<set_logical_expr> ::= <set_contain_expr>|<superset_expr>|<subset_expr>**

Set expression list, set expressions and set logical expressions demonstrate set operations and set initialization.

Set expression list is composed of operations which return no values.

Set expressions are composed of operations which are set initialization and returns set value.

Set logical expressions are composed of operations which return boolean value.

- **<int_expr> ::=<identifier><assign_op><integer>**
- **<bool_expr> ::=<identifier><assign_op>(<bool_type>|<set_logical_expr>)**
- **<element_expr> ::= <identifier><assign_op><element>**

The program has three types of variables which are integer, boolean, element.

Integer is composed of positive or negative numbers. And it can be initialized with given integer number.

Boolean is composed of true or false value. And it can be initialized with a given boolean value or return value of set logical expressions methods.

Element is one element of sets.

- **<set_contain_expr> ::= <set_type>.contain<LP>(<identifier>|<element>|<integer>|<set_type>)<RP>**
- **<superset_expr> ::= <set_type>.isSuperset<LP><set_type><RP>**
- **<subset_expr> ::= <set_type>.isSubset<LP><set_type><RP>**

These methods return boolean value.

Contains method checks whether a set contains a given value which can be a defined identifier, element, integer or set.

Superset method checks whether a given set is a superset of the current set or not.

Subset method checks whether a given set is a subset of the current set or not.

Blocks/args

- **<args_type> ::= <identifier>|<bool_type>|<integer>|<element>|<set_type>**

Args is used in function call and function declaration. Type of args can be integer boolean elements or set.

- **<block_stmts> ::= pass <end_stmt>|<statements> return <args_type>|<statements>**

Block statements are used in functions declaration, loops and conditionals to allow the fill of brackets with statement. Functions might return variables therefore return is used and also functions, loops and conditionals needs to pass since if the user wants to write nothing, he/she can use it.

Loops

- **<loops> ::= <while_stmt>|<for_stmt>**

Loops are categorized as while and for.

- **while<LP>(<logical_expr>|<set_logical_expr>|<funct_call>)<RP><LB><block_stmts><RB>**

While statement is a loop and it takes set logical expressions, logical expressions and function calls as conditions to make the loop continue. Logical expressions indicate boolean value and function call and set logical expressions returns a boolean value and its value is used as condition.

- **<for_stmt> ::= for<LP><for_expr><RP><LB><block_stmts><RB>**

For statement is a loop and it takes an int type identifier and its initial and final value to make the loop continue. If the identifier value reaches the final value, the loop stops.

Conditionals

- **<conditional_stmt> ::= <if_stmt>**
- **if<LP>(<logical_expr>|<set_logical_expr>|<funct_call)<RP><LB><block_stmts><RB><else_stmt>**
- **<else_stmt>::=else<LB><block_stmts><RB>**

Conditionals are used to define a condition. It has two parts if and else part. If the given condition is valid program executes statements between if's brackets otherwise else's.

Functions

Functions can be used in the program when declaring and calling it.

- **<funct_dec> ::= func <identifier> <LP><args><RP><LB><block_stmts><RB>**
- **<args> ::= <identifier>|<composite_args>|**
- **<composite_args> ::= <args_type>, <composite_args>|<args_type>**

During function declaration, func is a keyword to understand that it is function declaration and it is followed by identifier, namely its name. After its name its argument/s should be defined. Argument part can include one argument or many arguments.

- **<funct_call_dec> ::= (<identifier>|<set_type>)<assign_op><funct_call>|<funct_call>**
- **<funct_call> ::= <identifier><LP><args><RP>**

The functions of the SetLab language can return an identifier, integer, element, set and our language allows return values to be assigned to either a set type or an identifier. Additionally, one can call the function without assigning the returning value to an identifier or a set type.

Sets

- **<set_delete_op> ::= <set_type>.delete()**

When the programmer decides that he/she is done with a set, he/she may delete the set by using this format.

- **<set_add_op>::= <set_type>.add(<identifier>)|**
  <set_type>.add(<element>)|<set_type>.add(<integer>)|<set_type>.add(<set_type>)

Programmers can make additions to their sets through the add keyword. They may add an element, integer, another set, or indirectly, add an identifier.

- **<input_element_expr> ::=<set_type>.input()**

The input keyword is used for programmer to add input elements

- **<output_expr> ::=<set_type>.print()**

The print keyword is used to print the contents of any set.

- **<set_initialize> ::= <set_type><assign_op><set_expr>**

Sets are initialized via several ways that are in <set_expr>.

- **<set_init>::= new Set**

While initializing a set from scratch, this combination of keywords are used. That way, the grammar distinguishes set initialization from others.

- **<set_union_op>::= <set_type>.union(<set_type>)**

The union keyword is used to indicate the union operation in mathematics. It simply merges two sets and returns the resulting set. This set should be taken from another set identifier.

- **<set_intersection_op> ::=<set_type>.intersection(<set_type>)**

The intersection keyword is used to demonstrate intersection operation in mathematics. It takes the elements of two sets that are common, and then, return the resulting set. Again, this resulting set also shall be taken by another set identifier.

- **<cartesian_expr> ::=<set_type>.cart(<set_type>)**

The cart keyword represents the mathematical cartesian operation. As a result of operation, one set which is a cartesian product of current and given sets is produced and returned.

- **<input_set_expr> ::= inputElements()**

The input elements operation provides users with opportunities to enter an input one element into the current set.

Descriptions of SetLab Non-Trivial Tokens

- **MAIN:** Token to start program.
- **RETURN:** Token to return values within a function.
- **PASS:** Token for empty blocks.
- **IF:** Token reserved for conditional if-else statements.
- **ELSE**: Token reserved for conditional if-else statements.
- **IDENTIFIER:** Token reserved for variables.
- **INTEGER:** Token reserved for integer type.
- **SET:** Token reserved for Set type initialization.
- **SET_TYPE**: Token reserved for set type variables.

- **BOOLEAN:** Token reserved for boolean data type.
- **COMMENT_SIGN:** Token reserved for comment keyword detection.
- **FUNCTION:** Token reserved for function declaration.
- **ASSIGN_OP:** Token reserved for variable assignments.
- **DELETE:** Token reserved for set deleting statements.
- **ADD:** Token reserved for adding items to a set.
- **UNION:** Token reserved for finding union between two sets statement.
- **INTERSECTION:** Reserved for finding intersection between two sets statement.
- **IS_SUBSET:** Token reserved for checking a set is a subset of another set statement.
- **IS_SUPERSET:** Token reserved for checking a set is superset of another set statement.
- **CARTESIAN:** Token reserved for cartesian operation.
- **CONTAIN:** Token reserved for checking whether an item contained within the set.
- **WHILE:** Token reserved for detection of a while loop.
- **FOR:** Token reserved for detection of a for loop.
- **CONSOLE_OUT:** Token reserved for printing contents of set to the console.
- **CONSOLE_IN:** Token reserved for reading from an input stream.
- **ELEMENT_IN:** Token reserved feeding items in the set.
- **END_STMT:** Token reserved for detecting the end of a statement.

How Conflicts Being Resolved

When testing the yacc and lex file together, initially 15 shift/reduce and 9 reduce/reduce conflicts were encountered. While initializing boolean type, integer type and element type, the parser couldn't differentiate between those three types resulting in a reduce/reduce conflict. We incorporated tokens to differentiate between those types to resolve the conflicts. Additionally to fix these conflicts, a new rule created and identifier initialization was moved into this rule. Also,

semi-column is used to indicate the end of an expression. Firstly, semi-column token was added into each expression's rule; however, it caused shift/reduce conflicts. To fix these conflicts, instead of adding semi-column token into each expression's rule, semi-column token (end statement) was added into only expression rule which is the main rule of every expression. Finally, one reduce/reduce conflict was not fixed since although rules were examined a considerable number of times, the reason for this conflict was not found.

How Precedence and Ambiguity Handled in SetLab

The precedence rules are implemented according to the BNF structure in our Yacc file. The parenthesis have the highest precedence followed by the relational and logical operators. In order to solve ambiguities we break down the complex expression into simple expression rules to deal with complexity of our language. We examined the order of the Yacc file to eliminate ambiguities of out SetLab language. We placed flags to detect conflicts and followed through the errors generated by parser to resolve ambiguities presented by our language. Additionally the errors generated by the parser helped us to refine our language.